RESTDoc: Describe, Discover and Compose RESTful Semantic Web Services using Annotated Documentations

Davis John and Dr. Rajasree M. S.

[1]Masters Student, Department of Computer Science and Engineering, College of Engineering Trivandrum
[2] Professor, Department of Computer Science and Engineering, College of Engineering Trivandrum
davisjohnhere@gmail.com, rajasree40@gmail.com

ABSTRACT

Development of a semantic web is gaining a lot of traction recently. At the same time, another change is also getting a lot popular on the web - a move from complex SOAP based web services to the simpler RESTful services that work over the existing HTTP infrastructure. Various techniques had been proposed to add semantics to RESTful services. But most of these solutions suffer from the fact that they are either extensions of solutions applicable for SOAP based services or they require external description files which leaves developers with one more artifact to develop and maintain. One of the objectives of this paper is to define a standard Microformats like syntax that helps to annotate semantics into the already existing documentation of these services doubling them as a machine-readable description. Further, the paper extends these basic annotations to link between related services and enable automatic discovery and composition. Syntax for discovering resources as users browse a website is also proposed and a proof-of-concept extension is developed for the Google Chrome browser.

Categories and Subject Descriptors
H.3.5 [Information Systems]: INFORMATION STORAGE AND RETRIEVAL – Online Information Services – [Web-based services]

General Terms
Design

Keywords
Semantic Web, intelligent agents, web service, semantic web service, REST, RESTful architecture, service discovery, service composition, Microformats, Poshformats, RDF.

1. INTRODUCTION

Semantic web is a collective movement towards adding semantics or meaning to the data available on the internet thus making them machine readable. Data with semantics will be key in the future web where human interaction can be reduced for exploring and using this information. There have been two main initiatives towards adding semantics to the web: Linked Data and semantic annotations.

Linked Data is an effort to create a web of data, parallel to the web of documents - the web we know and use today [1]. Since HTML, the language of the web was deemed insufficient to

accurately and expressively describe strongly typed relationships between data, a new XML based language named RDF (Resource Description Framework) was suggested [2]. RDF documents define relationships in the form of subject-predicate-object triplets and can thus model a wide variety of links. RDF documents are supported by schemas written in either RDFS [3] or OWL [4][5].

While Linked Data is very expressive when it comes to describing relationships, it has a downside as well: these descriptions are separate from the current web that humans use and leaves developers with another artifact to develop and maintain. This has lead into development of techniques that try to integrate the human web and the machine-readable web into one single entity. Two important standards developed in the area of semantic annotations are Microformats [6] and RDFa [7][8].

1.1 Semantic Web Services

The concept of semantic web has also been applied to web services, using semantic web techniques to (1) describe web services themselves and (2) add meaning to the results provided by web services to make them machine readable and to increase the utility of the information gathered.

Most of the research in semantic web services has been around SOAP based services. These are the "conventional" web services that have been the prime solution until a few years back. While SOAP packs a lot of power into accurately describing various aspects of a service, such verbosity is often undesired and leaves developers with a high entry-barrier that most chose not to take. The recent attention gained by RESTful services [9] - an entirely different service architecture - is a natural response to the prohibitive complexity of developing and describing SOAP based services. The REST philosophy prioritizes simplicity over verbosity and works over the existing HTTP infrastructure. Resources are represented as URLS and CRUD operations are defined by the POST, GET, PUT and DELTE HTTP methods respectively. Its resemblance to the way the web works has resulted in widespread adoption since the days of Web 2.0.

Some research has gone into utilizing semantic web techniques specifically for RESTful services. These solutions suffer from either of the two problems. (1) They are generalizations of existing mechanisms to describe SOAP services. REST being an entirely different architecture, such an approach results in a lot of unwanted markup and code. This conversion to another architecture steals some of the implicit properties of RESTful architecture like simplicity, focus on resources rather than services etc. (2) These solutions require an external description, which has to be created and maintained by the developer adding to the effort required.

Further, the solutions for describing RESTful services do not take into account the possibilities of automatic discovery and composition of services. Papers had been published on new proposals that work together with one of these markup languages to provide description. However, these solution usually suffer from the same issues as the markup languages they are mainly aimed at SOAP based services and do not fit well into the REST architecture.

2. RELATED WORK

Many solutions had been proposed for formally describing RESTful web services. These proposals approach the problem from different directions each providing a novel way of addressing the issue at hand. Most of these solutions were member submissions to the W3C but there is hardly any consensus on one global standard.

