Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D11.2.2
Report on Aligning WSMO With Other Approaches W3C Working Group

Activity:	4: Exploitation and Impact	
Work Package:	11: Standardization	
Due Date:		31/08/2010 (M30)
Submission Date:		31/08/2010
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible for Deliverable:		UIBK
Revision:		1.0
Author(s):	Reto Krummenacher (UIBK), Jacek Kopecký (OU), Dave Lambert (OU), Daniel Winkler (UIBK), Jürgen Vogel (SAP)	
Reviewers:		

Project co-funded by the European Commission within the Seventh Framework Programme (2007–2013)		
Dissemination Level		
PU	Public	☐
PP	Restricted to other programme participants (including the Commission)	☐
RE	Restricted to a group specified by the consortium (including the Commission)	☐
CO	Confidential, only for members of the consortium (including the Commission)	☒

Version History

Version	Date	Comments, Changes, Status	Authors, Reviewers
0.1	04/08/2010	Structure and Outline	Reto Krummenacher (UIBK)
0.2	09/08/2010	Section 3 – W3C Standards	Daniel Winkler (UIBK), Reto Krummenacher
0.3	09/08/2010	Section 4 – Activities	Reto Krummenacher, Jürgen Vogel (SAP)
0.4	12/08/2010	Section 2.1 – WSMO-Lite Submission	Dave Lambert (OU)
0.5	13/08/2010	Section 2 – Final	Jacek Kopecký (OU)
0.6	16/08/2010	Final Draft (Internal Review)	Reto Krummenacher
0.7	16/08/2010	Integration Review Feedback (John Davies, BT)	Reto Krummenacher, Jacek Kopecký
0.8	20/08/2010	Integration Review Feedback (Patrick Un, SAP)	Reto Krummenacher
0.9	20/08/2010	Updates Section 2/4	Jacek Kopecký
1.0	20/08/2010	Final Submission	Reto Krummenacher

Table of Contents

Glossary of Acronyms

Acronym	**Definition**
AC	Advisory Committee (W3C)
BLD	Basic Logic Dialect (RIF)
D	Deliverable
DLP	Description Logics Programs
DTB	DataTypes and Built-ins (RIF)
EC	European Commission
KIT	Karlsruhe Institute of Technology
OU	The Open University
OWL	Web Ontology Language
OWL-S	Semantic Markup for Web Services
RIF	Rule Interchange Format
SA-REST	Semantic Annotations for REST
SA-WSDL	Semantic Annotations for WSDL and XML Schema
SPARQL	SPARQL Protocol And RDF Query Language
SWS	Semantic Web Services
UIBK	University of Innsbruck
USDL	Unified Service Description Language
W3C	World Wide Web Consortium
WD	Working Draft (W3C)
WG	Working Group (W3C)
WP	Work Package
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
XG	Incubator Group (W3C)

Executive summary

The purpose of this deliverable is to provide an update on the activities done within SOA4All with respect to W3C standards and standardization. To ensure uptake of SOA4All technology and to increase the impact of project results, it is important to comply to established standards and to be involved and contribute to standardization activities. In the context of W3C, SOA4All is mainly involved in activities related to languages, conceptual models and reasoning.

This deliverable elaborates on the standardization efforts done so far in the scope of W3C, and reports on how SOA4All work is aligned with standards. Moreover, it also reports on related activities that are driven by SOA4All that should lead to more impact and more standardization in the context of Semantic Web services and service models in general.

To this end, the deliverable details in five points:

- W3C member submission of WMSO-Lite
- W3C member submission based on MicroWSMO and hRESTS
- Alignment with W3C recommendations RIF/OWL2
- Involvement in the SPARQL 1.1 definition
- Ongoing activities related to W3C

1 Introduction

To ensure greater impact of project results, SOA4All is involved in several relevant standardization activities and bodies; one of them being the World Wide Web Consortium W3C.[1] The nature of standardization makes such involvement a long-term effort, which often runs behind the actual research of the project. The coordination with many external players required or reaching adequate consensus on the standard simply takes time; often there is a requirement for multiple cycles and negotiations. In SOA4All, standardization activities are running during the whole project life-time, and encompasses the application of standards, the observation and contribution of standards, and the submission of own proposals to standardization bodies. In the context of W3C, those are called member submissions. This present deliverable provides a M30-snapshot of the work done within SOA4All with respect to W3C. Note that although this is the last report on standardization in the scope of W3C, SOA4All will continue to leverage standards and to contribute with long-term goals in mind.

1.1 Purpose and Scope

The purpose of this deliverable is to provide an updated overview on the activities done within SOA4All with respect to W3C standards and standardization. To ensure uptake of SOA4All technology and to increase the impact of project results, it is important to comply to established standards and to be involved and contribute to standardization activities. In the context of W3C, SOA4All is mainly involved in activities related to languages, conceptual models and reasoning.

The deliverable will thus report on the member submissions of WSMO-Lite and MicroWSMO and how they relate to previous submissions in the area of Semantic Web services and service annotations. Furthermore, we show how the project links to established working groups and their standards; namely the OWL2 WG, the RIF WG and the SPARQL WG.[2]

Additionally, we give some indications about ongoing initiatives that were established on SOA4All initiative with the aim to increase the visibility and impact of Semantic Web services' research, not at last with respect to standardization. In this context, there are discussions on the way to collaborate with the SA-REST team on the establishment of an Incubator Group at W3C, and with SAP on how to bring their Unified Service Description Language (USDL) closer to standardization.

1.2 Structure

After this short introduction, the deliverable focuses in Section 2 on the two member submissions from SOA4All, namely WSMO-Lite (Section 2.1) and the one coming from MicroWSMO (Section 2.2). Moreover, Section 2 reports on how the SOA4All submissions relate to other W3C member submissions.

In Section 3, we report on the alignment and contribution to existing W3C recommendations such as OWL2, RIF and SPARQL. This part mostly relates to the implementation

[1] http://www.w3.org

[2] OWL2: http://www.w3.org/2007/OWL/wiki/OWL_Working_Group; RIF: http://www.w3.org/2005/rules/wiki/RIF_Working_Group; SPARQL: http://www.w3.org/2009/sparql/wiki/Main_Page

of reasoners, tools and libraries that allow for working with standardized data sets and languages.

Section 4 summarizes some ongoing activities around Semantic Web services and related efforts. Notably, SOA4All established an initiative termed “Augmented Web Services” whose aim is to re-establish a strong community of researchers and practitioners in Semantic Web services. A main goal is to discuss strategies to create higher impact and acceptance of ongoing work, including the SAP-driven USDL.

Finally, Section 5 concludes the deliverable. In the appendix, the two member submissions are shipped along with the deliverable.

2 W3C Submissions

The key innovation in SOA4All's approach to service semantics are the frameworks for introducing lightweight semantics into Web pages: WSMO-Lite and MicroWSMO. For these two technologies to achieve maximum impact, both for the project and for service semantics in general, it is imperative that they become widely accepted. It has always been SOA4All's position that they both be sent to the W3C as Member Submissions, with the intention that they be considered for progression to full W3C Recommendations. The WSMO-Lite submission is currently under consideration by the W3C, but due to the W3C Process, **this information is confidential**. The submission based on MicroWSMO is prepared but requires further external coordination of efforts, and should also be treated in confidentiality.

2.1 WSMO-Lite Submission

WSMO-Lite builds on SAWSDL [6], which itself extends WSDL [11]. As SAWSDL provides a basic vocabulary for inserting Semantic Web references within a WSDL description, WSMO-Lite provides a small vocabulary to type the annotations made with SAWSDL. Using WSMO-Lite, annotators using SAWSDL can indicate more precisely the intent of the various annotations. WSMO-Lite can also be used in the same way to enrich the annotations made using MicroWSMO. WSMO-Lite specifies a very small number of new terms:

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

wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class .
wsl:Condition              a  rdfs:Class .
wsl:Effect                 a  rdfs:Class .
wsl:NonfunctionalParameter a  rdfs:Class .
```

When SAWSDL's `modelReference` is used to link to elements of some semantic model, WSMO-Lite terms can be used to 'type' those elements. `FunctionalClassificationRoot` marks a class that is a root of a service classification taxonomy. The `Condition` and `Effect` classes enable a reference to be typed as containing a logical expression specifying pre- or post- conditions for the service. `NonfunctionalParameter` specifies a placeholder for a concrete domain-specific nonfunctional property.

Following incubation in the CMS working group[1] and elaboration during the SOA4All project, the WSMO-Lite specification is now in its final form. The W3C process requires formal 'statements of support' from W3C members. At the time of writing, we have already secured such statements from two SOA4All partners (OntoText and The Open University), and are pursuing additional statements from other project partners. Regardless of whether further statements of support are obtained, the submission to W3C will be made by the end of August 2010.

[1] http://cms-wg.sti2.org/home/

2.2 MicroWSMO-based Submission

To include support for RESTful services, SOA4All has developed MicroWSMO and hRESTS, two microformats that are used to structure the HTML documentation of RESTful services so that they become machine-processable in a manner similar to WSDL and SAWSDL. The work also resulted in a minimal procedure-oriented service model common between WS-* services described in WSDL, and RESTful services described with hRESTS and MicroWSMO. The main elements of the model are shown below:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix posm: <http://www.wsmo.org/ns/posm/0.1#>.

# service structure
posm:Service            a rdfs:Class .
posm:hasOperation       a rdf:Property ;
                        rdfs:domain posm:Service ;
                        rdfs:range posm:Operation .
posm:Operation          a rdfs:Class .
posm:hasInputMessage    a rdf:Property ;
                        rdfs:domain posm:Operation ;
                        rdfs:range posm:Message .
posm:hasOutputMessage   a rdf:Property ;
                        rdfs:domain posm:Operation ;
                        rdfs:range posm:Message .
posm:hasInputFault      a rdf:Property ;
                        rdfs:domain posm:Operation ;
                        rdfs:range posm:Message .
posm:hasOutputFault     a rdf:Property ;
                        rdfs:domain posm:Operation ;
                        rdfs:range posm:Message .
# message structure
posm:Message            a rdfs:Class .
                        rdfs:subClassOf posm:MessagePart .
posm:MessagePart        a rdfs:Class .
posm:hasMessagePart     a rdf:Property ;
                        rdfs:domain posm:MessagePart ;
                        rdfs:range posm:MessagePart .
posm:hasMandatoryPart   rdfs:subPropertyOf posm:hasMessagePart .
posm:hasOptionalPart    rdfs:subPropertyOf posm:hasMessagePart .
```

On top of this model, MicroWSMO adopts all SAWSDL properties for semantic annotations equivalent to those in WSDL.

