SEMANTIC WEB SERVICES: STATE OF THE ART

Markus Lanthaler[1, 5], Michael Granitzer[2, 3] and Christian Gütl[1, 4]

[1] Instit. for Information Systems and Computer Media/[2] Instit. of Knowledge Management, Graz University of Technology
[3]Know-Center GmbH Graz - Graz, Austria
[4] School of Information Systems/[5] Digital Ecosystems and Business Intelligence Institute, Curtin University of Technology Perth, Australia

ABSTRACT

Service-oriented architectures (SOA) built on Web services were a first attempt to streamline and automate business processes in order to increase productivity but the utopian promise of uniform service interface standards, metadata, and universal service registries, in the form of the SOAP, WSDL and UDDI standards has proven elusive. Furthermore, the RPC-oriented model of those traditional Web services is not Web-friendly. Thus more and more prominent Web service providers opted to expose their services based on the REST architectural style. Nevertheless there are still problems on formal describing, finding, and orchestrating RESTful services. While there are already a number of different approaches none so far has managed to break out of its academic confines. This paper focuses on an extensive survey comparing the existing state-of-the-art technologies for semantically annotated Web services as a first step towards a proposal designed specifically for RESTful services.

KEYWORDS

Semantic Web; Web services; REST; SOA

1. INTRODUCTION

Service-oriented architectures (SOA) built on Web services were a first attempt to streamline and automate business processes in order to increase productivity. Thus, more and more organizations offer access to their information through Web services. But, while most current and previous research efforts mostly concentrate on traditional SOAP-based Web services, the utopian promise of uniform service interface standards, meta-data, and universal service registries, in the form of the SOAP, WSDL, and UDDI standards has proven elusive. Thus, the usage of SOAP-based services is mainly limited to the integration of legacy systems which have not been built to be Web-friendly. Instead of SOAP-based services with their high perceived complexity, prominent Web service providers like Microsoft, Google, Yahoo, and others have opted to use lightweight REST-style APIs.

REST is an architectural style developed specifically for the Internet that specifies constraints to enhance performance, scalability, and resource abstraction within distributed hypermedia systems(Fielding 2000), (Lanthaler and Guetl 2010). But, despite the foreseeable potential, the increasing interest on, and growing acceptance of lightweight services, there are still problems on formal describing, finding and orchestrating RESTful services. Research on these issues has already started but none of the approaches so far has managed to break out of its academic confines.

The lack of a widely accepted standard to create semantic RESTful services for various application domains and scenarios motivated us to research aspects of a holistic service description format for REST-based services. This paper in particular focuses on an extensive survey comparing the existing state-of-the-art technologies for semantically annotated Web services as a first step towards a proposal specifically designed for RESTful services.

The remainder of this paper is organized as follows. First we highlight REST's main advantages. Then we give an overview of different proposals for service interface description formats in section 3 and proposals for semantic annotation of (RESTful) Web services in section 4. Those approaches are then compared in section 5 to more pragmatic and widely accepted domain specific description formats. Finally, the concluding remarks are presented in section 6.

2. REPRESENTATIONAL STATE TRANSFER (REST)

Even though many successful distributed systems have been built on RPC and RPC-oriented technologies, such as SOAP, it is known for quite some time (Waldo et al. 1994) that this approach is flawed because it ignores the differences between local and remote computing. The major differences concern the areas of latency, memory access, partial failure and concurrency as described in detail in (Waldo et al. 1994). In Internet-scale systems intermediaries for caching, filtering, monitoring or, e.g., logging are “must haves” to ensure good performance, scalability, and maintainability. Fielding (2000) made meticulously chosen trade-offs for REST which address exactly those issues and allow building extensible, manageable, maintainable, and loosely-coupled distributed systems at Internet-scale. In REST caching, e.g., is relatively straightforward: clients retrieve data by (conditional) GET requests and servers can specify the cache validity duration by HTTP Cache-Control headers. This clearly follows HTTP’s semantics and doesn’t break any intermediaries relying on those semantics. The fact that the whole Web—the largest and most successful distributed system—is built on the REST principles should be evidence enough of REST’s superior scalability and interoperability.