2.1 Web Service Description Language (WSDL) 2.0

WSDL 2.0 [10] is an extension of the Web Service Description Language (WSDL) that was used to describe traditional SOAP based services. WSDL 2.0 supports describing RESTful services by mapping the HTTP methods into explicit services available at the given URLs. So every resource will translate into 4 or fewer different services: GET, POST, PUT and DELETE.

The advantage of WSDL 2.0 is that it provides a unified syntax for describing both SOAP based and RESTful services. It also has very expressive descriptions where you can define the specific data type, the cardinality and other advanced parameters for each input type. Normal or Body Text.

However, WSDL 2.0 requires RESTful services to be viewed and described from a different architectural platform: that of traditional RPC-like services. This forceful conversion negates many of the advantages of the RESTful philosophy. In addition, the expressiveness of the format comes at the price of losing the simplicity achieved by moving to the RESTful paradigm. These verbose files are not the easiest to be written by hand and also impose a maintenance headache. Hence, WSDL files are typically generated with the help of some tools. Further, WSDL descriptions are external files based on XML syntax that the developer has to create and maintain separately.

2.2 Web Application Description Language (WADL)

Web Application Description Language (WADL) [11] is another XML based description language proposed by Sun Microsystems. Unlike WSDL, WADL is created specifically to address the description of web applications, which are usually RESTful services. WADL documents describe resources instead of services but maintain the expressive nature of WSDL.

WADL still has some of the concerns associated with WSDL in that they still requires an external XML file to be created and maintained by the developer. It also results in boilerplate code.

2.3 hRESTS

hRESTS [12] is a Microformat that attempts to fortify the already existing service documentations with semantic annotations in an effort to reuse them as formal resource descriptions. The Microformat uses special purpose class names in the HTML code to annotate different aspects of the associated services. It defines the annotations service, operation, address, method, input and output. A parser can examine the DOM tree of an hRESTS fortified web page to extract the information and produce a formal description.

hRESTS is a format specifically designed for RESTful services and hence avoids a lot of unnecessary complexities associated with other solutions. It also reduces the efforts required from the developer since he no longer needs to maintain a separate description of the service.

One downside with hRESTS is that, despite being specifically designed for REST, it still adheres to an RPC-like service semantics. It is still required to explicitly mention the HTTP methods as the operations involved. Moreover, instead of representing the attributes of a resource, it attempts to represent them as input-output as in traditional services. This results in a lot of unnecessary markup.

2.4 SA-REST

Similar to hRESTS, SA-REST [13] is also a semantic annotation based technique. Unlike hRESTS, it uses the RDFa syntax to describe services, operations and associated properties. It shares much the same advantages and disadvantages as the hRESTS format. In addition, since SA-REST is strictly based on RDF concepts and notations, the developer needs to be well aware of the full spectrum of concepts in RDF.

2.5 SEREDASj

SEREDASj [14], a novel description method addresses the problem of resource description from a different perspective. While the other methods resort to the RPC-like semantics of input-operation-output, SEREDASj attempts to describe resources in their native format: as resources with attributes that could be created, retrieved, updated and deleted. This helps to reduce the inherent difference between operation oriented and resource oriented systems. The method also emphasizes on a simple approach that provides a low entry barrier for developers.

One interesting aspect about SEREDASj is that it uses JSON (JavaScript Object Notation) [15] to describe resources. JSON is an easy and popular markup technique used on the web - especially with RESTful service developers. The advantage is that, the target audience is already familiar with the notation used for markup and can reduce friction when it comes to adoption.

SEREDASj, however, addresses the documentation and description in the reverse order. You can create the description in JSON first and then generate the documentation from this. This can increase upgrade effort required for existing services and is not very flexible. It is still possible to embed these JSON descriptions into existing documentation but it floats as a data island, separated from the rest of the HTML page. This, again, causes some duplication of data between the documentation and the description and causes a maintenance hurdle.

2.6 Evaluation of Existing Solutions

The different solutions discussed so far creates a lot of variety in how RESTful services could be formally described. These differences range from how the thinking process works to minute differences in the representation. We will compare these solutions based on some of these aspects here.

