4th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2010)

Towards a RESTful Service Ecosystem

Perspectives and Challenges

Markus Lanthaler [1]
[1] Institute for Information Systems and Computer Media
Graz University of Technology
Graz, Austria

Christian Gütl [1,2]
[2] School of Information Systems
Curtin University of Technology
Perth, Australia

Abstract—**Average information workers spend most of their time for searching, analyzing, reformatting and consolidating information. The recent advent of service-oriented architectures (SOA) built on Web services is a first attempt to streamline respectively automate those tasks in order to increase productivity. SOAP-based services work well within a company and are thus mainly used to for the integration of legacy systems which have not been built to be Web-friendly or to make new systems more flexible for changing requirements in business ecosystems. Nevertheless, the utopian promise of uniform service interface standards, metadata and universal service registries, in the form of the SOAP, WSDL and UDDI standards have proven elusive. Instead, for Internet-scale applications, lightweight REST-based architectures which gained a lot of momentum recently provide a number of important advantages such as better scalability, reliability and visibility and are thus the preferred choice for Internet-scale applications. Despite the foreseeable potential, the increasing interest on and growing acceptance of lightweight services, there are still problems on formal describing, finding and orchestrating services as well as a lack of a holistic framework covering the entire service lifecycle. This paper focuses on an extensive survey comparing the traditional SOAP-based architecture to the emergent lightweight REST-based architectural style as a first step towards a framework proposal.**

Index Terms—Autonomic computing, Internet, Web services, semantic Web services, service discovery, service composition, service orchestration and choreography, Web 2.0, Web 3.0

I. INTRODUCTION

The World Wide Web has liberated information from its physical containers such as books, journals and newspapers allowing information to flow faster and more independently. This led to tremendous progress in information creation, distribution and usage which resulted in huge productivity gains. Nevertheless, according to Feldman et al. [1], average information workers spend roughly a quarter of their time searching for information and another quarter analyzing it. Every week they waste about 3.8 hours reformatting from multiple formats into one document format and about 3.5 hours searching for information never found causing costs of millions of dollars. This affects application domains such as decision support, knowledge acquisition and management activities as well as learning and training settings. A great part of this problem is due to the fact that, while most information is already stored in a structured form inside databases on the Web, it is still flattened out for presentation, segmented into "pages", and aggregated into separate "sites"; many services remain isolated islands in the huge information sea of the Web. So, just as the first automobiles looked like horse carriages, reflecting outdated assumptions about the way they would be used, information resources on the Web still resemble their physical predecessors [2].

The recent advent of service-oriented architectures (SOA) built on Web services is a first attempt to streamline or automate business processes in order to increase productivity. More and more organizations offer access to their information through Web services. By composing different Web services, it is even possible to create value-added services or new applications to provide functionalities that were not available or defined at design time.

But, while most current and previous research efforts mostly concentrate on SOAP-based services, 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. Thus, the usage of SOAP-based services is traditionally mainly limited to the integration of legacy systems which have not been built to be Web-friendly or to make new systems more flexible for changing business processes within a company which helps organizations to cope with changing requirements in business ecosystems. Instead of SOAP-based services with their high perceived complexity, prominent Web service providers like Microsoft, Google, Yahoo, Amazon, Sun and eBay have opted to use lightweight protocols like RSS and ATOM to push data to consumers, while exposing their service provisions as simple and lightweight REST-style APIs. Some of them even started to retire their SOAP-based services and replace them with REST services [3].

Lightweight services are easier to consume, are more often used to provide services across organizational borders and are interesting for community-driven services. REST is an architectural style that specifies constraints to enhance performance, scalability, and resource abstraction within distributed hypermedia systems. It provides the same interface to access all resources. Indeed the whole Web is a REST-based system and can be seen as an expansive application framework that proves REST's constraints and effectiveness [4]. By providing data in a form easy to process, tedious and error prone tasks such as consolidating and reformatting data which cost millions of dollars [1] could be completely eliminated or at least minimized.