The REST architectural style (Fielding 2000) is based on a client-server architectural style where the communication is stateless such that each request from client to server must contain all of the information necessary to understand the request. It cannot take advantage of any stored context on the server; the session state is kept entirely on the client. To mitigate the overhead caused by this statelessness, cache constraints, requiring that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable, have been added. To further improve scalability, the layered system constraint, that restricts the knowledge of the system to a single layer, has been added. But the central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components. REST is defined by four interface constraints (Fielding 2000): 1) identification of resources, 2) manipulation of resources through representations, 3) self-descriptive messages, and 4) hypermedia as the engine of application state. Finally there exists an optional constraint which allows client functionality to be extended by downloading and executing code in the form of applets or scripts on demand to simplify clients and improve extensibility. Since it reduces visibility it is only an optional constraint within REST.

REST’s “identification of resources” constraint, which specifies that every resource has to be addressable, makes REST a natural fit for the vision of the Semantic Web (Berners-Lee, Hendler and Lassila 2001) and creates a network of Linked Data (Bizer, Heath and Berners-Lee 2009); no parallel exists for SOAP’s remote method invocation. It follows that REST-based Web services are an ideal carrier for semantic data and would even provide the additional benefit of resource resolvability in human-readable HTML.

2.1 A Word on the HATEOAS Constraint

Unfortunately the *hypermedia as the engine of application state* (HATEOAS) constraint, which refers to the use of hyperlinks in resource representations as a way of navigating the state machine of an application, is one of the least understood constraints, and thus seldom implemented correctly. It is exactly that rule that allows building reliable loosely coupled systems.

In contrast, most of the time, SOAP-based systems rely heavily on implicit state-control flow control. The allowed messages and how they have to be interpreted depends on what messages have been exchanged before and thus in which implicit state the system is. Third parties or intermediaries trying to interpret the conversation need the full state transition table and the initial state to understand the communication. This in turn implies that states and transitions between them have to be identifiable which demands (complex) technologies like Web Services Business Process Execution Language (WS-BPEL).

According to Fielding (2008), “a REST API should be entered with no prior knowledge beyond the initial URI (bookmark) and set of standardized media types. […] From that point on, all application state transitions must be driven by client selection of server-provided choices that are present in the received representations or implied by the user’s manipulation of those representations.”

The “human Web” is obviously based on this type of interaction and state-control flow where very little is known a priori. Humans are able to quickly adapt to such new control flows (e.g. a change in the order sequence or a new login page to access the service). On the other hand, machine-to-machine communication is often based on static knowledge and tight coupling. The challenge is thus to bring some of the human Web’s adaptivity to the Web of the machines to allow the building of loosely coupled, reliable, and scalable systems.

After all, a Web service can be seen as a special Web page meant to be consumed by an autonomous program as opposed to a human being. The Web already supports machine-to-machine communication, what's not machine-processable about the current Web is not the protocol (HTTP), it is the content.

3. SERVICE INTERFACE DESCRIPTION

As outlined in an earlier paper (Lanthaler and Guetl 2010), there has to be an agreement or contract on the used interfaces and data formats in order for two (or more) systems to communicate successfully. In the traditional Remote Procedure Call (RPC) model, where all differences between local and distributed computing are hidden, usually static contracts in the form of an Interface Description Language (IDL) are used to specify those interfaces. The data types that such an IDL offers are abstractions of the data types found in actual programming languages to allow interoperability between different platforms. In SOAP this is usually done by using WSDL and XML Schema. That way, automatic code generation on both, the client and the server side, are possible.

In contrast REST's HATEOAS constraint is characterized by the use of contextual contracts where the set of actions varies over. Additionally the interface variability is almost eliminated due to REST's uniform interface. In consequence REST-based services are almost exclusively described by human-readable documentation describing the URLs and the data expected as input and as output. Even though it would be possible to describe REST services with WSDL 2.0 different other approaches have been proposed. Most of them, such as WRDL (Prescod 2002), NSDL (Walsh 2005), SMEX-D (Bray 2005), Resedel (Cowan 2005), RSWS (Salz 2003), and WDL (Orchard 2010) were more or less ad-hoc inventions designed to solve particular problems and haven't been updated for many years. The most recent, respectively only regularly updated proposals are, to our best knowledge, hRESTS (HTML for RESTful Services) (Kopecký, Gomadam and Vitvar 2008) and WADL (Web Application Description Language) (Hadley 2009).