Specificity
While most of the solutions discussed here are specific to RESTful services, WSDL 2.0 provides a generic solution that is applicable to both SOAP and RESTful services. This generality has the advantage of providing a unified method for describing web services irrespective of the service architecture. This could be a good thing for developers coming from the SOAP based domain, who wants to provide an alternative way to access their services. However, when it comes to those who prefer the RESTful philosophy for its inherent simplicity and the thought process behind it, might find that it cancels out the advantages achieved by the paradigm shift. This has prevented WSDL 2.0 from being accepted among most RESTful service developers.

2.6.1 External or Embedded

Description techniques like WSDL 2.0 and WADL use external files to provide a formal description of services while others attempt to reuse and fortify the already existing service documentations with machine readable annotations. An external description can have some minor advantages when there is a clear distinction between developers and designers in the team. However, with web design growing into a multi-disciplinary craft, most small teams consist of people who work in both the design and development worlds. This has resulted in such a clear distinction being even less desirable. Moreover, having the descriptions inscribed into the documentation saves a lot of time and effort when it comes to creating and maintaining the services.

2.6.2 Markup Language

WSDL 2.0 and WADL uses XML to present service descriptions. XML with its advanced features and accurate type definitions can be used to precisely describe these services. However, this precision comes at the price of sacrificing simplicity and maintainability. SA-REST uses RDFa to annotate service descriptions in documentations. RDFa uses some additional attributes added to XHTML to represent relationships. The hRESTS Microformat, on the other hand, uses simpler Microformat based techniques for annotations. SEREDASj uses an entirely different approach by using JSON as the notation used for markup.

2.6.3 Description Semantics

Most of the solutions discussed here uses the traditional RPC-like semantics of input-operation-output. This is true irrespective of whether the solution is specific to RESTful services or not. WADL and SEREDASj are worth mention in this aspect that instead of going with an RPC-oriented approach, services are described as resources with attributes. This approach reduces the amount of markup required to describe RESTful services.

2.7 Motivational Factors

In this section, we have evaluated different solutions to the problem of describing RESTful services and compared them focusing on different aspects like the specificity, external vs. embedded, markup language used and the description semantics. Certain solutions excel in specific aspects but lack in others. There is room for further research and enhancement in this area to produce an integrated description technique that extends existing documentations into machine readable descriptions and describes services as resources with attributes instead of using an RPC-like input-operation-output concept. Further, if we could work out a system that allows services to be interlinked, we could use these links to discover new services and possibly compose them together to create a complex services from the pieces. This is in alignment with the direction Semantic Web is progressing and would be a step forward towards a more intelligent web.

3. PROPOSED SOLUTION

In this paper, we propose a solution – RESTDoc – that uses a combination of Microformats-style markup and RDFS to provide a comprehensive framework for describing, discovering and composing RESTful services by adding semantics. Microformats, being simple and reusing a lot of the properties of HTML, provides users with a low entry-barrier for developers, which can increase adoption rate. These annotations are not visible to user but hidden in the HTML source and hence do not come in the way of users browsing the site.

In order to enable strong interlinking between services, a more robust solution like RDF and a backing RDF Schema is needed. This is achieved by providing an adapter for automatic conversion from the Microformat to RDF and providing a ready-made RDF Schema for the purpose. This way, the developers need not be concerned with the RDF descriptions that work in the background.

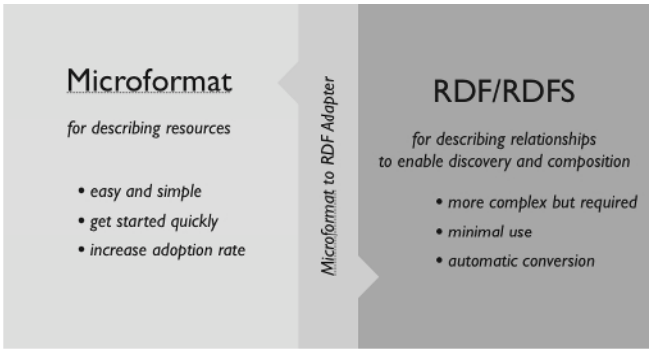

Figure 1. Overview of the proposed system

The solution currently addresses RESTful services that represent resources using JavaScript Object Notation (JSON). It is possible to extend the idea to XML based services as well by making some assumptions about the XML serialization.

3.1 Description

Services are described using special purpose annotations in the HTML code. These annotations are specified in the class attribute of the associated tags. This attribute is usually used for classification of HTML elements and also as selectors for JavaScript access and to style elements via CSS. Using special purpose annotations as class names help us to reuse whatever is already provided by HTML and JavaScript. Following annotations are proposed for describing resources in a RESTful API.