Despite the foreseeable potential, the increasing interest on and growing acceptance of lightweight services, there are still problems on formal describing, finding and orchestrating as well as trust and reliability issues in creating and providing business services. Research and initiatives on standardizations of these issues have already started but no agreed standards exist yet. There is also a lack of a holistic framework for practical usage of lightweight Web services which should cover the entire lifecycle beginning at the description, the semantic annotation, the application, the discovery of suitable services and finally the composition of services to build mashups.

The lack of such a holistic framework and practical guidelines for usage applicable to various application domains and scenarios motivated us to focus on lightweight services in order to research aspects of a holistic REST-based framework. This paper in particular focuses on a first survey by comparing heavyweight and lightweight approaches on all aspects of the aforementioned lifecycle as a first step towards a framework proposal.

To this end, the remainder of this paper is organized as follow. Section II discusses the pros and cons of service interface descriptions and presents the latest proposals for REST-based services. Section III presents different approaches for the semantic annotation of the service's descriptions as well as their input and output data and shows how a parallel to the SOAP stack can be built for REST-based services. Section IV covers the usage of Web services and shows areas where REST-based services are more powerful and why they are Web-friendlier. Section V addresses service discovery and composition. Finally, the concluding remarks are presented in section VI.

II. Service Interface Description

In order for two (or more) systems to communicate, there has to be an agreement or contract on the used interfaces and data formats. In the traditional Remote Procedure Call (RPC) model, where all differences between local and distributed computing are hidden, usually an Interface Description Language (IDL) is 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. That way, automatic code generation on both, the client and the server side, are possible.

SOAP, the successor of XML-RPC [5], is based on exactly the same model. SOAP service interfaces are usually defined by a Web Service Description Language (WSDL) file to describe the methods with their expected inputs and outputs and XML Schema(s) to describe the schemas for those inputs and outputs. Given that those documents are machine-readable, it is possible to automatically generate code stubs which aim to improve the developer's productivity. But sometimes this causes several interoperability problems due to the impedance mismatch between XML Schemas (XSD) and object oriented programming constructs (O/X impedance mismatch). The XML Schema language has a number of type system constructs which simply do not exist in commonly used object oriented programming languages such as, e.g., Java [5]. In consequence, this leads to interoperability problems because each SOAP stack has its own way of mapping the various XSD type system constructs to objects in the target platform's programming language and vice versa.

In contrast, 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 [6], [7]. This obviously renders automatic code generation or automatic validation of requests and responses impossible.

There have been very controversial discussions whether REST even needs machine-readable interface descriptions. Good APIs expose "hackable" URLs [8] which at least human users can easily understand and in consequence modify parts of if to retrieve the desired content. Another fact is that, due to REST's uniform interface [4], the interface variability is almost eliminated. REST uses the HTTP verbs to apply CRUD operations (Create, Read, Update and Delete) to resources defined by their URI. What most parties agree on is that something simpler than WSDL is needed, or, to say it with Norman Walsh's words: "I think something dramatically simpler than WSDL could get the job done most of the time. We know the hard things are possible, we just have to make the easy things easy." [9]

A natural way to describe the interface of a REST service is to use HTML containing hyperlinks and forms—with HTML5 forms also the HTTP-methods PUT and DELETE will get supported [10]. The drawback of that solution is the inflexibility for the request data type and the lack of a definition of possible response data types.

To solve those issues, different approaches have been proposed. Most of them, such as WRDL [11], NSDL [12], SMEX-D [13], Resedel [14], RSWS [15] and WDL [16] were more or less ad hoc inventions designed to solve particular problems and haven't been updated since some years. The most recent, respectively only regularly updated proposals are, to our best knowledge, hRESTS (HTML for RESTful Services) [17] and WADL (Web Application Description Language) [18].

WADL's approach is closely related to WSDL by generating a monolithic file containing all the information about the service interface while the idea of hRESTS is to enrich the, mostly already existent, human-readable documentation with so called microformats [19] to make it machine-processable. Both offer, as most of the other mentioned proposals, a relatively straightforward solution to describe the resources and the supported methods; however, there is some lack of support when describing the used data schemas. WADL relies on Internet Media Types and optional XML or RelaxNG schemas in contrast to hRESTS that, apart from a potential label, does not provide any support for further machine-readable information about the inputs and outputs. Extensions like SA-REST [20] and MicroWSMO [21] address this issue. More information can be found in section III.