Given REST's constraints, it is arguable whether REST even needs a service interface description. In contrast to the before mentioned approaches we argue that the description of the resource representations, i.e., the transport format, combined with the use of hypermedia should be enough to achieve a high degree of automation for RESTful services. Nevertheless, for the sake of completeness, the following two sections provide a short overview of WADL and hRESTS.

3.1 WADL (Web Application Description Language)

WADL's approach (Hadley 2009) is closely related to WSDL by generating a monolithic XML file containing all the information about the service interface. Given that it was specifically designed for describing RESTful services, it models the resources provided by the service and the relationships between them, in contrast to WSDL's operation-based manner. In WADL each service resource is described as a request containing the used HTTP method and the required inputs as well as a response describing the expected service response representation and HTTP status code.

The main critique of WADL is that it is complex and thus requires developers that have a certain level of training and tool support to enable the usage of WADL. This complexity contradicts the simplicity of RESTful services. In addition, WADL urges the use of specific resource hierarchies which introduce an obvious coupling of the client and server. Servers should have the complete freedom to control their own namespace. In consequence, WADL currently is not widely used.

3.2 hRESTS (HTML for RESTful Services)

hRESTS' (Kopecký, Gomadam and Vitvar 2008) idea is to enrich the, mostly already existent, human-readable documentation with so called microformats (Khare and Çelik 2006) to make it machine-processable. While it offers a relatively straightforward solution to describe the resources and the supported methods, there is some lack of support for describing the used data schemas. Apart from a potential label, hRESTS does not provide any support for further machine-readable information about the inputs and outputs. Extensions like SA-REST (Sheth, Gomadam and Lathem 2007) and MicroWSMO (Kopecký and Vitvar 2008) address this issue. More information about those extensions can be found in section 0 and 0.

4. SEMANTIC ANNOTATION OF (RESTFUL) SERVICES

Most of the time, the syntactic description of a service's interface is not enough. Indeed, two services can have the same syntactic definition but perform significantly different functions. Thus, also the semantics of the data and the behavior of the service have to be documented and understood. This is normally done in the form of a textual description which is, hopefully, easily understandable by a human being. Machines, on the other hand, have huge problems to understand such a document and cannot extract enough information to use such a service in a semantic correct way. To address this problem the services have to be annotated semantically; the resulting service is called a Semantic Web Service (SWS) or a Semantic RESTful Service (SRS). Those supplemental semantic descriptions of the service's properties can in consequence lead to higher level of automation for tasks like discovery, negotiation, composition, and invocation. Since most SWS technologies use ontologies as the underlying data model they also provide means for tackling the interoperability problem at the semantic level and, more importantly, enable the integration of Web services within the Semantic Web.

4.1 OWL-S

OWL-S (Web Ontology Language for Web Services) ('OWL-S: Semantic Markup for Web Services' 2004) is an upper ontology based on the W3C standard ontology OWL used to semantically annotate Web services. OWL-S consists of the following main upper ontologies: 1) the *Service Profile* for advertising and discovering services; 2) the *Service (Process) Model*, which gives a detailed description of a service's operation and describes the composition (choreography and orchestration) of one or more services; and 3) the *Service Grounding*, which provides the needed details about transport protocols to invoke the service (e.g. the binding between the logic-based service description and the service's WSDL description). Generally speaking, the Service Profile provides the information needed for an agent to discover a service, while the Service Model and Service Grounding, taken together, provide enough information for an agent to make use of a service, once found ('OWL-S: Semantic Markup for Web Services' 2004).

The main critique of OWL-S is its limited expressiveness of service descriptions in practice. Since it practically corresponds to OWL-DL it allows only the description of static and deterministic aspects; it does not cover any notion of time and change, nor uncertainty. Besides that, in contrast to WSDL, an OWL-S process cannot contain any number of completely unrelated operations (Klusch 2008, Lara et al. 2004).