While MicroWSMO and hRESTS are formulated as microformats that are concise and author-friendly, the same annotations can also be expressed in the W3C standard approach for annotating HTML with RDF, called RDFa [5]. Our W3C submission of hRESTS and MicroWSMO is formulated with RDFa, since such a formulation introduces the least amount of new syntax, and as such, may be more acceptable to the W3C and its members. The tools and other implementations in SOA4All use the microformat syntaxes, however a switch to an RDFa approach would not affect the users of SOA4All software, or any evaluation results of the project, in any way.

The draft submission document is in Appendix B; at the time of writing, we are securing support from SOA4All partners that are also W3C members. Since the submission contains the hRESTS service model, which was developed together with a member of the team that also submitted SA-REST (see the next section), the MicroWSMO-based submission also requires coordination with that team; a meeting with the team's leader is planned in September. Until publication, information about this submission is confidential.

2.3 Relationship to Other Member Submissions

The W3C has accepted a number of submissions in the area of Semantic Web Services: WSMO[2], OWL-S[3], WSDL-S[4] and, most recently, SA-REST[5]. WSMO and OWL-S are comprehensive fully-fledged SWS frameworks; despite their completeness, these frameworks have seen very limited uptake.

WSDL-S is a lightweight proposal that takes the approach of annotating WSDL descriptions; it is closer to the actual Web services practice and it was agreed as the basis for the first step of standardization. Based on WSDL-S, the W3C has produced the simple standard SAWSDL, as a way of making the necessary first step towards standardization of SWS without committing to any of the diverse models.

WSMO-Lite builds directly on top of SAWSDL, with the intent of being the second small step towards standardization of SWS; in a way, it is an extract of WSMO and OWL-S, built on top of SAWSDL, and focusing only on the modeling of services for supporting their clients.

The recent submission of SA-REST turns the attention to RESTful services, an approach to Web services that has so far eluded (or not required) standardization. The submitted version of SA-REST provides several properties that appear to be an elaboration of SAWSDL, formulated as annotations of HTML documents. However, the SA-REST properties are only attached to parts of HTML pages, without giving them any meaning related to Web services.

Continuing along the direction of SA-REST, our submission based on MicroWSMO and hRESTS would provide a concrete and yet lightweight RDF-based approach for uniting RESTful services with WS-* services in terms of machine-processable descriptions. For semantic annotations, we directly adopt SAWSDL; but the model may also carry SA-REST properties, if they prove useful.

The W3C has done very little work around RESTful services; SA-REST and our submission together may be an impulse for the W3C to turn its attention to the increasingly important and popular set of technologies and conventions that shape RESTful APIs.

Section 4 provides further details on our collaboration with the submitters of SA-REST.

[2]http://www.w3.org/Submission/WSMO
[3]http://www.w3.org/Submission/OWL-S
[4]http://www.w3.org/Submission/WSDL-S
[5]http://www.w3.org/Submission/SA-REST

3 Aligning with W3C Language Recommendations

3.1 WSML, WSML2Reasoner and W3C Standards

The WSML2Reasoner framework is at the core of the recent efforts of aligning the updated language family (WSML v2.0) with existing W3C recommendations. The most important work by W3C to be targeted is related to the OWL 2 and RIF recommendations.

WSML2Reasoner not only provides a pluggable platform for different reasoning components, but is also shipped with strong language support and a number of parsers and serializers for standard W3C languages. RIF4J, for example, that was developed by the University of Innsbruck in collaboration with the SOA4All project team, provides an object model for RIF rules that can be used on top of WSML2Reasoner. RIF4J offer the technicalities to translate RIF rules to WSML rules. Different RIF profiles are mappable to specific WSML dialects; cf. Section 3.2. To this end, with help of RIF4J, the WSML2Reasoner framework supports various RIF language profiles.

Furthermore, the WSML v2.0 language and the WSML2Reasoner framework have (almost) full support for the latest XML Schema Datatypes recommendation by W3C [3]; currently not supported are ENTITIES, NMTOKENS and IDREFS — details are given in Deliverable D3.2.6 [9]. Likewise there is extensive support for the RIF datatypes and built-ins that were very recently standardized by W3C [4].

In summary, the WSML2Reasoner framework provides a large collection of tools and reasoning facades to align various W3C language standards with WSML and to reasoning over OWL 2 knowledge bases and/or RIF rule bases.

3.2 IRIS – A Reasoner for RIF Profiles

IRIS is a Datalog reasoner that is used within the WSML2Reasoner framework for rule-based reasoning; technical details about the latest release of IRIS are given in [9].[1]. IRIS supports the profiles RIF Core and RIF BLD,[2] and as such has support for rule head equality too. Moreover, as for WSML2Reasoner, IRIS has (almost) full XML Schema Datatype and RIF DTB (datatypes and built-ins) support; except for RIF lists.

From a language perspective, WSML-Rule v2.0 is aligned with RIF BLD, while WSML-Flight v2.0 corresponds to RIF BLD without function symbols, thus subsumes RIF Core.

To this end, IRIS not only provides a reasoning engine for the rule-based WSML languages, but yields a suitable implementation for the W3C RIF standards. Notably, the implementation for RIF is listed by the RIF working group.[3]

3.3 ELLY – A Reasoner for OWL 2 Profiles

ELLY is a reasoner for entailment and satisfiability checking of ELP rule bases.[4] ELP has been presented as a useful hybrid between both rule-based and description logics that combines EL++ with DLP in a novel way and captures the semantics of the OWL 2 profiles RL and EL. From an implementation perspective, ELLY is built on top of the IRIS reasoner

[1]See also `http://www.iris-reasoner.org`

[2]BLD := Basic Logic Dialect, [1].

[3]`http://www.w3.org/2005/rules/wiki/Implementations`

[4]`http://korrekt.org/page/ELP`

and thanks to the integrated use of the OWL API (`http://owlapi.sourceforge.net/`) it can be used as reasoner for the OWL 2 EL and OWL 2 RL profiles. ELLY is presented and evaluated in Deliverable D3.2.7 [10] and can be downloaded at `http://elly.sourceforge.net/`.

W3C recognizes ELLY as an implementation for EL and RL and lists the reasoner in the OWL Working Group's implementation table.[5] There, the following description of ELLY is given: "Data-centric implementation of reasoning and query answering based on translating EL/RL to datalog in a way that preserves assertional entailments." Furthermore, ELLY passes all but one OWL 2 EL conformance tests, as noted on to the OWL Test Suite Status page.[6]

3.4 SPARQL 1.1

The processing of RDF data in various ways is an essential baseline functionality for SOA4All. Effectively, storing, changing and querying RDF-based service descriptions, monitoring data and user profiles is at the core of the SOA4All platform; hence also the investments in semantic spaces. The standard access language and protocol for RDF data is SPARQL. Consequently, the SOA4All consortium has interests and needs in the recent development around SPARQL 1.1, in particular the SPARQL 1.1 Update language and the corresponding SPARQL 1.1 Uniform HTTP Protocol [7, 2]. This feature of publishing and altering RDF data and RDF graphs was entirely missing in SPARQL 1.0 which only standardized the basic query language and answer serialization.

In 2009, the W3C started a SPARQL WG (previously Data Access WG) to develop such new features. The SOA4All consortium observed the developments in the SPARQL WG and contributed reviews and feedback to the latest working drafts published in June 2010 with a focus on the aforementioned drafts, as well as the Service Description vocabulary [8].

[5] `http://www.w3.org/2007/OWL/wiki/Implementations`

[6] `http://www.w3.org/2007/OWL/wiki/Test_Suite_Status`

4 Ongoing Standardization Activities Related to W3C

The standardization of Semantic Web service research results seemed to be on a very good track with the W3C member submissions of the OWL-S and WSMO conceptual models and languages in 2004/2005.[1] Nonetheless, the expected success of Semantic Web services fell short of the expectations. In April 2006, W3C started the SAWSDL Working Group as part of the W3C Web Services Activity. The working group finished in August 2007 with a first small recommendation for annotating Web services (SAWSDL [6]). The only follow-up activity at W3C was the very recent member submission of SA-REST that seeks to standardize three basic properties that can be used to annotate REST API descriptions in HTML/XHTML documents.[2] The SOA4All-driven member submissions of WSMO-Lite and MicroWSMO (Section 2) explicitly complement SAWSDL and SA-REST, and help to foster a new community at W3C. Effectively, the W3C team has already concretely suggested to launch an Incubator Group (XG) on the matters of annotating Web services and REST APIs.

Not at last thanks to these standardization activities, various members of the SOA4All consortium, namely KIT, OU and UIBK, are in continuous contact with the editors of SA-REST, and joint or complementary follow-up actions are going on. The authors of SA-REST, for example, joined the Semantic Web Services Status & Strategy meeting in Frankfurt that was organized as a SOA4All initiative to promote Semantic Web service research, technology and developments and to adjust the research roadmap and business strategies. Although initiated and organized by members of SOA4All, the initiative in on purpose kept open to offer a non-restrictive communication platform for various activities around Semantic Web services, and related fields such as the Web, Semantic Web or Linked Data.

The goal of this SOA4All-based initiative, also promoted under the name Augmented Web Services,[3] is to establish a series of events with the aim of bringing together researchers and practitioners interested in the recent developments, their evolution and cross-fertilization. Such wider discussions and collaborations are essential for successful standardization of Semantic Web services models and languages.