The use of Internet Media Types has the benefit that the restriction that the payload has to be an XML document is removed. One of the fundamental design decisions for SOAP-based Web services was that all exchanged data must either be

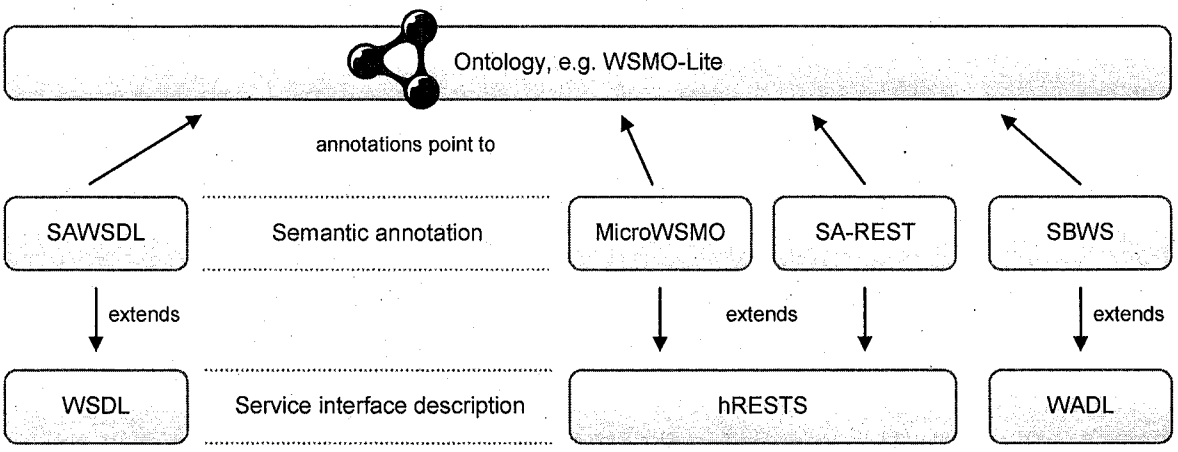

Figure 1. Comparing the SOAP- and REST-based Web service stack.

an XML document or modelled as an XML document. This led to the development of numerous schemes like SOAP with Attachments (SwA), Direct Internet Message Encapsulation (DIME), WS-Attachments, Message Transmission Optimization Mechanism (MTOM) and XML-binary Optimized Packaging (XOP) to support the transport of non-XML data.

Even though interface description languages such as hRESTS and WADL allow automatic code generation, is has to be made sure that developers do not fall in the "RPC trap". Developers should at any point be aware whether local or remote resources are accessed in order to threat the differences accordingly; otherwise there is an imminent danger of significantly reduced scale, greater client-server coupling and more difficult system modification and maintenance ([22]-[24]).

III. SEMANTIC ANNOTATION

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 behaviour 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 automatically. 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.

There are basically four types of service semantics: 1) functional semantics describing what the services does; 2) behavioural semantics defining how a client talks to the service; 3) information model semantics specifying the exchanged data (incl. lifting/lowering to the grounding schema, i.e., the data structure of the ontology); and 4) non-functional descriptions like policies, QoS, price, location and more. WSMO-Lite [25], e.g., is one of the ontologies specifying all of the above mentioned aspects of service semantics.

In SOAP-based services semantic annotation is now, after number of efforts including OWL-S [26], WSMO [27] and WSDL-S [28], preferably addressed by the W3C recommendation Semantic Annotations for WSDL and XML Schema (SAWSDL) [29]. 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. A summary of the extension attributes defined by SAWSDL is given below (taken from [29]):

- An extension attribute, named *modelReference*, to specify 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.
- Two extension attributes, named *liftingSchemaMapping* and *loweringSchemaMapping*, that 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. Both schema mappings and 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 [29].

