WSMO Deliverable
D38v0.1
MICROWSMO:
SEMANTIC DESCRIPTION
OF RESTFUL SERVICES

WSMO Working Draft – 19th February 2008

Authors:
Jacek Kopecký
Tomas Vitvar
Dieter Fensel

Editors:
Jacek Kopecký
Tomas Vitvar

This version:
http://www.wsmo.org/TR/d38/v0.1/20080219/

Latest version:
http://www.wsmo.org/TR/d38/v0.1/

Previous version:
none

Copyright © 2008 DERI ®. All Rights Reserved.
DERI liability, trademark, document use, and software licensing rules apply.
Abstract

This deliverable describes MicroWSMO, a service ontology and a way of annotating RESTful Web services with this ontology in order to achieve Semantic Web Service automation.
## Contents

1. Introduction  
2. RESTful Web services  
3. MicroWSMO semantic descriptions  
4. Semantic annotations for RESTful services  
5. Conclusions and Future Work
1 Introduction

Internet-scale distributed computing requires an interoperable set of technologies for communication. There are currently two major alternative directions in these technologies, often named “WS-*” and “REST”. The WS-* set of specifications (based on SOAP [5] and WSDL [8]) uses the messaging paradigm and specialized service interfaces, with standardized infrastructure protocols (e.g. for security, transactions etc.). The REST direction relies on the architectural style of the World Wide Web (known as Representational State Transfer — REST [1]) and it views Web services as sets of resources accessible through the uniform interface of HTTP.

The public Web is an increasingly large repository of RESTful services, while the WS-* technologies are mostly deployed within enterprises (and behind firewalls). The open nature of the public Web, together with the architectural constraints of REST, encourages serendipitous reuse, as discussed in [6].

Up to now, research on Semantic Web Service (SWS) has focused on Web services in the WS-* style, which is similar to preceding distributed computing technologies often used in AI distributed agent scenarios. Technologies such as WSMO [3] and its light-weight version WSMO-Lite [7] define semantic models for Web services and attach SWS descriptions to WSDL.

In this deliverable, we investigate ways of semantically annotating RESTful Web services in order to be able to increase the automation of service discovery, composition and use. Our work is related to the work of Sheth et al. on semantic mashups (“smashups,” [4]). Bringing semantic descriptions to RESTful Web services should simplify planned reuse and further help serendipity.

The main contribution of our work, called MicroWSMO, consists of a service ontology for RESTful Web services, and a method for annotating descriptions of RESTful services in a machine-processable way so that their use can be automated.

todo describe the structure of the deliverable

2 RESTful Web services

The REST architectural style specifies a number of constraints on the architecture of the World Wide Web, which in effect guarantee certain desirable properties, such as cacheability, loose coupling, evolvability etc. However, REST only talks about Web resources, and not about Web services.

So far, there is no clear agreed model of what constitutes a single RESTful Web service. For the purposes of this work, we will define a Web service as a set of related and interlinked Web resources\(^1\). While a single resource can provide any number of distinct operations, it is usually beneficial for a Web service to split its functionality into multiple resources, and then hyperlinks guide the execution of the service.

One major constraint in the REST architectural style is the uniform interface: all resources on the Web share the same limited set of operations, which balances interface expressivity (the number of different operations) with reusability across many applications. The four operations implemented in HTTP, and called methods, are GET, POST, PUT and DELETE\(^2\), with their availability subject to access control and the implementation of the resource.

**GET** returns a representation of the resource (a pure data retrieval operation)

**POST** accepts data for any kind of processing, e.g. creation or manipulation of other resources

---

\(^1\)Unrelated resources have no business being together part of a single Web service, and REST mandates that hypermedia is the engine of application state, so resources need to be interlinked.

\(^2\)These four methods are specific to HTTP, a particular RESTful system, but they are not mandated by REST itself.
Figure 1: The interface of a Web resource

**PUT** replaces the content of the resource with the provided data

**DELETE** removes the resource

While the methods **GET**, **PUT** and **DELETE** have clearly defined function, **POST** is a general-purpose method which can implement any kind of functionality necessary in a distributed system. This differentiates the HTTP interface from the common CRUD set of operations (Create, Retrieve, Update, Delete), which cannot accommodate arbitrary functionality. The general-purpose nature of the POST method implies the need for semantic annotations, if we want an automated client to know what the operation will do.

Figure 1 shows a Web resource with its interface: all methods return a status code, indicating the success or failure of the operation; the methods **GET** and **DELETE** take no inputs; **PUT** receives the content of the resource and **GET** returns it, and **POST** takes some input and returns some output, both generally different from the content of the resource.

Resources are identified by URIs. Some resources are long-lived singletons with well-known URIs (i.e. discovered by means external to the Web service’s own hyperlinks), whereas others are dynamically-created and possibly short-lived. For illustration, in a search engine, the main page is the long-lived singleton, and the form serves as a link to the short-lived, dynamically created result listing resources. In a publishing system such as AtomPub\(^3\), the main collection (e.g. a blog) is the long-lived singleton: it manages the various entries and, on demand, provides their links to the clients.