1. hresource: This is the root annotation that marks the resource description. All other annotations are contained within an element marked with class="hresource". A client parsing a page could treat the presence of this annotation as an indication of the existence of a resource description on the page. Unless all other annotations are encapsulated in an hresource, they will not be parsed.
2. name: Annotates the name of the resource. This can be any human readable name and need not have any programming significance.
3. url: Annotates the URL at which the resource is accessible.
4. attribute: Annotates an attribute/property of the resource. All attributes of a resource should be annotated with this annotation. Specific characteristics of the attribute could be further specified by more annotations that are used together with the attribute annotation.
5. comment: Provides a human-readable description of the attribute.
6. required: Indicates a required attribute. This annotation is always used along with the attribute annotation.
7. queryable: Indicates an attribute that may be provided in the HTTP querystring during a GET operation to filter the results. This annotation is always used along with the attribute annotation.
8. read-only: Indicates a read-only attribute. A read-only attribute may be retrieved during a GET operation but may not be included in a POST or a PUT. This annotation is always used along with the attribute annotation.
9. write-once: Indicates a write-once attribute that can be specified only during the create operation (POST) but not during update (PUT). This annotation is always used along with the attribute annotation.
10. guid: Indicates if an attribute is a globally unique identifier for the resource that could be used across multiple services.

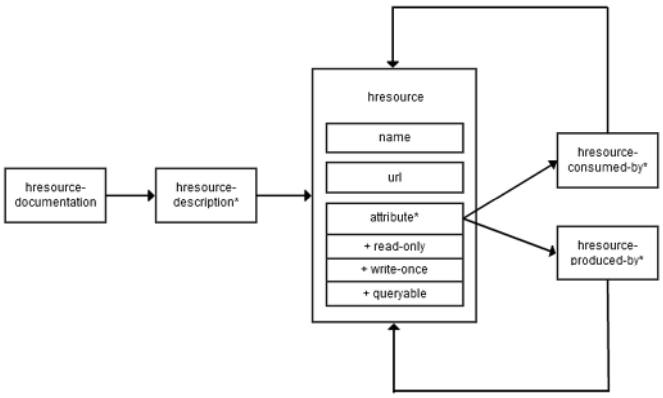

Figure 2. Relationship between the semantic annotations ('*' indicates a one-to-many relationship)

The use of these annotations is explained in the example provided in Fig. 3.

An intelligent agent extracts information about the service by traversing the DOM tree of the HTML page. APIs for traversing trees are available in almost all programming languages including JavaScript which is used extensively on the web and also to make extensions for many popular browsers such as Chrome and Safari. Browsers or browser extensions could then could parse this data and automate the creation of clients for these services.

3.2 Discovery

The discovery mechanism works to enable discovery of new, similar and related services. Our proposal for service discovery addresses two different aspects of the problem: discovery by users and discovery by services.

```
<div class="hresource">
   <h1 class="name">User</h1>

   URL:
   <span class="url">
      http://example.com/api/user
   </span>

   <ol>
      <li>
         <code class="attribute required
               write-once queryable">
            username
         </code> -
         <span class="comment">
            required, must be unique.
         </span>
      </li>

      <li>
         <code class="attribute required">
            password
         </code> -
         <span class="comment">
            required.
         </span>
      </li>

      <li>
         <code class="attribute required
               queryable">
            email
         </code> -
         <span class="comment">
            email address of the user.
         </span>
      </li>

      <li>
         <code class="attribute queryable">
            country
         </code>
      </li>

      <li>
         <code class="attribute read-only">
            ip_address
         </code>
      </li>
   </ol>
</div>
```

Figure 3. An example User resource

3.2.1 Discover-as-you-browse:

This deals with enabling browsers (or browser extensions) to hint users about the presence of resources as they browse a website. This works similar to how users discover RSS feeds. In this paper, we propose the use of the link element provided by HTML to alert browsers about the presence of REST resources on the website. This can be supplemented with automatic client creation since the browser can now link to the resource descriptions and read the data from the page using DOM traversal. The syntax of the link tag to use is as follows:

```
<link rel="hresource-documentation" href="http://example.com/api/" />
```

The tag goes into the header of a page and the value hresource-documentation in the rel attribute specifies that the linked item is a REST resource. From the documentation page, links to individual resources in the API are annotated with hresource-description as the value of the rel attribute:

```
<a rel="hresource-description" href="http://example.com/api/user/"> User

</a>
```

The annotation is added to each resource linked from the API documentation start page. The client thus traverses from the homepage to the documentation page and from there to the individual resource descriptions to discover the resources and present this to the user. Fig. 4 shows a screenshot of the RSS notification that is already present in popular browsers and the proposed implementation of the resource discovery feature that works in a similar way. The toolbar button shows a badge with the number of resources found when visiting a site with embedded resource descriptions.