4.2 WSMO

Another approach to describe Web services semantically is the Web Service Modeling Ontology (WSMO) (Roman 2005). It defines a conceptual model and a formal language called WSML (Web Service Modeling Language) together with a reference implementation of an execution environment (WSMX; Web Service Execution Environment) for the dynamic discovery, selection, mediation, invocation, and interoperation of Semantic Web services based on the WSMO specification.

WSMO offers four top-level notions to describe the different aspects of Web services: 1) *Ontologies* that define the formalized domain knowledge; 2) *Goals*, which specify objectives that a client might have when consulting a Web service; 3) *Service Descriptions* for describing services that are requested by service requesters, provided by service providers, and agreed between service providers and requesters; and 4) *Mediators* for enabling interoperability and handling heterogeneity between all these components at data (mediation of data structures) and process level (mediation between heterogeneous communication patterns) to allow loose coupling between services, goals, and ontologies.

In contrast to most other description formalisms, WSMO propagates a goal-based approach for SWS. It is particularly designed to allow the search for Web services by formulating the queries in terms of goals. So the task of the system is to automatically find and execute Web services which satisfy the client's goal. This goes beyond of OWL-S' idea whose principal aim is to describe the service's offers and needs.

One of the main critiques of WMO has been that its development has been done in isolation of the W3C standards. This raised serious concerns by the W3C which were expressed in the official response to the WSMO submission in 2005 (Bournez 2005). To address those issues, a lightweight version called WSMO-Lite (Vitvar et al. 2008) has been created; see section 0 for details. Another critique is that guidelines for developing mediators, which seem to be the essential contribution of WSMO in concrete terms, are missing.

4.3 SAWSDL

After number of efforts (including the above mentioned OWL-S ('OWL-S: Semantic Markup for Web Services' 2004) and WSMO (Roman 2005)) semantic annotation of SOAP-based services is now preferably addressed by the W3C recommendation Semantic Annotations for WSDL and XML Schema (SAWSDL) (2007). SAWSDL defines how to add semantic annotations to various parts of a WSDL document such as inputs, outputs, interfaces, and operations.

However, SAWSDL does not specify a language for representing the semantic models. Instead, it just defines how semantic annotation is accomplished using references to semantic models, e.g. ontologies, by providing three new extensibility attributes to WSDL and XML Schema elements.

The *modelReference* extension attribute defines the association between a WSDL or XML Schema component and a concept in some semantic model. It is used to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. The other two extension attributes, named *liftingSchemaMapping* and *loweringSchemaMapping*, are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML. SAWSDL allows multiple semantic annotations to be associated with WSDL elements. Schema mappings as well as model references can contain multiple pointers. Multiple schema mappings are interpreted as alternatives whereas multiple model references all apply. SAWSDL does not specify any other relationship between them ('Semantic Annotations for WSDL and XML Schema (SAWSDL)' 2007).

The major critique of SAWSDL is that it comes without any formal semantics. This hinders logic-based discovery and composition of Web services described with SAWSDL but calls for "magic mediators outside the framework to resolve the semantic heterogeneities. (Klusch 2008)

4.4 WSMO-Lite

As already mentioned in the previous section, SAWSDL does not specify a language for representing the semantic models but just defines how to add semantic annotations to various parts of a WSDL document. WSMO-Lite (Vitvar et al. 2008) has been created as a lightweight service ontology to fill the SAWSDL annotations with concrete service semantics to allow bottom-up modeling of services. It adopts the WSMO model and makes its semantics lighter. The biggest differences are that WSMO-Lite treats mediators as infrastructure elements and specifications for user goals as dependent on the particular discovery mechanism used, while WSMO defines formal user goals and mediators. Furthermore, WSMO-Lite defines the behavior semantics only implicitly. WMO-Lite also does not exclusively use WSML, as WSMO does, but allows the use of any ontology language with a RDF-syntax.

WSMO-Lite describes the following four aspects of a Web service: 1) the *Information Model*, which defines the data model for input, output, and fault messages; 2) the *Functional Semantics*, which define the functionality, which the service offers; 3) the *Behavioral Semantics*, which define how a client has to talk to the service; and 4) the *Non-functional Descriptions*, which define non-functional properties such as quality of service or price.