As already mentioned, the use of RESTful services is guided by the hyperlinks between the involved resources — hypermedia is the engine of the application state\(^4\). The hyperlinks can come in a number of forms:

1. a direct link to some given resource (such as `<a href="...">` in HTML, or similar references in a Service Description Document in AtomPub etc.),
2. an HTML `<form method="get">` which represents a parametrized URI,
3. a textual description of how to create resource links (e.g. Flickr API\(^5\)),
4. a Web application description in W ADL [2].

These are only a few most commonly used options. There is no widely accepted description language for RESTful Web services, which would fulfill the role that WSDL plays within WS-*\(^6\); WADL is a proposed language that has yet to see significant adoption. In fact, there is an ongoing debate on whether or not such a description language is even necessary on the Web. Nevertheless, in order to automate the use of Web services, we need a machine-readable way of attaching semantic annotations to Web services.

---


\(^4\) See an article which discusses this REST constraint at [http://www.infoq.com/articles/mark-baker-hypermedia](http://www.infoq.com/articles/mark-baker-hypermedia)

\(^5\) [http://www.flickr.com/services/api/](http://www.flickr.com/services/api/)
Finally, a major part of the uniform interface constraint of REST is the understanding that the client and the resource should agree on data formats. Semantic mediation is expected to help resolve conflicts when heterogeneities are encountered, therefore semantic annotations are also necessary on the inputs and outputs or Web resources, not only on the meaning of the POST method, as discussed above.

3 MicroWSMO semantic descriptions

In WSMO-Lite (WSMO deliverable D11v0.2) we decompose a service contract into the following components:

- **Information Model** defines the data model for input, output and fault messages.
- **Functional Descriptions** define service functionality, that is, what a service can offer to its clients when it is invoked.
- **Non-Functional Descriptions** define any incidental details specific to a service provider, or the service implementation or its running environment.
- **Behavioral Descriptions** define external (public choreography) and internal (private workflow) behavior.
- **Technical Descriptions** define messaging details, such as message serializations, communication protocols, and physical service access points.

This decomposition is also valid with respect to RESTful Web services, with several minor changes:

- Information model applies to the content of the resource, which can be retrieved with GET and replaced with PUT, and to the inputs and outputs of POST.
- Functional descriptions deal either with the whole Web service, or with the functionality of the POST method on the resources that allow it.
- Behavioral descriptions incorporate the hypermedia structure of the Web service.

Naturally, technical descriptions differ the most between RESTful Web services and WS-*; in fact, since the Web is based on the HTTP standard, many of the messaging details (e.g. message exchange patterns, the communication protocol) are fixed. The client only needs to know the URI of the resource (the physical access point), and potentially the expected data format — the Web has a built-in content negotiation protocol, though, which in some situations enables dynamic agreement on the data formats.

Figure 2 shows the model of a RESTful Web service as a set of hyperlinked resources. The left-hand side of the non-semantic (lower) level contains the data descriptions for the resource contents, and the inputs and outputs of the POST methods, as applicable. The links to the semantic level indicate the following semantic annotations:

- **A1**: the data formats are linked to the respective ontology elements to indicate the meaning of the data,
- **A2**: the data formats are additionally linked to data grounding transformations for lifting or lowering the data to and from a semantic representation,
- **A3**: the Web service is annotated with a high-level description of its functionality (for discovery purposes),
- **A4**: any resources with active POST method are annotated with a description of its functionality on the given resource,
A5: the Web service is annotated with any appropriate non-functional properties.

For expressing concrete semantics, MicroWSMO adopts the WSMO-Lite service ontology, shown here in Listing 1.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix wl: <http://www.wsmo.org/ns/wsmo-lite#> .

wl:Ontology rdf:type rdfs:Class;
  rdfs:subClassOf owl:Ontology.
wl:ClassificationRoot rdfs:subClassOf rdfs:Class.
wl:NonFunctionalParameter rdf:type rdfs:Class.
wl:Condition rdfs:subClassOf wl:Axiom.
wl:Effect rdfs:subClassOf wl:Axiom.
wl:Axiom rdf:type rdfs:Class.
```

Listing 1: WSMO-Lite Service Ontology, adopted for MicroWSMO

The following section describes concrete mechanisms for attaching semantic annotations to RESTful Web service hyperlinks and descriptions.

## 4 Semantic annotations for RESTful services

todo how to attach the service ontology to the restful service descriptions
todo SA-REST, WADL+S, HTML forms+S, AtomPub service description documentation+S?
todo self-describing: the resource, upon GET, returns its own description (or a link to it); RDDL, GRDDL

## 5 Conclusions and Future Work

todo
References