After the aforementioned first meeting in Frankfurt on June 15, 2010, another informal meeting is planned for September between the SA-REST team and the OU, where standardization plans will also be discussed; and then there is a next workshop in preparation that will take place on September 30/October 1 in Karlsruhe on connecting the Unified Service Description Language (USDL)[4] with existing Web standards. The goal of the meeting is to elaborate strategies for USDL towards an even higher impact, in particular in regards to the relationships to other standards, and potential standardization activities.

The USDL initiative was launched by SAP in response to an observed divergence between available service technologies and service consumption in business environments. Services are increasingly being exposed and delivered outside company firewalls and on the Internet through the rise of on-demand applications, cloud computing, and business networks. As the forms of services being exposed become more sophisticated, the conventional approach for describing and accessing services, largely through technical interface descriptions, pose limitations. These approaches are concerned with technical access, leaving open the way consumers understand details of business operations, pricing, legal and other implications of using services. In addition to consumers requesting services, emerging

[1] `http://www.w3.org/Submission/OWL-S`, `http://www.w3.org/Submission/WSMO/`
[2] `http://www.w3.org/Submission/SA-REST/`
[3] `http://www.augmented-webservices.org`
[4] `http://www.internet-of-services.com/index.php?id=12`

third-party intermediaries who can deliver and further monetize services, in the form of brokers, channel partners and cloud providers, also raise questions about how the high-level details of services can be described in a generic and universal way.

The objective of USDL is to complement existing service description frameworks with a fundamental service model that could serve as the basis for an open Internet of Services. To this end, USDL provides a conceptual model for describing business and technical services by offering means to specify crucial service semantics on top of the more technical descriptions of existing standards, e.g. the pricing model of a service. Such information can be seen as non-functional service properties encoded in a pre-defined way. USDL therefore aims to support the seamless interaction of different participants in service networks, e.g., service providers, consumers, or hosters.

For best adoption and utility, the to goal is to evolve USDL into a standard applicable to all types of Web services. At the current point in time, the general mechanism of describing a Web service with a USDL description is best seen to be standardized by the W3C, analogue to WSDL and SAWSDL. For this reason, SOA4All has established a new task (T11.3 ”USDL Pre-Standardization”) to join forces into the investigation of such possibilities. The meeting in Karlsruhe serves as an open kick-off meeting to the task.

5 Conclusions

In this deliverable we listed a number of SOA4All results and ongoing activities related to W3C standards and standardization. In this context, SOA4All is mainly involved in work related to languages, conceptual models for services and reasoning. Consequently, the deliverable elaborated on the long-term activities that are driven by SOA4All in order to increase the impact and the standardization of Semantic Web services-related technologies. To this end, the deliverable detailed the work of SOA4All in five points: W3C member submission of WMSO-Lite, W3C member submission based on MicroWSMO, Alignment with W3C recommendations RIF/OWL2, contribution to the SPARQL 1.1 specification, and further ongoing activities related to W3C in terms of SA-REST, USDL and other initiatives.

The submissions of WSMO-Lite and MicroWSMO are only the next important steps towards greater impact, and are likely to further improve the chances for an Incubator Group at W3C. This demands, as much as all other efforts, the coordination with many external players, which makes the subsequent standardization efforts a time-consuming task. To this end, although being the last deliverable on W3C-related standardization, the efforts described in this document only provide a snapshot about ongoing efforts, and not a final report. The work will continue, also beyond the duration of SOA4All.

References

[1] H. Boley and M. Kifer. RIF Basic Logic Dialect. W3C Recommendation, June 2010.

[2] Ch. Ogbuji. SPARQL 1.1 Uniform HTTP Protocol for Managing RDF Graphs. W3C Working Draft, June 2010.

[3] D. Peterson, S. Gao, A. Malhotra, C.M. Sperberg-McQueen, and H.S. Thompson. XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. W3C Working Draft, December 2009.

[4] A. Polleres, H. Boley, and M. Kifer. RIF Datatypes and Built-Ins 1.0. W3C Recommendation, June 2010.

[5] RDFa in XHTML: Syntax and Processing. Recommendation, W3C, October 2008. Available at `http://www.w3.org/TR/rdfa-syntax/`.

[6] Semantic Annotations for WSDL and XML Schema. Recommendation, W3C, August 2007. Available at `http://www.w3.org/TR/sawsdl/`.

[7] S. Schenk, P. Gearon, and A. Passant. SPARQL 1.1 Update. W3C Working Draft, June 2010.

[8] G.T. Williams. SPARQL 1.1 Service Description. W3C Working Draft, June 2010.

[9] Daniel Winkler and Matthias Pressnig. D3.2.6 Second Prototype Rule Reasoner for WSML-Rule v2.0, August 2010. Deliverable D3.2.6 of the project SOA4All.

[10] Daniel Winkler and Matthias Pressnig. D3.2.7 Second Prototype for Description Logic Reasoner for WSML-DL v2.0, August 2010. Deliverable D3.2.7 of the project SOA4All.

[11] Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at `http://www.w3.org/TR/wsdl20/`.

A WSMO-Lite Submission

The following document is confidential until the W3C decides to acknowledge it as a submission and publishes it.

WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web

W3C Member Submission 23 August 2010

This version:
http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/
Latest version:
http://www.w3.org/Submission/WSMO-Lite/

Authors:
Dieter Fensel, University of Innsbruck
Florian Fischer, University of Innsbruck
Jacek Kopecký, The Open University (formerly University of Innsbruck)
Reto Krummenacher, University of Innsbruck
Dave Lambert, The Open University
Tomas Vitvar, University of Innsbruck

Abstract

In this document we define a lightweight set of semantic service descriptions in RDFS that can be used for annotations of various WSDL elements using the SAWSDL annotation mechanism. These annotations cover functional, behavioral, nonfunctional and information semantics of Web services, and are intended to support tasks such as (semi-)automatic discovery, negotiation, composition and invocation of services. We exploit RDF and RDFS as well as their various extensions such as OWL and RIF for semantic service descriptions.

Status of this document

Table of Contents

1. Introduction

Existing service description languages, especially WSDL, allow service providers to describe their service offerings so that a client can make an up-front decision on whether and how to consume the service's functionality. Their uptake will lead to environments where thousands of services will have to be searched, integrated and mediated, and where automation will be the key enabler of service provisioning to end-users. In order to fulfill these challenges, existing service specifications need to be augmented with semantic descriptions [Studer et al., 2007]

In 2007, the W3C finished its work on Semantic Annotations for WSDL and XML Schema (SAWSDL). SAWSDL defines simple extensions for WSDL and XML Schema to link WSDL components with arbitrary semantic descriptions. It thus provides the building blocks for a bottom-up approach to semantic service modeling: it supports the piecemeal addition of semantics on top of WSDL, without prescribing a semantic framework, thereby allowing selected results from various existing approaches to be adopted in an incremental fashion. As the basis for bottom-up modeling, SAWSDL is independent of any particular semantic technology, i.e., it does not define any types, forms or languages for semantic descriptions.

In this document, we specify the WSMO-Lite service ontology as the next evolutionary step after SAWSDL, filling the SAWSDL annotations with concrete semantic service descriptions. With the ultimate goal to support real-world challenges in intelligent service integration, WSMO-Lite addresses the following requirements:

- Identify the types and a simple vocabulary for semantic descriptions of services (a service ontology).
- Define an annotation mechanism for WSDL using this service ontology.
- Provide the bridge between WSDL, SAWSDL and (existing) domain-specific ontologies such as classification schemas, domain ontology models, etc.

Even though we adopt the base Web service model from WSDL and SAWSDL, WSMO-Lite is inspired by the WSMO ontology, focusing on a small subset of it to define a limited extension of SAWSDL.

2. Web Service Descriptions

Our conceptual model for Web Service Descriptions adopts the following general types of service contracts (adapted from [Sheth, 2003]):

- *Information Model Descriptions* define the data model for input, output, and fault messages, as well as for the data relevant to the other aspects of the service description.
- *Functional Descriptions* define service functionality, that is, what a service can offer to its clients when it is invoked.
- *Nonfunctional Descriptions* define any incidental details specific to a service provider or to the service implementation or its running environment. An example nonfunctional property is the price; the functionality of a service is generally not affected by the price, even though the desirability may be. Nonfunctional properties also include Quality of Service (QoS) aspects such as performance, reliability, and so on.
- *Behavioral Descriptions* define external and internal behavior. The former is the description of a public *choreography*, the protocol that a client needs to follow when consuming a service's functionality; and the latter is a description of a workflow, i.e., how the functionality of the service is aggregated out of other services.
- *Technical Descriptions* define messaging details such as message serializations, communication protocols, and physical service access points.

In the following sections, we show how the above general description types for service contracts are represented at the syntactic and the semantic level.

2.1 Syntactic Level

In regard to SOA technology developments today, the Semantic Service Stack represents service contracts at the non-semantic level using the existing standards: WSDL, SAWSDL, and related WS-* specifications. They all use XML as a common flexible data exchange format. Service contracts are represented as follows:

- *Information Model Description* is represented using XML Schema.
- *Functional Description* is represented using a WSDL interface and its operations.
- *Nonfunctional Description* is represented using various WS-* specifications (WS-Policy, WS-Agreement, WS-Reliability, WS-Security, etc.).
- *Behavioral Description* is represented using the relevant WS-* specifications, such as WS-BPEL (for the workflow) and WS-CDL (for the choreography).
- *Technical Description* is represented using WSDL Binding for message serializations and underlying communication protocols, such as SOAP, HTTP; and using WSDL Service for physical endpoint information.

In addition, while SAWSDL does not fall into any of the service contract descriptions categories listed above, it is an essential part of the non-semantic level of the stack, providing the groundwork for the semantic layer. SAWSDL defines a simple extension layer that allows WSDL components to be annotated with semantics, using three extension attributes:

- *modelReference* for pointing to semantic concepts that describe a WSDL component,
- *loweringSchemaMapping* and *liftingSchemaMapping* for specifying the mappings between the XML data and the semantic information model.