Figure 4. The RSS discovery mechanism and the proposed service discovery mechanism.

3.2.2 Automated Discovery:

Automated discovery deals with the ability of a service to discover similar services in the same domain and to link to them. Service discovery allows clients to find services that could provide data required to access another service or could consume data received from another service. Existing service discovery mechanisms use a directory-oriented approach and are suited only for SOAP based services. We will extend our description syntax to provide a discovery mechanism for RESTful services that works peer-to-peer without any dependence on a central controller.

The system works by identifying the different links as it comes across new resources and building up a graph connecting them. At a later stage, this graph could be traversed to discover new possibilities and to look for other sources of input.

The following annotations for inter-links between services are defined to enable discovery.

3.2.2.1 Link to Superclass

When a resource is a subclass of another resource, this link is indicated by the rel attribute hresource-is-a. This implies that wherever the superclass is accepted, the subclass is also accepted. For e.g. if a publisher defines a Book resource to provide a search of their catalog, they could annotate the resource to be a subclass of a more generic Book resource.

```
<a rel="hresource-is-a"
   href="http://dublincore.org/book/">Book</a>
```

If there is another service from a bookshop that is known to accept a generic book resource for a purchase process, the client could infer that the specific book resource from the catalog would also be accepted there and use it. For this linking to work properly, we need a core set of resources that can be extended by others. Fortunately, there is already a project named Dublin Core running that has defined many commonly used resources. We could reuse these resources for our purpose and use them as the root resources.

3.2.2.2 Link to Consumers

When an attribute of a service is consumed by another known service, this is annotated using a rel attribute hresource-consumed-by. This enables a software agent to find out what all can be done with the resource that it has already retrieved.

```
<code class="attribute">
   ISBN
</code>
Consumers:
<ul>
   <li rel="hresource-consumed-by">
      http://bookshop.com/book-order\#isbn
   </li>
   <li rel="hresource-consumed-by">
      http://library.com/rented-book\#isbn
   </li>
</ul>
```

3.2.2.3 Link to Producers

Similar to the link to consumers, services can annotate a link to a producer of one of its attributes. This helps reverse traversal of resources and also makes the system more peer-to-peer. This way, a link needs to be provided in either at one of the consumers or at the provider and an agent can identify this with link traversal. The annotation is made with the rel attribute hresource-produced-by.

3.3 Composition

Service composition is made possible by using the same annotations that were made for discovery. A graph is constructed starting from a resource and then traversing the parent, consumer and producer links recursively. At each page, the descriptions are extracted and converted to RDF to update the graph. This way, a software agent that does not have the identifier

or a search parameter to access a specific resource could traverse the graph to figure out what other information could be used to lookup the identifier and present the choice to the user.

For e.g. the service provided by a bookshop might need the ISBN to order a book. However, the traversal of the graph could reveal a catalog service that retrieves book resources using titles or author names. This allows the software agent to provide a choice to the user where he can enter either the ISBN or the title. This works in the reverse direction also. Having received access to a resource, the software agent can suggest what all operations can accept the resource. So effectively a service that has a book resource can provide options for services from shops that let the user order the book or libraries that let the user lend the book.

4. CASE STUDY

To demonstrate the working of the framework and to analyze its effectiveness, a set of related RESTful services were developed and annotated using the framework. The following three services were created:

A catalog service from a book publisher that lets users search for their books.
A library's service that supports lending books online.
A bookshop's service for ordering books online.

Details that are not relevant to our framework like payment processing were not considered in the implementation. The service documentations were annotated using the proposed syntax and the interlinking between the services was also defined. The ISBN number, which is as a globally unique identifier, is used for linking between the services. Basic HTTP authentication is used for user access control with the services.