A major advantage of the WSMO-Lite approach is that it is not bound to a particular service description format, e.g., WSDL. As a result WSMO-Lite can be used to integrate approaches like, e.g., hRESTS (in conjunction with MicroWSMO) with the traditional WSDL-based service descriptions.

4.5 MicroWSMO

MicroWSMO is an attempt to adapt the SAWSDL approach for the semantic description of RESTful services. It uses, just as hRESTS, on which it relies, microformats for adding semantic annotations to the HTML service documentation.

Similar to SAWSDL, MicroWSMO has three types of annotations: 1) *Model*, which can be used on any hRESTS service property to point to appropriate semantic concepts; 2) *Lifting*, and 3) *Lowering*, which specify the mappings between semantic data and the underlying technical format such as XML. Therefore, MicroWSMO enables the semantic annotation of RESTful services basically in the same way in which SAWSL supports the annotation of Web services described by WSDL.

It is important to point out that, since both MicroWSMO and SAWSDL can apply WSMO-Lite service semantics, REST-based services can be integrated with WSDL-based ones (Maleshkova, Kopecký and Pedrinaci 2009). Therefore, tasks such as discovery, composition, and mediation can be performed completely independently from the underlying Web service technology.

4.6 SA-REST

Another approach for the semantic description of RESTful services is SA-REST (Sheth, Gomadam and Lathem 2007). It relies on RDFa for marking service properties in an existing HTML service description, similar to hRESTS with MicroWSMO. As a matter of fact it was the first approach reusing the already existing HTML service documentation to create machine-processable descriptions of RESTful services. The main differences between the two approaches are indeed not the underlying principles but rather the implementation technique.

SA-REST offers the following service elements: 1) *Input* and 2) *Output* to facilitate data mediation*;* 3) *Lifting* and 4) *Lowering* schemas to translate the data structures that represent the inputs and outputs to the data structure of the ontology, the grounding schema*; 5*) *Action*, which specifies the required HTTP method to invoke the service; 6) *Operation* which defines what the service does; and 7) *Fault* to annotate errors.

5. DOMAIN SPECIFIC DESCRIPTIONS

All the above mentioned approaches try to be as general as possible to allow the creation of suitable service annotations for a wide range of application domains. In contrast to that, there exist a number of service description formats such as OpenSearch (Clinton 2005) and the Atom Publishing Protocol which are tailored for very specific application domains. The semantics of those descriptions are implicit, i.e., all services described by such an approach are created for the same or very similar use cases.

In this chapter the two probably most successful domain specific description formats, Atom and OpenSearch, are described.

5.1 Atom

The Atom suite consists of two related standards: the Atom Syndication Format and the Atom Publishing Protocol (also known as AtomPub or APP). The Atom Syndication Format is a XML-based format used by publishers to syndicate their content to providing users with frequently updated content in the form of so called Web feeds or news feeds. The Atom Publishing Protocol is an application-level protocol for publishing, editing, and deleting Web resources. Atom is designed to be an extensible format and so foreign markup (markup which is not part of the Atom vocabulary) is allowed almost anywhere in an Atom document.

Both, the Atom Syndication Format as well as the Atom Publishing Protocol, are fully based on the REST architectural style and thus extremely Web-friendly. This, and the above mentioned extensibility, led to the adoption of AtomPub for the implementation of various kinds of Web services. The most prominent examples might be the Google Data Protocol (GData) (2010) and Microsoft's Open Data Protocol (Odata) (2010). They use Atom's extensibility to implement APIs for their services. It is thus arguable that Atom is one of the world's most successful RESTful Web service stories.

5.2 OpenSearch

OpenSearch (Clinton 2005) was developed by A9, an Amazon.com subsidiary, and was first unveiled in 2005. It is a collection of simple formats that allow the interface description of search engines and publishing of search results in a format suitable for syndication and aggregation. OpenSearch thus allows search clients, such as Web browsers, to invoke search queries and process the responses. By now all major Web browsers support OpenSearch and use it to add new search engines to the browser's search bar. This way the user can invoke a query directly from the browser without having to load the search engine's homepage first.