2.2 Semantic Level

With the WSMO-Lite service ontology we represent service contracts at the semantic level as follows (see Section 3 for a detailed description of WSMO-Lite):

- *Information Model Descriptions* are represented using a domain ontology, along with associated data lifting and lowering transformations.
- *Functional Descriptions* are represented as *capabilities* and/or functionality *classifications*. A capability defines *conditions* which must hold in a state before a client can invoke the service, and *effects* which hold in a state after the service invocation. Classifications define the service functionality using some classification ontology (i.e., a hierarchy of *categories*).
- *Nonfunctional Descriptions* are represented using an ontology, semantically representing some policy or other nonfunctional properties.
- *Behavioral Descriptions* are not represented explicitly in WSMO-Lite (see discussion in Section 3).
- *Technical Descriptions* are not represented semantically in the service ontology, as they are sufficiently covered by the non-semantic description in WSDL.

In order to create or reuse domain-specific service ontologies, a service engineer can use any RDF-compliant language. This preserves the choice of language expressivity according to domain-specific requirements. Such languages may include RDF Schema (RDFS), Web Ontology Language (OWL), Rule Interchange Format (RIF) or the Web Service Modeling Language (WSML).

RDFS: On top of RDF, RDF Schema (RDFS) defines constructs that allow the expression of some semantics for the RDF model: RDFS allows the definition of classes describing the terminology of the domain of discourse, properties of those classes as well as class and property hierarchies (i.e. *subClassOf* and *subPropertyOf*). Thus, RDFS provides a set of constructs that allow the specification of lightweight ontologies.

OWL, WSML, RIF (on top of RDFS): Where the expressivity of RDFS is not sufficient for modeling of the required knowledge, various specializations of RDFS can be used. Such specializations are being developed both inside and outside of W3C along the lines of knowledge representation paradigms of Description Logic (DL) and Logic Programming (LP).

The Web Ontology Language (OWL) provides a vocabulary along with a formalism based on Description Logics. The Web Service Modeling Language (WSML) defines several variants allowing for both paradigms of Description Logics (WSML-DL) and Logic Programming (WSML-Flight, WSML-Rule). All WSML variants can be represented using RDF syntax and they are layered on top of RDFS. While WSML-DL has a direct mapping to OWL, WSML-Rule is the basis of the Web Rule Language (WRL) specification which served as one input to the W3C's Rule Interchange Format recommendations.

SKOS: as a lightweight ontology for simple knowledge organization systems, SKOS is well suited for capturing classifications of service and operation functionalities. As presented in this submission (in the section below), WSMO-Lite does not directly support SKOS functional classifications, but it would be straightforward to add such support, and the exact specification in the context of WSMO-Lite is part of our planned future work.

3. WSMO-Lite Service Ontology

Listing 1 shows the WSMO-Lite service ontology in RDFS, serialized in Notation 3. Below, we explain the semantics of the WSMO-Lite elements.

Listing 1. WSMO-Lite Service Ontology

```
1  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
2  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix wsl:  <http://www.wsmo.org/ns/wsmo-lite#> .
4
5  wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class .
6  wsl:NonfunctionalParameter  a  rdfs:Class .
7  wsl:Condition  a  rdfs:Class .
8  wsl:Effect     a  rdfs:Class .
```

- *wsl:FunctionalClassificationRoot* (line 5) marks a class that is a root of a classification which also includes all the RDFS subclasses of the root class (the actual functional categories). A classification (taxonomy) of service functionalities can be used for functional description of a service.
- *wsl:NonfunctionalParameter* (line 6) specifies a placeholder for a concrete domain-specific nonfunctional property.
- *wsl:Condition* and *wsl:Effect* (lines 7 and 8) together form a *capability* in a functional service description. Both are expected to use a concrete logical language to describe the logical expressions for conditions and effects.

WSMO-Lite defines behavioral descriptions through functional annotations of operations. While it does not have a special construct for choreography descriptions, the behavioral aspects can be inferred from functional (capability) annotations of individual service operations. Such annotations can be transformed into a WSMO Choreography, using the algorithm described in [Vitvar et al., 2008].

4. WSMO-Lite Annotations for WSDL

In this section, we define the particular types of annotations supported by WSMO-Lite. For this purpose we define two types of annotations, namely *reference annotations* and *transformation annotations*. A reference annotation points from a WSDL component (XML Schema element declaration or type definition, WSDL interface, operation, service) to a WSMO-Lite semantic concept. SAWSDL represents this type of annotation using the *modelReference* extension attribute. A transformation annotation specifies a data transformation called *lifting* from a component of XML schema to an element of ontology, and a reverse transformation (from ontology to XML) called *lowering*. SAWSDL represents this annotation using extension attributes *liftingSchemaMapping* and *loweringSchemaMapping* respectively. The actual expression of concrete lifting or lowering transformations is out of scope of this specification. Non-normatively, we expect WSMO-Lite to work well with transformation technologies such as XSLT, XQuery, and especially XSPARQL.

Listing 2 depicts an example ontology which we use to show various annotations.

Listing 2. Example of domain-specific service ontology

```
 1  # namespaces and prefixes
 2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
 3  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
 4  @prefix wsl:  <http://www.wsmo.org/ns/wsmo-lite#> .
 5  @prefix ex:   <http://example.org/onto#> .
 6  @prefix xs:   <http://www.w3.org/2001/XMLSchema#> .
 7
 8  # domain ontology
 9  ex:Customer  a  rdfs:Class .
10  ex:hasService  a  rdf:Property ;
11      rdfs:domain ex:Customer ; rdfs:range ex:Service .
12  ex:Service  a  rdfs:Class .
13  ex:hasConnection  a  rdf:Property ;
14      rdfs:domain ex:Customer ; rdfs:range ex:NetworkConnection .
15  ex:NetworkConnection  a  rdfs:Class .
16  ex:providesBandwidth  a  rdf:Property ;
17      rdfs:domain ex:NetworkConnection ; rdfs:range xs:integer .
18  ex:VideoOnDemandService rdfs:subClassOf ex:Service .
19
20  # capability description example
21  ex:VideoOnDemandSubscriptionPrecondition  a  wsl:Condition ;
22      rdf:value
23          "Prefix (ex <http://example.org/onto#>)
24           Prefix (pred <http://www.w3.org/2007/rif-builtin-predicate#>)
25           And (?Customer#ex:Customer
26                ?Customer[ex:hasConnection->?Connection]
27                ?Connection#ex:NetworkConnection
28                ?Connection[ex:providesBandwith->?Y]
29                External(pred:numeric-greater-than(?Y 1000)))" .
30  ex:VideoOnDemandSubscriptionEffect  a  wsl:Effect ;
31      rdf:value
32          "Prefix (ex <http://example.org/onto#>)
33           Prefix (pred <http://www.w3.org/2007/rif-builtin-predicate#>)
34           And (?Customer#ex:Customer
35                ?Customer[ex:hasService->?Service]
36                ?Service#ex:VideoOnDemandSubscription)" .
37  # nonfunctional property example
38  ex:PriceSpecification rdfs:subClassOf wsl:NonfunctionalParameter .
39  ex:VideoOnDemandPrice  a  ex:PriceSpecification ;
40      ex:pricePerChange "30"^^ex:euroAmount ;
41      ex:installationPrice "49"^^ex:euroAmount .
42
43  # classification example
44  ex:SubscriptionService  a  wsl:FunctionalClassificationRoot .
45  ex:VideoSubscriptionService rdfs:subClassOf ex:SubscriptionService .
46  ex:NewsSubscriptionService rdfs:subClassOf ex:SubscriptionService .
```

The listing defines a simple ontology for a telecommunication service (lines 9-18); the capability for a concrete Video-on-Demand subscription service (lines 21-36) for which the condition says that the customer must have a network connection with some minimal bandwidth and the effect says that the customer will be subscribed to the service; a nonfunctional property describing the pricing (lines 38-41); and a simple functionality classification with three categories (lines 44-46). In this example, the condition and effect are specified in the W3C's Rule Interchange Format. For clarity and brevity, the example uses RIF's presentation syntax, but in practice RIF rules would be given in the XML serialization syntax.

Figure 1. Illustration of WSMO-Lite Annotations

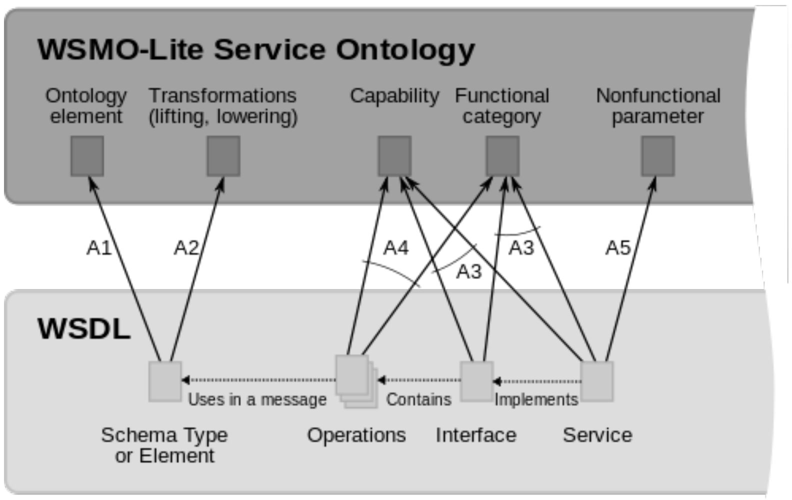


Figure 1 illustrates the positioning of WSMO-Lite annotations (marked *A1...A5*), which are defined as follows:

Annotation A1: Ontological annotations of XML Schema. The schema used in WSDL to describe messages, i.e., the element declarations and type definitions (and in WSDL 1.1, messages and their parts), can carry reference annotations linking to classes from the service information model ontology.

Annotation A2: Transformation annotations of XML Schema. To be able to communicate with a service, the client needs to transform data between its semantic model and the service-specific XML message structures. The schema may contain transformation annotations (lifting or lowering) which specify the appropriate mappings.

The following listing shows an example of annotations *A1* and *A2* (the content of the lowering transformation is omitted for brevity).