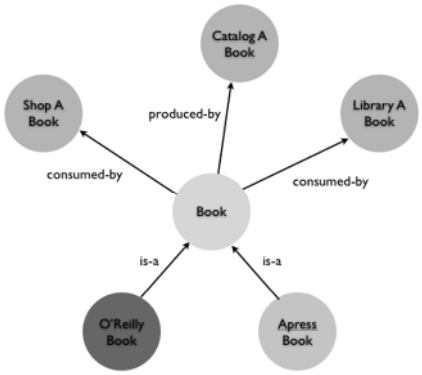

Figure 5. Graph constructed from the links in resource descriptions

The proposed browser extension was developed for the Google Chrome browser. On visiting a properly marked up website with a RESTful API, the extension indicates the presence of the service and provides details of the exposed resources to the user. The extension also presents an option to automatically generate a client for the service in PHP. It is possible to extend this further to support client generation in other languages as well.

A proof of concept application was developed using PHP for parsing the service documentations and to provide an interface for service discovery. The application was tested using the sample

services created. The user first invokes the catalog service with his inputs and presents the output to the user. On receiving the output, the application parses the annotations, identifies services that consume the obtained resource and presents further options available with the resource to the user. The application then displays options to lend the book and purchase the book using the related services from which the user could select one and proceed to complete the action.

5. CONCLUSION

Adding semantics to web service descriptions is an important step towards a better web that is accessible to both humans and machines alike. With the web taking a definite turn towards RESTful web services from the traditional RPC-like SOAP based services, it is important to devise a semantic description method for such services. However, for these solutions to be really adopted by developers, they should adhere to the RESTful philosophy and provide a low entry barrier. RESTDoc aligns with the REST philosophy and architecture and reuses existing service documentations to double them as machine-readable descriptions. The Microformats-like syntax used by the solution is simple and easy to write and maintain from a developer perspective. Further, the language describes services as resources with attributes instead of using an RPC-like input-operation-output concept used by most of the current solutions. This markup syntax is then extended to add interlinking between RESTful services, enabling automatic discovery and composition of services. The solution should further reduce the entry barrier for developers and thus increase adoption, resulting in a widely accepted standard. The existence of such a standard technique can make the wealth of resources available on the Internet accessible to machines, enabling the creation of intelligent software agents that can get a lot of work done with no to minimal intervention from humans.

6. REFERENCES

[1] Christian Bizer, Tom Heath, Tim Berners-Lee, “Linked Data - The Story So Far”, in *International Journal on Semantic Web and Information Systems IJSWIS*, 2009.

[2] Frank Manola, Eric Miller, Brian McBride, “RDF Primer”, *W3C Recommendation,* 10 February 2004.

[3] Deborah L. McGuinness, Frank van Harmelen, “OWL Web Ontology Language: Overview*”*, *W3C Recommendation,* 10 February 2004.

[4] W3C OWL Working Group, “OWL 2 Web Ontology Language: Document Overview”, *W3C Recommendation,* 27 October 2009.

[5] Dan Brickley, R.V. Guha, Brian McBride, “RDF Vocabulary Description Language 1.0: RDF Schema”, *W3C Recommendation,* 10 February 2004.

[6] Microformats, *http://www.microformats.org/*.

[7] Ben Adida, Mark Birbeck, Shane McCarron, Steven Pemberton, “RDFa in XHTML: Syntax and Processing: A collection of attributes and processing rules for extending XHTML to support RDF”, *W3C Recommendation*, 14 October 2008.

[8] Ben Adida, Mark Birbeck, “RDFa Primer - Bridging the Human and Data Webs”, *W3C Recommendation*, 14 October 2008.

[9] R.T. Fielding, “Architectural styles and the design of network-based software architectures”, PhD dissertation, Department of Information and Computer Science, University of California, Irvine, 2000.

[10] Roberto Chinnici,Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", *W3C Recommendation*, 26 June 2007.

[11] Marc Hadley, "Web Application Description Language", *W3C Member Submission*, 31 August 2009.

[12] Jacek Kopecky, Karthik Gomadam, Tomas Vitvar, "hRESTS: an HTML Microformat for Describing RESTful Web Services," *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2008.

[13] Karthik Gomadam, Ajith Ranabahu, Amit Sheth, "SA-REST: Semantic Annotation of Web Resources", *W3C Member Submission*, 05 April 2010.

[14] Markus Lanthaler, Christian Gütl, "A Semantic Description Language for RESTful Data Services to Combat Semaphobia", *5th IEEE International Conference on Digital Ecosystems and Technologies (IEEE DEST 2011)*, 31 May -3 June 2011, Daejeon, Korea.

[15] Crockford, "RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON)", Network Working Group, July 2006.