OpenSearch consists of the following four formats: 1) the description document, 2) the URL template syntax, 3) the response elements, and 4) the Query element. The OpenSearch description document describes the web interface of a search engine in the form of a simple XML document. It may also contain some metadata such as the name of the search engine and its developer. The URL template syntax represents a parameterized form of the URL by which a search engine is queried. Simply speaking it describes the used GET parameters to invoke a query. An example of a such a template looks as follows: *http://example.com/search?q={searchTerms}*. All parameters are enclosed in curly braces and are by default considered to be part of the OpenSearch template namespace. By using the XML namespace prefix conventions it is possible to add new parameter names, which brings extensibility. The OpenSearch response elements are used by search engines to augment existing XML formats such as Atom and RSS with search-related metadata. Finally, the OpenSearch Query element can be used to define specific search requests that can be performed by a search client. The Query element attributes correspond to the search parameters in a URL template. One use case is, e.g., the definition of related queries in a search result element.

6. CONCLUSIONS AND FUTURE WORK

The attempt to standardize Web services has taken years, but still there are no clear definitions of what constitutes a service at a conceptual level. While a number of different approaches, such as OWL-S, WSMO, WSMO-Lite, MicroWSMO, SAWSDL, and SA-REST as described in the previous sections, have been proposed, none so far has managed to break out of its academic confines. Looking at REST-based services the situation looks even worse; there does not even exists a widely accepted standard for the (syntactic) description of a service's interface respectively a machine-readable description of the resource representations.

We argue that the lack of acceptance of those approaches stems from the fact that they do not provide any imminent incentive and thus experience a classic chicken-and-egg problem. No services are semantically described because there are no applications making use of that information and no applications are developed because there are no semantically annotated services. Facebook's recently introduced Open Graph Protocol (2010) clearly shows the willingness of Web site publishers to semantically annotate their content if it's easy enough and if there is an imminent incentive to do so. More than 50,000 sites implemented the protocol within the first week of its publication (Huang 2010). Another factor for the lacking acceptance is the high complexity of most of the approaches. Often their functioning resembles the flawed RPC-model (remote procedure call) which is a problem especially with regard to RESTful services, whose approach is fundamentally different.

To solve those and other problems, we are currently working on an approach which tries to create the machine counterpart of the human Web (as described in section 0) by combining different proven technologies. By basing the approach on well-known technologies we hope to lower the barrier for developers to develop scalable semantic Web services. The most challenging part is how to communicate the domain knowledge without overburdening developers. We aim to develop a prototype and investigate how usable the approach is to model complex Web services. We also aim to improve and investigate methods that enable discovery of services with minimal complexity.

ACKNOWLEDGEMENT

The Know-Center GmbH Graz is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.

REFERENCES

Berners-Lee, T., Hendler, J. and Lassila, O. 2001, “The Semantic Web”. *Scientific American*, vol. 284, no. 5, pp. 34-43.

Bournez, C. 2005, “Team Comment on Web Service Modeling Ontology (WSMO) Submission”. *W3C Submissions*, viewed 23 June 23 2010, <http://www.w3.org/Submission/2005/06/Comment.html>

Bray, T. 2005, “SMEX-D (Simple Message Exchange Descriptor)”, <http://www.tbray.org/ongoing/When/200x/2005/05/03/SMEX-D>

C. Bizer, C., Heath, T. and Berners-Lee, T. 2009. “Linked Data - The Story So Far”. *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 3, pp. 1-22.

Clinton, D. 2005, OpenSearch 1.1 Draft 4, <http://www.opensearch.org/Specifications/OpenSearch/1.1>

Cowan, J. 2005, “Resedel”, <http://recycledknowledge.blogspot.com/2005/05/resedel.html>

Fielding, R. T. 2008, “REST APIs must be hypertext-driven”. *Untangled musings of Roy T. Fielding*, <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

Fielding, R. T. 2000, “Architectural Styles and the Design of Network-based Software Architectures”, PhD thesis, Department of Information and Computer Science, University of California, Irvine, USA, 2000.