Listing 3. Example of annotations A1 and A2

```
<xs:element name="NetworkConnection" type="NetworkConnectionType"
   sawsdl:modelReference="http://example.org/onto#NetworkConnection"
   sawsdl:loweringSchemaMapping="http://example.org/NetCn.xslt"/>
```

Annotation A3: Functional annotations of WSDL Interface and Service. Functional descriptions (both capabilities and categories) apply both to concrete Web services and to the reusable and abstract interfaces. A reference annotation points from a service or an interface to its appropriate functional description.

The following listing shows an example of multiple *A3* annotations.

Listing 4. Example of annotation A3

```
<wsdl:interface name="NetworkSubscription"
   sawsdl:modelReference="http://example.org/onto#VideoSubscriptionService
                          http://example.org/onto#VideoOnDemandSubscriptionPrecondition
                          http://example.org/onto#VideoOnDemandSubscriptionEffect">
    <wsdl:operation name="CheckNetworkConnection" ... />
</wsdl:interface>
```

Note that a WSDL interface may be shared by multiple services, therefore the functional description of the interface should be general. A concrete functional description attached to the service then refines the functional description of the interface. Additionally, aggregate interfaces or services (i.e., those that combine multiple potentially independent functionalities) may be annotated with multiple functional descriptions.

Annotation A4: Functional annotations of WSDL Interface operations. Functional descriptions (both capabilities and categories) apply also to interface operations, to indicate their particular functionalities. A reference annotation points from an operation to its appropriate functional description.

Functional annotation of interface operations can be used for services whose functionality has different, separate sub-parts. For example, a network subscription service may offer operations for subscription to a bundle, cancelation of a subscription, or price inquiry. A client will generally only want to use one or two of these operations, not all three. Service invocation will therefore need to select only the operations applicable to the current goal, and invoke them in the correct order, which is also partially implied by the functional operation annotations.

Please note that annotations *A3* and *A4* apply to both types of functional descriptions, i.e., a capability made up of logical conditions, or a category from some functional classification. It is even possible to combine capabilities and classification together. However, WSMO-Lite does not define any formal relationship between capabilities and classifications on the same component, or between annotations *A3* on a service or its interface, and *A4* on the operations of that interface.

Annotation A5: Nonfunctional annotations of WSDL Service. Nonfunctional descriptions apply to a concrete instance of a Web service, that is, a WSDL Service. A reference annotation can point from a service component to a nonfunctional property. The following listing (Listing 5) shows an example of annotation *A5*, attaching pricing information to a Web service on line 3.

Listing 5. Example of annotation A5

```
1  <wsdl:service name="ExampleCommLtd"
2      interface="NetworkSubscription"
3      sawsdl:modelReference="http://example.org/onto#VideoOnDemandPrice">
4    <wsdl:endpoint ...  />
5  </wsdl:service>
```

Nonfunctional descriptions are always specific to a concrete service, therefore nonfunctional properties should not be expressed on a WSDL interface or on interface operations. In case nonfunctional properties need to be specified on the operations of a service, WSDL binding operation components (which mirror the operations of some interface) may be used to capture these properties.

5. Notes On Related Work

The major stream of related work is in the frameworks for Semantic Web services, including WSMO, Semantic Markup for Web Services (OWL-S) and Web Service Semantics (WSDL-S). Recently there has been increased attention given to RESTful services and Web APIs where, in particular, the work on hRESTS and MicroWSMO also connects to WSMO-Lite.

WSMO is a top-down conceptual model for SWS that defines four top-level components: ontologies, mediators, goals and web services. As we already mentioned, WSMO was the major inspiration for WSMO-Lite. WSMO-Lite was created to address the need for a lightweight service ontology that directly builds on the newest W3C standards and allows bottom-up modeling of services. On the other hand, WSMO is an established framework for Semantic Web Services representing a top-down model identifying semantics, useful in a semantics-first environment. WSMO-Lite adapts the WSMO model and makes its semantics lighter in the following major aspects:

- Beside Web services, WSMO also defines goals and mediators; these are currently not present in WSMO-Lite which is geared towards only describing services through ontologies.
- Where WSMO expects the functional capabilities of services to be described using logical expressions framed as preconditions, assumptions, postconditions, and effects, WSMO-Lite distinguishes only between preconditions and effects (which may include assumptions or postconditions, respectively). WSMO-Lite further specifies a coarser-grained mechanism to describe functionality using functionality classifications (taxonomies), for which WSMO does not have direct support.

Earlier, **OWL-S** was the first major ontology for SWS; it defined three interlinked ontologies: Service Profile (for the functional and nonfunctional descriptions), Service Model (for the behavioral descriptions), and Service Grounding (for physical Web service access). There was also some work published on OWL-S grounding that uses SAWSDL [Martin et al., 2007] [Paolucci et al., 2007]. In comparison with that work, WSMO-Lite takes the additional step of simplifying the annotations into a lightweight ontology.

WSDL-S was created in the METEOR-S project as a specification of how WSDL can be annotated with semantic information. WSDL-S itself does not provide a concrete model for SWS, instead it assumes the concrete model will be expressible as annotations in WSDL and XML Schema documents. The core parts of WSDL-S were taken as the basis for SAWSDL. WSDL-S also included WSDL extensions for attaching preconditions, effects, and categories, but these were out of scope for SAWSDL, and can be moved out to ontologies and attached through model references, as we do in WSMO-Lite.

Finally, **hRESTS and MicroWSMO** aim to enrich the informal descriptions of RESTful services, usually available in HTML, with microformat or RDFa annotations [Kopecký et al., 2008]. In effect, hRESTS forms an analogue of WSDL for RESTful services, and MicroWSMO is analogous to SAWSDL. The WSMO-Lite service semantics ontology is directly applicable in MicroWSMO and hRESTS annotations.

Acknowledgements

The work on WSMO-Lite has been mainly supported by the European Union project SOA4All. The editors would like to thank to all the members of the Conceptual Models for Services Working Group (CMS WG) for their advice and input to this document.

References

[Martin et al., 2007] D. Martin, M. Paolucci, and M. Wagner: *Bringing Semantic Annotations to Web Services: OWL-S* *from the SAWSDL Perspective*, in Proc. of 6th International Semantic Web Conference (ISWC/ASWC), pp. 340-352, 2007.

[Kopecký et al., 2008] J. Kopecký, K. Gomadam, and T. Vitvar: *hRESTS: An HTML Microformat for Describing RESTful Web Services*, in Proc. of IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), pp. 619-625, 2008.

[Paolucci et al., 2007] M. Paolucci, M. Wagner, and D. Martin: *Grounding OWL-S in SAWSDL*, in Proc. of 5th International Conference on Service-Oriented Computing (ICSOC), pp. 416-421, 2007.

[Sheth, 2003] A.P. Sheth: *Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration*, Invited Talk at Workshop on E-Services and the Semantic Web, at WWW 2003 (Presentation).

[Studer et al., 2007] R. Studer, S. Grimm, and A. Abecker: *Semantic Web Services: Concepts, Technologies, and Applications*, Springer-Verlag New York, Inc. Secaucus, NJ, USA. 2007.

[Vitvar et al., 2008] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel, *WSMO-Lite Annotations for Web Services*, in Proc. of 5th European Semantic Web Conference (ESWC), pp. 674-689, 2008.

B MicroWSMO-based Submission

The following document is confidential until it is submitted to the W3C and then until the W3C decides to acknowledge it as a submission and publishes it.

hRESTS^{RDFa}: Describing and Semantically Annotating RESTful Services with RDFa

W3C Member Submission XXX XXX 2010

This version:
http://www.w3.org/Submission/2010/SUBM-hRESTS-RDFa-20100409/
Latest version:
http://www.w3.org/Submission/hRESTS-RDFa/
Author:
Jacek Kopecký, The Open University (formerly University of Innsbruck)
Contributors (in alphabetical order): todo move some contributors to authors? add contributors?
Dieter Fensel, University of Innsbruck
Karthik Gomadam, University of South California (formerly Wright State University)
Dave Lambert, The Open University
Maria Maleshkova, The Open University
Carlos Pedrinaci, The Open University
Tomas Vitvar, University of Innsbruck

Abstract

The Web has evolved to include increasing numbers of Web sites provide machine-oriented APIs and Web services (often called RESTful APIs). However, most APIs are only described with text in HTML documents. The lack of machine-readable API descriptions hinders tool support for the clients of RESTful APIs, and the feasibility of applying semantic annotations and technologies. This specification describes a minimal service model adopted from hRESTS that applies to RESTful APIs, and it shows how this model can be used in RDFa annotations of HTML Web API documentation, making it machine-processable and amenable to semantic annotation with SAWSDL.

Status of this document

Table of Contents

1. Introduction

The Web has gone through great changes since it became popular, evolving from an infrastructure for static content of pages consumed by individuals to a massive-scale communication platform where individuals, companies, and devices alike provide, consume and synthesize content and services. The value of Web applications is no longer only in providing content to consumers but also in exposing functionality through increasing numbers of public APIs designed for machine consumption. Web applications and APIs follow the Web architecture style called REST (Representational State Transfer [Fielding 2001]), and public APIs on the Web are often called "RESTful Web services".

Web application APIs are generally described using plain, unstructured HTML documentation useful only to a human developer. Finding suitable services, composing them ("mashing them up"), mediating between different data formats etc. are currently completely manual tasks. In order to provide tool support or even a degree of automation, we need the API descriptions to be machine-readable. With such descriptions, search engines can gather better information about existing services, and developers can easier use these services. Tools enabled by the existence of such descriptions can support the developer in using the Web APIs and mashing them up with others. With semantic annotations, the Web services and APIs can even be discovered and used automatically in Semantic Web Services systems.

Recently, the W3C has finalized RDFa, "a set of XHTML attributes to augment visual data with machine-readable hints", used "to turn the existing human-visible text and links into machine-readable data without repeating content" [RDFa Primer]. With RDFa, the existing API documentation in HTML can be annotated to support machine processing.

In this specification, we adopt from [Kopecký et al. 2008] a simple service model in RDF that we apply through RDFa to provide machine-readable descriptions of RESTful APIs, supporting many of the same use cases as WSDL [WSDL 2.0].