REST-based services described by hRESTS, on the other hand, can be semantically annotated by proposals like SA-REST [20] and MicroWSMO [21], WADL-described services

with, e.g., SBWS [30] and REST-based services defined by WSDL 2.0 can be directly annotated with SAWSDL. The MicroWSMO/SA-REST microformat as well as the SBWS approach adds SAWSDL-like annotations to hRESTS- respectively WADL-based service descriptions. In effect, this builds a parallel to the stack of WSDL and SAWSDL for REST-based services, as shown in Figure 1, and can thus be used to integrate REST-based services with WSDL-based ones. It is important to point out that, since both MicroWSMO/SA-REST and SAWSDL can apply WSMO-Lite service semantics, REST-based services can be integrated with WSDL-based ones [31]. Therefore, tasks such as discovery, composition and mediation can be performed completely independently from the underlying Web service technology.

REST's uniform interface makes it a Web-friendly architecture. More specifically, its "identification of resources" constraint, which specifies that every resource has to be addressable, makes REST a natural fit for the vision of a Semantic Web [32] and creates a network of Linked Data [33]; no parallel exists for SOAP's remote method invocation. It follows that REST-based Web services are an ideal carrier of semantic data and would even provide the additional benefit of resource resolvability in human-readable HTML [30].

For both REST- [34] and WSDL-based [35] Web services there exist tools which provide developers support for annotating Web services semantically.

IV. Application

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 [22] 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 [22]. In Internet-scale systems intermediaries for caching, filtering, monitoring or, e.g., logging are "must haves" to ensure good performance, scalability and maintainability.

SOAP-based systems usually don't support such intermediaries directly due to their Web-unfriendly architecture abusing HTTP as a pure transport protocol while it is in fact an application protocol. In SOAP, e.g., data is often retrieved by POSTing a SOAP-request to the service which then returns the desired data. This breaks intermediaries that serve as proxies or caches which typically perform their functions based on the standard semantics associated with the HTTP verbs and headers in the messages flowing through them. In contrast, Fielding [4] 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.

Another important aspect which eases tasks like monitoring and logging is that REST doesn't use implicit state transitions. Each request from client to server must contain all the information necessary for the server to understand the request; a client cannot take advantage of any stored context on the server. The server of course knows about the state of its resources but doesn't keep track of individual client sessions, so all session state is kept entirely on the client [4]. The state is represented by a set of hyperlinked resources, or, to say it with Fielding's words: "[REST uses] hypermedia as the engine of application state" [4]. This greatly improves reliability and scalability as well as the before mentioned visibility of services. Since all the needed state information is contained in every request, recovery from partial failures [22] is a lot easier which improves reliability. Scalability, on the other hand, is improved because not having to store state between requests allows the server to quickly free resources, and further simplifies implementation because the server doesn't have to manage resource usage across requests [4]. The disadvantage of this statelessness is that the network performance might be decreased since all the state information has to be transferred in every request and can't be left on the server.

In contrast, SOAP-based systems most of the time 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 identifiable which in turn implies the need for (complex) technologies like Web Services Business Process Execution Language (WS-BPEL) [36].

SOAP's before described opaqueness leads to some severe security problems in enterprise scenarios. Since SOAP runs on top of HTTP it "goes through firewalls like a knife through butter", according to Tim Bray (an editor of the XML specification) and it is difficult to inspect and filter the transported data. So, e.g., SOAP has several different ways of encoding and transporting binary data and there is nothing which specifies if a method just reads data or if it creates/modifies/deletes data. REST, if implemented correctly, on the other hand, clearly specifies the type of the method by the used HTTP verb. Such architecture makes it trivial for an administrator to declare parts of his network (which, as an additional advantage, can be clearly specified by URI patterns) as "read only" by filtering requests based on the used HTTP verb.

Message confidentiality and integrity are not to be forgotten when speaking about security. While specifications such as WS-Security [37] address exactly those issues for SOAP-based services, REST-based services typically fall back on HTTPS. The problem of HTTPS is that in large data centres, SSL is typically terminated at the edge of the network—at the firewall, load balancer or router. This opens the door for man-in-the-middle attacks [38], thus additional measures have to be taken at the data layer to provide true end-to-end security, just as SOAP-based services do.