Google Data Protocol, 2010, viewed 5 July 2010 <http://code.google.com/apis/gdata/>.

Hadley, M. J. 2009, “Web Application Description Language (WADL)” <http://www.w3.org/Submission/2009/SUBM-wadl-20090831/>

Huang, S. L. 2010, “After f8 - Resources for Building the Personalized Web”. *Facebook Developer Blog*, viewed 7 July 2010, <http://developers.facebook.com/blog/post/379>

Khare, R. and Çelik, T. 2006, *Microformats: A Pragmatic Path to the Semantic Web*, CommerceNet Labs, Palo Alto, CA, USA, Tech. Rep. 06-01, Jan. 2006, < http://wiki.commerce.net/images/e/ea/CN-TR-06-01.pdf>

Klusch, M. 2008, “Semantic Web Service Description”. In M. Schumacher, H. Schuldt, & H. Helin, CASCOM: Intelligent Service Coordination in the Semantic Web (pp. 31-57). Basel: Birkhäuser Basel.

Kopecký, J. and Vitvar, T. 2008, “D38v0.1 MicroWSMO: Semantic Description of RESTful Services”, <http://wsmo.org/TR/d38/v0.1/20080219/d38v01_20080219.pdf>

Kopecký, J., Gomadam, K. and Vitvar, T. 2008, “hRESTS: an HTML Microformat for Describing RESTfulWeb Services”. *Proc. 2008 IEEE/WIC/ACM Int. Conf.on Web Intelligence and Intelligent Agent Technology*, vol. 1, pp. 619-625.

Lanthaler, M. and Guetl, C. 2010, “Towards a RESTful Service Ecosystem - Perspectives and Challenges”. *Proceedings of the 2010 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST),* Dubai, UAE.

Lara, R., Roman, D., Polleres, A., Fensel, D. 2004, “A Conceptual Comparison of WSMO and OWL-S”. *European Conference on Web Services (ECOWS 2004),* Erfurt, Germany, pp. 254-269.

Maleshkova, M., Kopecký, J. and Pedrinaci, C. 2009, “Adapting SAWSDL for Semantic Annotations of RESTful Services”, *LNCS 5872*, pp. 917-926.

Open Data Protocol, 2010. Viewed 5 July 2010, <http://www.odata.org/>

Open Graph Protocol, 2010. Viewed 7 July 2010, <http://opengraphprotocol.org/>

Orchard, D. 2010, “Web Description Language (WDL)”, viewed 7 January 2010, <http://www.pacificspirit.com/Authoring/WDL>

OWL-S: Semantic Markup for Web Services, W3C Member Submission, <http://www.w3.org/Submission/OWL-S/>

Prescod, P. 2002, “Web Resource Description Language (‘Word-dul’)”, <http://www.prescod.net/rest/wrdl/wrdl.html>

Roman, D., Keller, U., Lausen, H., & Bruijn, J. D. 2005, “Web Service Modeling Ontology”, *Applied Ontology*, vol. 1, issue 1, pp. 77-106.

Salz, R. 2003, “Really Simple Web Service Descriptions”, <http://webservices.xml.com/pub/a/ws/2003/10/14/salz.html>

Semantic Annotations for WSDL and XML Schema (SAWSDL), 2007, W3C Recommendation.

Sheth, A. P., Gomadam, K. and Lathem, J. 2007, “SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups”, *IEEE Internet Computing*, vol. 11, issue 6, pp. 84-87.

Vitvar, T., Kopeck, J., Viskova, J., & Fensel, D. 2008, “WSMO-Lite Annotations for Web Services”, *Proceedings of the 5th European Semantic Web Conference (ESWC 2008),* LNCS 5021, pp. 674-689. Tenerife, Spain.

Waldo, J., Wyant, G., Wollrath, A. and Kendall, S. 1994, *A Note on Distributed Computing*, SMLI TR-94-29, Sun Microsystems Laboratories, Mountain View, CA, USA.

Walsh, N. 2005, “WITW: NSDL”, <http://norman.walsh.name/2005/03/12/nsdl>