Once a machine-readable description of a Web service is available, it can be further annotated with additional information, such as semantic descriptions (the functionality of operations, the meaning of the input and output data), or nonfunctional properties (e.g., the price of using the service, QoS guarantees, security and privacy policies). Such annotations extend the utility of service descriptions. Therefore, we adopt SAWSDL [SAWSDL] to add semantic annotations to service descriptions, intended for semantic automation of discovery and use of Web APIs. Such automation has been researched under the name Semantic Web Services (SWS, [Studer et al. 2007]), where the aim is to use semantic technologies to help with the following tasks: *discovery* matches known Web services against a user goal and returns the services that can satisfy that goal; *ranking* orders the discovered services based on user requirements and preferences so the best service can be selected; *composition* puts together multiple services when no single service can fulfill the whole goal; *invocation* then communicates with a particular service to execute its functionality; and *mediation* resolves any arising heterogeneities.

There are several existing formats for machine-readable description of Web APIs, e.g. WADL [Hadley 2009] and even WSDL 2.0 [WSDL 2.0], both amenable to SAWSDL annotations. Probably due to the perceived complexity of these XML formats, they do not seem to be gaining traction with API providers; service descriptions remain mostly in unstructured text. Therefore we propose the lightweight service model applied with RDFa over the existing HTML documentation as a more accessible approach.

In this document, we use the following namespaces in snippets and listings of XHTML and RDF:

Prefix	Namespace URI
ex	http://example.com/
hr	http://www.wsmo.org/ns/hrests#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
sawsdl	http://www.w3.org/ns/sawsdl#
xsd	http://www.w3.org/2001/XMLSchema#

This specification is structured as follows: in Section 2, we discuss the structure of RESTful services and we present a minimal service model, which is then formalized in RDFS in Section 3. Section 4 shows how the model can be applied with RDFa in the HTML documentation of RESTful services. Finally, in Section 5 we straightforwardly adopt SAWSDL for semantic annotations of service descriptions.

2. Modeling RESTful services and Web APIs

Web APIs and RESTful Web services are hypermedia applications consisting of interlinked resources (Web pages) that are oriented towards machine consumption. In their structure and behavior, RESTful Web services can be very much like common Web sites [Richardson & Ruby 2007]. Common Web sites and RESTful Web services both use HTTP [RFC 2616] as the communication protocol. The orientation of RESTful Web services towards machine consumption manifests mainly in the data formats: clients generally interact with RESTful services by sending structured data (e.g. XML, JSON), as opposed to the standard Web document markup language, HTML, which is a human-oriented presentation language.

From the Architecture of the Web [WebArch] and from the REST architectural style [Fielding 2001], we can extract the following concepts inherent in RESTful services:

- a *resource*, identified by a URI that also serves as the endpoint address where clients can send requests;
- every resource has a number of *methods* (in HTTP, the most-used methods are GET, POST, PUT and DELETE) that are invoked by means of request/response message exchanges.
- The messages can carry *hyperlinks*, which point to other resources and which the client can navigate when using the service.
- A hyperlink can simply be a URI, or it can be a *form* which specifies not only the URI of the target resource, but also the method to be invoked and the structure of the input data.

Note that the architecture of the Web contains no formal concept of a *service* as such; a service is a grouping of resources that is useful for developing, advertising and managing related resources.

While the resources of the service (the *nouns*) form a hypermedia graph, the interaction of a client with a RESTful service is a series of operations (the *verbs* or *actions*) where the client sends a request to a resource and receives a response that may link to further useful resources. The hypermedia graph (the links between resources) guides the sequence of operation invocations, but the meaning of a resource is independent of where it is linked from; the same link or form, wherever it is placed, will always lead to the same action. Therefore, the operations of a RESTful Web service can be considered independently from the graph structure of the hypertext.

In the subsection below, we use an example hotel reservation service with a RESTful API to illustrate the terms of Web architecture and how the clients interact with the service effectively by invoking a set of operations.

The client-side independence of operations from the resource and hypermedia structure of a RESTful API allows us to view RESTful services through a procedure-oriented service model with terminology adopted from WSDL, as shown in Table 1. In Section 3, we formalize the model in RDFS, and in Section 4, we apply this model as RDFa annotations of the documentation of RESTful services.

Table 1. Mapping RESTful services into a procedure-oriented minimal service model

RESTful services	Procedure-oriented minimal service model
Service (*a group of resources*)	Service
Resource	– *(not significant to clients on the semantic level)*
Resource method	Operation
Method request/response	Operation input/output
Hyperlink	– *(treated as part of message data)*

Since a WSDL description can trivially be mapped into the same minimal model, this view allows a single semantic client system to support WS-* and RESTful services without regard to their technological differences.

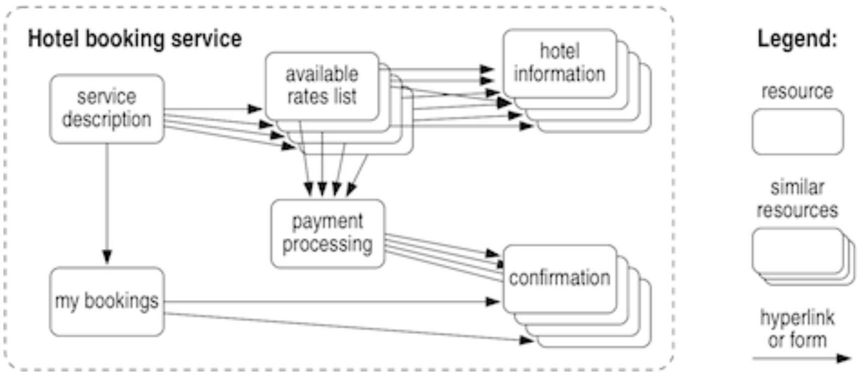
2.1 Illustrative example

To illustrate the applicability of the procedure-oriented minimal service model to RESTful Web services, we analyze an example RESTful Web service. First, in Section 2.1.1 we describe the service in terms of its hypertext structure. In Section 2.1.2 we show that we can view the service as a set of operations, independent from its hypertext structure. Finally, in Section 2.1.3, we show how our example service would typically be described.

2.1.1 Example Service as Hypertext

Figure 1 illustrates an example RESTful hotel booking service, with its resources and the links among them. Together, all these resources form the hotel booking service; however, the involved Web technologies actually work on the level of resources, so *service* is a virtual term here and the figure shows it as a dashed box.

Figure 1. Structure of an example service

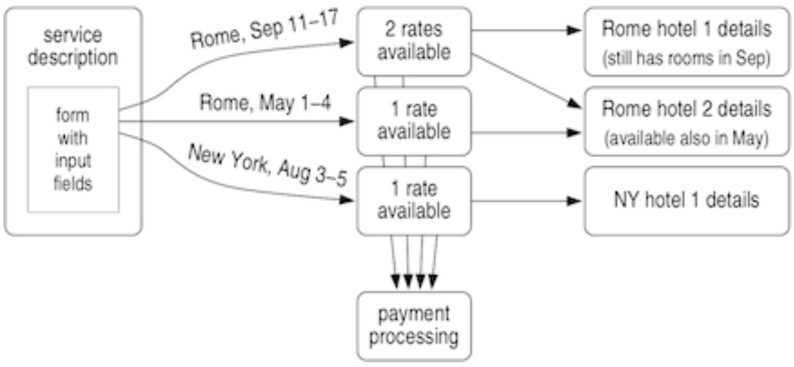


The "service description" is a resource with a stable address and information about the other resources that make up the service. It serves as the initial entry point for client interaction. In a human-oriented Web application, this would be the homepage, such as `http://hotels.example.com/`.

The existence of such a stable entry point lowers the coupling between the service and its clients, and it enables the evolution of the service, such as adding or removing functionality. A client need only rely on the existence of the fixed entry point, and it can discover all other functionality as it navigates the hypermedia. (However, in many cases, a programmatic client is programmed against a given service description before it uses the service, making it harder to react dynamically to changes of the service. This is especially true in IDL-driven technologies such as most of tools for WS-* Web services, but it is also common with programmatic clients and access libraries for Web APIs.)

The service description resource of our example service contains a form for searching for available hotels, given the number of guests, the start and end dates and the location. The search form serves as a parametrized hyperlink to search results resources that list the available rates, as detailed in Figure 2; one resource per every unique combination of the input data. The form prescribes how to create a URI that contains the input data; the URI then identifies a resource that returns the list of available hotels and rates for the particular inputs. As there is a large number of possible search queries, there is also a large number of results resources, and the client does not need to know that all these resources are likely handled by a single software component on the server.

Figure 2. Detail of example service resources



The search results are modeled as separate resources (as opposed to, for instance, a single data-handling resource that takes the inputs in a request message), because it simplifies the reuse of the hotel search functionality in other services or in mashups (lightweight compositions of Web applications), and it also supports caching of the results. Creating the URIs for individual search results resources and retrieving the results (with HTTP GET) is easier in most programming frameworks than POSTing the input data in a structured data format to a single Web resource that would then reply with the list of available hotels and rates.

Search results are presented as a list of concrete rates available at the hotels in the given location, for the given dates and the number of guests, as also shown in Figure 2. Each item of the list contains a link to further information about the hotel (e.g. the precise location, star rating, guest reviews and other descriptions), and a form for booking the rate, which takes as input the payment details (such as credit card information) and an identification of the guest(s) who will stay in the room. The booking data is submitted (POSTed) to a payment resource, which processes the booking and redirects the client to a confirmation resource, as shown in Figure 1. The content of the confirmation can serve as a receipt.

The service description resource also contains a link to "my bookings", a resource that lists the bookings of the current user (which requires authentication functionality). This resource links to the confirmations of the bookings done by the authenticated user. With such a resource available to them, client applications no longer need to have a local store for the information about performed bookings.

The confirmation resources may further provide a way of canceling the reservation (not shown in the pictures, could be implemented with the HTTP DELETE method).