On the data layer, SOAP-based services rely on the automatic mapping between the exchanged XML data and the object oriented constructs of the used programming languages. As already mentioned in section II this mapping is brittle [5] and results all too often in severe interoperability problems. In REST, however, the developer usually deals directly with the exchanged data which is often XML, but includes also other formats such as, e.g., JSON. Since REST heavily relies on standard Internet Media Types often special libraries for handling those common data formats exist. When no library exists, the developer has to deal directly with the (XML) data. Extensions for common languages such as Cω or LINQ (Language Integrated Query) for C# or E4X (ECMAScript for XML) for JavaScript ease the data handling enormously and avoid the inherent O/X impedance mismatch (some details about how Cω avoids the impedance mismatch can be found in [24]).

V. Service Discovery and Composition

Before a Web service can be used it has to be found. In traditional SOAP-based services this is usually done by a technology called Universal Description, Discovery and Integration (UDDI) [39]. A UDDI registry stores information about Web service providers, the Web services they make available and the technical interfaces which can be used to access those services as well as metadata about those services. Another, less known, technology used in the SOAP context is the Web Services Inspection Language (WS-Inspection) [40] developed by Microsoft and IBM. In contrast to UDDI, which follows a centralized approach and is thus most often used within companies, WS-Inspection relies on a completely decentralized, distributed model for providing service-related information. Each service provider places WS-Inspection documents with fixed names (*"inspection.wsil"*) on common entry points, e.g., the company's website root directory. Obviously, WS-Inspection is not used that much as a simple Google search[1] reveals just 26 hits.

For REST-based services no similar technologies exist. The usual practice to find a REST-based service is to go to a website like ProgrammableWeb[2] which collects and categorizes services. If REST services would be specified by technologies such as hRESTS it would be rather trivial to write (or augment) crawlers to index REST-based services. Additionally, if the service description would be augmented by semantic annotations such as SA-REST/MicroWSMO it would be even possible to use that semantic data for service discovery. The same mechanism could be used for SOAP-based services in conjunction with (semantically annotated) WS-Inspection.

Since both, REST- as well as SOAP-based services, can be described by the same semantic annotation (as described in section III), all previous work and research for semantic discovery could be reused independently of the used technology. Lumina [41], a semantic discovery tool that is built on top of UDDI, e.g., is one of these tools for SAWSDL.

After finding suitable services, they are often combined to so called mashups. A common and still open problem is data mediation/integration. Typically a developer implements a special mediation layer, which often represents the major part of the needed code, to translate the data formats between different services. Also here, semantic annotation enables further automation.

One approach is to specify, additionally to the used ontology concept, a lifting and lowering schema. Those schemas are then used to translate the service's native data format to the data structure of the ontology, the so called grounding schema. This clearly adds another abstraction layer and is a scalable solution because each service provider has to provide only one lifting/lowering schema pair in contrast to the usual way which demands a mapping to all other services that may want to use this service in the future. By using those abstractions, semantic mashups (termed as smashups) could be generated completely automatically and interesting new approaches, such as querying data from different services by SPARQL queries as described in [30] could become generally possible. But unfortunately it is not always possible to define a complete mapping to the grounding schema; as ontologies grow in richness and detail the frequency with which data mapping cannot be onto will surely go up; this needs further research.

VI. Conclusions and Future Work

In this paper we have outlined the main differences between SOAP- and REST-based Web services. We have shown that REST-based services can be more scalable, reliable and visible and are thus the preferred choice for Internet-scale applications. On the other hand, SOAP-based services often better fit for in-company deployments were legacy systems, which initially have not been built to be Web-friendly, have to be integrated with other services and systems.

However, the RESTful service landscape still suffers from shortcomings on formal describing, finding and orchestrating services as well as the non-existence of a holistic framework covering the entire service lifecycle.

The root of those issues is the lack of an agreed standard to describe a REST-based Web service. There have been numerous approaches, but none gained broad support so far. While WADL [18] seems to be the most mature one, hRESTS [17] is in our opinion the most interesting approach. It not only concentrates all service documentation (human- and machine-readable) in one accessible document following therefore strictly the Don't Repeat Yourself (DRY) principle [42], but is also easily discoverable by search engines and offers a unique entry point for the service's usage.