2.1.2 Turning Hypertext into Operations

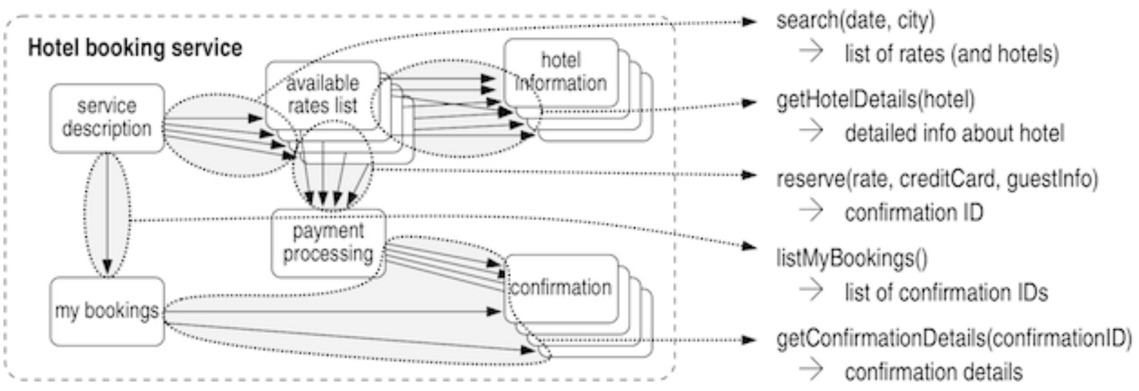
So far, our description of the example hotel reservation service has focused on the hypermedia aspect: we described the resources and how they link to each other. Alternatively, we can also view the service as a set of operations available to the clients — as an API.

The resources of the service (the *nouns*) form a hypermedia graph (shown in Fig. 1). The interaction of a client with a RESTful service is a series of operations (the *verbs* or *actions*) where the client sends a request to a resource and receives a response that may link to further useful resources. Importantly, the links need not be only simple URIs, but they can also be *input forms* that indicate the URI, the HTTP method, and the input data.

The graph nature of a hypermedia service guides the sequence of operation invocations, but the meaning of a resource is independent of where it is linked from; the same link or form, wherever it is placed, will always lead to the same action. Therefore, the operations of a RESTful Web service can be considered independently from the graph structure of the hypertext.

In Figure 3, we extract the operations present in our example service. The search form in the homepage represents a search operation, the hotel information pages linked from the search results can be viewed as an operation for retrieving hotel details, the reservation form for any particular available rate becomes a reservation operation, and so on.

Figure 3. Operations of the example service



2.1.3 HTML Description of the Example Service

Web APIs, or indeed services of any kind, need to be described in some way, so that potential clients can know how to interact with them. While Web applications are self-describing to their human users, Web services are designed for machine consumption, and someone has to tell the machine how to consume any particular service.

Public RESTful Web services are universally described in human-oriented documentation (for instance, see Flickr API and Amazon Simple DB) using the general-purpose Web hypertext language HTML, which is the medium of choice for dissemination of information about Web APIs, along with a vast majority of other textual content.

Typically, such documentation will list the available operations (calling them API calls, methods, commands etc.), their URIs and parameters, the expected output and error conditions and so on; it is, after all, intended as the documentation of a programmatic interface.

The following might be an excerpt of a typical operation description:

> ACME Hotels service API
> **Operation `getHotelDetails`**
>
> Invoked using the method GET at `http://example.com/h/{id}`
> **Parameters:** `id` - the identifier of the particular hotel
> **Output value:** hotel details in an `ex:hotelInformation` document

In HTML, the description can be captured as shown in Listing 1.

Listing 1. Example HTML service description

```
<h1>ACME Hotels service API</h1>
<h2>Operation <code>getHotelDetails</code></h2>

<p>Invoked using the method GET at <code>http://example.com/h/{id}</code><br/>
   <strong>Parameters:</strong>
       <code>id</code> – the identifier of the particular hotel<br/>
   <strong>Output value:</strong> hotel details in an
       <code>ex:hotelInformation</code> document
</p>
```

Such documentation has all the details necessary for a human to be able to create a client program that can use the service. In order to tease out these technical details, the textual documentation needs to be amended in some way, such as using RDFa and the procedure-oriented minimal service model defined in the following section.

In the hypertext of the example service, the service has five operations but only two are directly accessible from the service description resource. While all five operations can be described in a single HTML document, the client may not know any concrete hotel identifiers to invoke `getHotelDetails()`, or any confirmation ID to invoke `getCofirmationDetails()`. The client may save hotel or confirmation identifiers and use them later to invoke these operations without going through availability searches or the list of "my bookings"; this behavior is roughly equivalent to how bookmarks work in a Web browser.

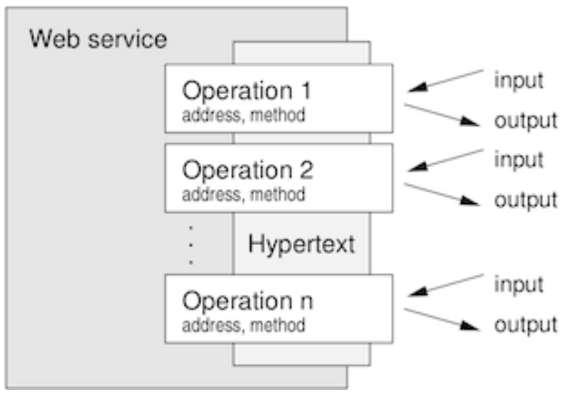
3. hRESTS Minimal Service Model in RDFS

The interaction of a client with a RESTful *service* such as the one in our example is a series of *operations* where the client sends a *request* to a *resource* (using one of the HTTP *methods* GET, POST, PUT or DELETE), and receives a *response* that may *link* to further useful resources. The emphasized words indicate the key concepts that the client encounters:

- *service* is the service (a set of related resources) that the client deals with,
- *operation* is a single action that the client can perform on that service,
- *resource* determines the address (URI) where the operation is invoked,
- *method* captures the HTTP method that implements the operation,
- *request* and *response* are the messages sent as input and output of the operation,
- *links*, especially in the output messages, make up a run-time hypertext graph of related resources.

This leads us to a service model shown in Figure 4. A Web service has a number of operations, each with potential inputs and outputs, and a hypertext graph structure where the outputs of one operation may link to other operations. This model captures the requirements for what we need to represent in a machine-readable description. The model is very similar in its structure to WSDL [WSDL 2.0], only instead of hypertext, WSDL services use the terms "process" or "choreography" for the sequencing of operations.

Figure 4. Functional model of a RESTful Web service



An operation description specifies an address (a URI or a parametrized URI template (URI templates are defined for instance in WSDL 2.0 HTTP Binding in Section 6.8.1.1), the HTTP method (GET, POST, PUT or DELETE), and the input and output data formats. In principle, the output data format can be self-describing (self-description is a major part of Web architecture), but the API documentation should specify what the client can expect.

While at runtime the client interacts with concrete resources, the service description may present a single operation that acts on many resources (e.g. `getHotelDetails()` which can be invoked on any hotel details resource), therefore an operation specifies an *address* as a URI template whose parameters are part of the input data.

Listing 2 shows an RDFS realization of this service model, together with the operation properties described above. Services, their operations, and messages can also have human-readable names, which can be attached using the `rdfs:label` property.

Listing 2. Procedure-oriented minimal service model for RESTful services in RDFS (Turtle syntax)

```
@prefix hr:    <http://www.wsmo.org/ns/hrests#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

# classes and properties of the procedure-oriented minimal service model
hr:Service  a  rdfs:Class .
hr:hasOperation  a  rdf:Property ;
    rdfs:domain  hr:Service ;
    rdfs:range  hr:Operation .
hr:Operation  a  rdfs:Class .
hr:hasInputMessage  a  rdf:Property ;
    rdfs:domain  hr:Operation ;
    rdfs:range  hr:Message .
hr:hasOutputMessage  a  rdf:Property ;
    rdfs:domain  hr:Operation ;
    rdfs:range  hr:Message .
hr:Message  a  rdfs:Class .

# operation properties for RESTful services
hr:hasAddress  a  rdf:Property ;
    rdfs:range  hr:URITemplate .
hr:hasMethod  a  rdf:Property ;
    rdfs:range  xsd:string .

# a datatype for URI templates
hr:URITemplate  a  rdfs:Datatype .
```

4. Applying the hRESTS model with RDFa to documentation of RESTful services

We have seen that a RESTful Web service can be viewed as a hypertext graph of interlinked resources, or as a set of operations to be invoked by the client. While navigating the hypertext graph is natural for the human users, programmatic clients deal rather with the operations, even though they can use the links they receive in the response messages. Also the HTML documentation of RESTful APIs is commonly formulated as a set of operation descriptions. In this section, we specify how to apply our procedure-oriented minimal service model together with RDFa to structure existing RESTful Web service documentation so that key pieces of information are machine-processable.

The HTML documentation of a RESTful Web service consists of one or more Web pages that describe the service in general and its operations in particular. To capture the service descriptions in RDF, the markup of the HTML documentation can be amended with RDFa, which specifies a collection of XML attributes (applicable to any markup language, but especially to XHTML) for expressing RDF data.

To illustrate the RDFa annotations explained in the following text, Listing 3 shows the HTML description from Listing 1, annotated with RDFa as hRESTS data; all the extra markup is highlighted in red.

Listing 3. Example hRESTS$^{\text{RDFa}}$ description

```
 1  <div typeof="hr:Service" about="#svc"
 2           xmlns:hr="http://www.wsmo.org/ns/hrests#"
 3           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 4    <span rel="rdfs:isDefinedBy" resource="" />
 5    <h1><span property="rdfs:label">ACME Hotels</span> service API</h1>
 6    <div rel="hr:hasOperation"><div typeof="hr:Operation" about="#op1">
 7      <h2>Operation <code property="rdfs:label">getHotelDetails</code></h2>
 8      <p>Invoked using the method <span property="hr:hasMethod">GET</span>
 9        at <code property="hr:hasAddress" datatype="hr:URITemplate"
10                    >http://example.com/h/{id}</code><br/>
11        <span rel="hr:hasInputMessage"><span typeof="hr:Message">
12          <strong>Parameters:</strong>
13          <code>id</code> - the identifier of the particular hotel
14        </span></span><br/>
15        <span rel="hr:hasOutputMessage"><span typeof="hr:Message">
16          <strong>Output value:</strong> hotel details in an
17          <code>ex:hotelInformation</code> document
18        </span></span>
19      </p>
20  </div></div></div>
```

First, any portion of the HTML document that describes a given part of the service (an operation, its input our output, or the service as a whole) should be enclosed in a single HTML container element, such as `<body>`, `<p>`, or in a general-purpose block such as `<div>` or `<span>`. In many cases this will already be so; otherwise the annotator can introduce a new container element with minimal effect on the presentation of the HTML document in a Web browser. In the listing, the added container elements are on lines 1–20 (service), 6–20 (operation), 11–14 (input), and 15–18 (output).

These container elements can then be marked with the RDFa `typeof` attribute with the appropriate hRESTS class: `hr:Service` (line 1), `hr:Operation` (line 6), or `hr:Message` (lines 11 and 15). To link a service to its operations, and the operations to their input and output messages, we use the RDF properties `hr:hasOperation`, `hr:hasInputMessage` and `hr:hasOutputMessage` in the RDFa `rel` attribute, as shown on lines 6, 11 and 15.

The duplicate `<div>` and `<span>` container elements on lines 6, 11 and 15 are required to explicitly type operations and messages with the respective hRESTS classes. This type information can also be inferred from the RDFS schema of hRESTS, so the `typeof` annotations (and their associated extra container elements) could potentially be omitted.

While it is not explicit in the hRESTS model in Listing 2, services and operations can usefully carry human-readable labels, using the RDFS property `rdfs:label` in the RDFa `property` attribute used for textual properties (see lines 5 and 7; note how identifying the label on line 5 also needed a `<span>` wrapper element). Additionally, as shown on line 4, it is useful to include an `rdfs:isDefinedBy` link back to the given service's HTML documentation; such a link will allow tools to show the relevant documentation snippets when a user browses the API (this would be similar to how JavaDoc snippets are shown in Java programming IDEs).

A description of an operation can specify the URI template and the method where the operation can be invoked; for this, we use the properties `hr:hasAddress` and `hr:hasMethod` (shown on lines 8 and 9; identifying the method in this example needed another `<span>` wrapper).

Listing 4 shows the RDF data encoded in Listing 3, if that document were available at `http://example.com/api/docs`.

Listing 4. RDF data encoded in Listing 3 (in Turtle syntax)

```
@prefix hr: <http://www.wsmo.org/ns/hrests#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://example.com/api/docs#svc>  a  hr:Service ;
     rdfs:label  "ACME Hotels" ;
     rdfs:isDefinedBy  <http://example.com/api/docs> ;
     hr:hasOperation  <http://example.com/api/docs#op1> .
<http://example.com/api/docs#op1>  a  hr:Operation ;
     rdfs:label  "getHotelDetails" ;
     hr:hasAddress  "http://example.com/h/{id}"^^hr:URITemplate ;
     hr:hasMethod  "GET" ;
     hr:hasInputMessage  [  a  hr:Message  ] ;
     hr:hasOutputMessage  [  a  hr:Message  ] .
```

5. Semantic annotations of hRESTS^{RDFa} descriptions of RESTful APIs

With RDFa, hRESTS structures the HTML documentation of RESTful Web services so it is amenable to machine processing. The RDF model identifies key pieces of information that are already present in the documentation, effectively creating an analogue of WSDL, which is used by messaging (non-RESTful) Web services. hRESTS forms the basis for further extensions, where service descriptions are annotated with added information to facilitate further processing. In this section, we describe an extension towards expressing service semantics.

Because the hRESTS view of services (Section 3) is so similar to that of WSDL, we can adopt SAWSDL [SAWSDL] properties to add semantic annotations. SAWSDL is an extension of WSDL that specifies how to annotate service descriptions with semantic information. It defines the following three XML attributes, along with RDF properties with the same names:

- `modelReference` is used on any component in the service model to point to appropriate *semantic concepts* identified by URIs. SAWSDL speaks about semantic concepts in general, which is not to be confused with the specialized use of the term *concept* in some literature to denote what is called *class* in OWL; a model reference can point to any element of a semantic description.
- `liftingSchemaMapping` and `loweringSchemaMapping` are used to associate messages with appropriate transformations, also identified by URIs, between the underlying technical format such as XML and a semantic knowledge representation format such as RDF.

SAWSDL annotations are URIs that identify semantic concepts and data transformations. Such URIs can be added to the HTML documentation of RESTful services in the form of hypertext links, with `sawsdl:modelReference`, `sawsdl:liftingSchemaMapping` or `sawsdl:loweringSchemaMapping` as the value of the RDFa `rel` attribute, as appropriate. If a clickable link is not appropriate for a particular semantic annotation, another markup element can be used, for example an empty `<span>` with the RDFa attributes `rel` and `resource`.

To illustrate SAWSDL semantic annotations on top of the example in Listing 3, Listing 5 contains two model references into an ontology at `http://example.com/onto.owl` and one link to a lowering mapping; all the extra markup is highlighted in red.

Listing 5. Example hRESTS^{RDFa} description extended with SAWSDL annotations

```
 1   <div typeof="hr:Service" about="#svc"
 2            xmlns:hr="http://www.wsmo.org/ns/hrests#"
 3            xmlns:sawsdl="http://www.w3.org/ns/sawsdl#"
 4            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 5     <span rel="rdfs:isDefinedBy" resource="" />
 6     <h1><span property="rdfs:label">ACME Hotels</span> service API</h1>
 7       <p>This service is a
 8         <a rel="sawsdl:modelReference" href="http://example.com/onto.owl#Hot
 9            hotel reservation</a> service.
10       </p>
11     <div rel="hr:hasOperation"><div typeof="hr:Operation" about="#op1">
12       <h2>Operation <code property="rdfs:label">getHotelDetails</code></h2>
13       <p>Invoked using the method <span property="hr:hasMethod">GET</span>
14         at <code property="hr:hasAddress" datatype="hr:URITemplate"
15                     >http://example.com/h/{id}</code><br/>
16         <span rel="hr:hasInputMessage"><span typeof="hr:Message">
17           <strong>Parameters:</strong>
18           <a rel="sawsdl:modelReference" href="http://example.com/onto.owl#H
19              <code>id</code></a> - the identifier of the particular hotel
20              (<a rel="sawsdl:loweringSchemaMapping"
21                  href="http://example.com/hotel.xsparql">lowering</a>)
22         </span></span><br/>
23         <span rel="hr:hasOutputMessage"><span typeof="hr:Message">
24           <strong>Output value:</strong> hotel details in an
25           <code>ex:hotelInformation</code> document
26         </span></span>
27       </p>
28   </div></div></div>
```

In the listing, the new paragraph on lines 7–10 contains a model reference that associates the service with the concept HotelReservation, which may be in a taxonomy of service functionalities; and the new link on lines 18–19 represents a model reference that defines the `id` parameter to be an instance of the class Hotel.

Lines 20–21 show a link to a lowering transformation. The transformation would presumably map a given instance of the class Hotel into the ID that the service expects as a URI parameter. (The description of concrete data lifting and lowering technologies is out of scope of this specification.)

The SAWSDL properties added in Listing 5 are shown in RDF in Listing 6.

Listing 6. Additional RDF data encoded in Listing 5 (showing only differences from Listing 4; in Turtle syntax)

```
@prefix hr: <http://www.wsmo.org/ns/hrests#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .

<http://example.com/api/docs#svc>
     sawsdl:modelReference <http://example.com/onto.owl#HotelReservation> .

<http://example.com/api/docs#op1>
     hr:hasInputMessage
         [ a hr:Message ;
             sawsdl:loweringSchemaMapping <http://example.com/hotel.xsparql> ;
             sawsdl:modelReference <http://example.com/onto.owl#Hotel>
         ] .
```

6. Acknowledgements

The work on the procedure-oriented minimal service model has been mainly supported by the European Union project SOA4All. The editors would like to thank to all the members of the Conceptual Models for Services Working Group (CMS WG) for their advice and input to this work.

7. References

[Fielding 2001] R. Fielding: *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University Of California, Irvine, 2001, available at http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[Hadley 2009] M. Hadley: *Web Application Description Language*, W3C Member Submission 31 August 2009, available at http://www.w3.org/Submission/wadl/.

[Kopecký et al. 2008] J. Kopecký, K. Gomadam, and T. Vitvar: *hRESTS: An HTML Microformat for Describing RESTful Web Services*, in Proc. of IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), pp. 619-625, 2008.

[RDFa Primer] W3C Semantic Web Deployment Working Group and XHTML2 Working Group: *RDFa Primer — Bridging the Human and Data Webs*, W3C Working Group Note 14 October 2008, available at http://www.w3.org/TR/xhtml-rdfa-primer/.

[RFC 2616] IETF: *Hypertext Transfer Protocol – HTTP/1.1*, IETF RFC 2616, 1999, available at http://rfc.net/rfc2616.html.

[Richardson & Ruby 2007] L. Richardson and S. Ruby: *RESTful Web Services*, O'Reilly Media, May 2007, ISBN 9780596529260.

[SAWSDL] W3C SAWSDL Working Group: *Semantic Annotations for WSDL and XML Schema*, W3C Recommendation 28 August 2007, available at http://www.w3.org/TR/sawsdl/.

[Studer et al. 2007] R. Studer, S. Grimm, and A. Abecker: *Semantic Web Services: Concepts, Technologies, and Applications*, Springer-Verlag New York, Inc. Secaucus, NJ, USA. 2007.

[WebArch] W3C Technical Architecture Group (TAG): *Architecture of the World Wide Web*, W3C Recommendation 15 December 2004, available at http://www.w3.org/TR/webarch/.

[WSDL 2.0] W3C Web Services Description Working Group: *Web Services Description Language (WSDL) Version 2.0*, W3C Recommendation 26 June 2007, available at http://www.w3.org/TR/wsdl20/.