Nevertheless, most of the time, the syntactic description of a service's interface is not enough—also the semantics of the data and the behaviour of the service have to be documented. By using hRESTS in combination with extensions such as SA-REST [20] and MicroWSMO [21] the textual description addressing human users can be augmented with semantic annotations which in consequence allow machines to interpret the service documentation. In the next step all this information can be leveraged to build powerful discovery as well as composition methods.

[1] Search for WS-Inspection documents using Google's *inurl* operator: http://www.google.com/search?q=inurl:inspection.wsil&filter=0

[2] http://www.programmableweb.com/

Despite the inherent differences between the two architectural styles, both, the traditional SOAP-based as well as the lightweight REST-based Web service stack, can be integrated in a vibrant service ecosystem. To this end, in future work we plan to define and develop a holistic framework which builds on pre-existing results and extends current research. The framework's aim is to provide a simplified and coherent approach for describing, semantically annotating, finding, using and composing services with less effort.

We plan to follow an approach which combines both the knowledge and assistance of the crowd as well as the power of software engineering (computer processing). Towards the proposed framework research includes, but is not limited to, the following core research topics: 1) easy service description; 2) enhanced semantic annotation; 3) scalable and reliable application; and 4) expressiveness for service discovery and effortless composition of services.

REFERENCES

[1] S. Feldman, J. Duhl, J. Rahal Marobella and A. Crawford, *"The Hidden Costs of Information Work"*, Mar. 2005

[2] D. Huynh, S. Mazzocchi, D. Karger, "Piggy Bank: Experience the Semantic Web Inside Your Web Browser", LNCS 3729, pp. 413-430, Oct. 2005

[3] E. Tholomé, *"A well earned retirement for the SOAP Search API"*, Retrieved: Dec. 22, 2009. Available: http://googlecode.blogspot.com/2009/08/well-earned-retirement-for-soap-search.html

[4] R. T. Fielding, *"Architectural Styles and the Design of Network-based Software Architectures"*, Ph.D. dissertation, Dept. Inform. Comput. Sci., Univ. California, Irvine, USA, 2000.

[5] S. Loughran and E. Smith, *"Rethinking the Java SOAP Stack"*, IEEE Int. Conf. Web Services (ICWS) 2005, Orlando, Florida, USA, Jul. 2005.

[6] Web Services Description Language (WSDL) Version 2.0, 2007.

[7] H. Haas, *"Reconciling Web Services and REST Services"*, presented at the the 3rd IEEE European Conference on Web Services (IEEE ECOWS 2005), Växjö, Sweden, Nov. 2005. Available: http://www.w3.org/2005/Talks/1115-hh-k-ecows/

[8] J. Nielsen. (1999, March 21). *URL as UI* [Online] Available: http://www.useit.com/alertbox/990321.html

[9] N. Walsh. (2006, July 14). *"WITW: WSDL: 1, Norm: 0"* [Online] Available: http://norman.walsh.name/2005/02/24/wsdl

[10] HTML5, Editor's Draft 21 December 2009, Revision 1.3556 Available: http://www.w3.org/html/wg/html5/

[11] P. Prescod. (2002, August 1). *"Web Resource Description Language ('Word-dul')"* [Online] Available: http://www.prescod.net/rest/wrdl/wrdl.html

[12] N. Walsh. (2005, June 22). *"WITW: NSDL"* [Online] Available: http://norman.walsh.name/2005/03/12/nsdl

[13] T. Bray (2005, May 3). *"SMEX-D (Simple Message Exchange Descriptor)"* [Online] Available: http://www.tbray.org/ongoing/When/200x/2005/05/03/SMEX-D

[14] J. Cowan. (2005, May 9). *"Resedel"* [Online] Available: http://recycledknowledge.blogspot.com/2005/05/resedel.html

[15] R. Salz. (2003, October 14). *"Really Simple Web Service Descriptions"* [Online] Available: http://webservices.xml.com/pub/a/ws/2003/10/14/salz.html

[16] D. Orchard, *"Web Description Language (WDL)"*, Retrieved: Jan. 7, 2010 [Online]Available: http://www.pacificspirit.com/Authoring/WDL

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

[18] M. J. Hadley, "Web Application Description Language (WADL)", Nov. 2006. Available: https://wadl.dev.java.net/wadl20061109.pdf

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

[20] A. P. Sheth, K. Gomadam and J. Lathem, *"SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups"*, IEEE Internet Computing 11 (6), pp. 84-87, Nov./Dec. 2007

[21] J. Kopecký and T. Vitvar, *"D38v0.1 MicroWSMO: Semantic Description of RESTful Services"*, Feb. 2008. Available: http://wsmo.org/TR/d38/v0.1/20080219/d38v01_20080219.pdf

[22] J. Waldo, G. Wyant, A. Wollrath and S. Kendall, "A Note on Distributed Computing", Sun Microsystems Labs., Mountain View, CA, USA, Tech. Rep. SMLI TR-94-29, Nov. 1994.

[23] S. Vinoski, "Toward Integration—Demystifying RESTful Data Coupling", *IEEE Internet. Comput.*, vol. 12, no. 2, pp. 87-90, Mar. 2008.

[24] S. Vinoski, "RPC Under Fire", *IEEE Internet. Comput.*, vol. 9, no. 5, pp. 93-95, Sept. 2008.

[25] T. Vitvar, J. Kopecký, J. Viskova and D. Fensel, *"WSMO-Lite Annotations for Web Services"*, LNCS 5021, pp. 674–689, 2008

[26] OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004. Available: http://www.w3.org/Submission/OWL-S/

[27] Web Service Modeling Ontology (WSMO), WSMO Final Draft 21 October 2006. Available: http://www.wsmo.org/TR/d2/v1.3/

[28] Web Service Semantics - WSDL-S, W3C Member Submission 7 November 2005. Available: http://www.w3.org/Submission/WSDL-S/

[29] *Semantic Annotations for WSDL and XML Schema (SAWSDL)*, W3C Recommendation, 2007.

[30] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)", *J. Web Semantics*, vol. 6, no. 1, pp. 61-69, Feb. 2008.

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

[32] T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web", *Scientific Amer.*, vol. 284, no. 5, pp. 34-43, May 2001.

[33] C. Bizer, T. Heath and T. Berners-Lee, "Linked Data - The Story So Far", *Int. J. Semantic Web Inform. Syst. (IJSWIS)*, to be published.

[34] M. Maleshkova, C. Pedrinaci and J. Domingue, "Semantically Annotating RESTful Services with SWEET", presented at the 3rd Int. Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR²) at the 8th Int. Semantic Web Conf., Washington D.C., USA, Oct. 2009.

[35] A. Heß, E. Johnston and N. Kushmerick, *"ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services"*, LNCS 3298, pp. 320-334, 2004

[36] *Web Services Business Process Execution Language Version 2.0*, OASIS Standard, 2007.

[37] *WS-Security Core Specification 1.1*, OASIS Standard, 2006.

[38] G. Peterson. (2007, Feb. 28). *CAPEC-57: Utilizing REST's Trust in the System Resource to Register Man in the Middle* [Online] Available: http://capec.mitre.org/data/definitions/57.html

[39] *Universal Description, Discovery and Integration (UDDI) Version 3.0.1*, OASIS Standard, 2003.

[40] *Web Services Inspection Language (WS-Inspection)*, 2001..Available: http://www.ibm.com/developerworks/library/specification/ws-wsilspec/

[41] *Lumina - Semantic Web Service Discovery*, Dep. Comput. Sci., Univ. Georgia, USA.. Available: http://lsdis.cs.uga.edu/projects/meteor-s/downloads/Lumina

[42] A. Hunt and D. Thomas, "The Evils of Duplication", in *The Pragmatic Programmer: From Journeyman to Master*, Old Tappan, NJ: Addison-Wesley Prof., 1999, ch. 7, pp. 26-33.