A Semantic Description Language for RESTful Data Services to Combat Semaphobia

Markus Lanthaler [1, 2]
[1] Institute for Information Systems and Computer Media
Graz University of Technology, Graz, Austria
[2] Digital Ecosystems and Business Intelligence Institute
Curtin University of Technology, Perth, Australia

Christian Gütl [1,3]
[3] School of Information Systems
Curtin University of Technology
Perth, Australia

Abstract—**RESTful Web services are an increasingly popular way for companies to expose their data on the Web. On the other hand, the Linked Open Data initiative is gaining traction recently. Since REST's principles align well with those of Linked Data, there is an increasing interest in the relationship of the two. Nevertheless, in practice they still largely remain separated, creating islands of data instead of a global graph of data forming the envisioned Semantic Web. There are different reasons for this. One is the reluctance of most Web developers to use Semantic Web technologies, a phenomenon we denote as *Semaphobia*. Another reason is that the Semantic Web is considered to be a disruptive technology which does not consider existing infrastructure. This makes it difficult for enterprises to update their legacy systems. To solve these and other issues, we propose a novel approach to semantically describe RESTful Data Services which in consequence leads to a mechanism to transform the data provided by such services to semantic resources. This aims to contribute to the availability of more semantic datasets. By keeping the approach as familiar and simple as possible for Web developers, we hope to lower the entry barrier and to foster the adoption of our approach.**

Index Terms—***Semantic Web, Web services, REST, Web 3.0, Internet, SOA, SEREDASj, Semaphobia***

I. INTRODUCTION

Web services, especially RESTful ones, are increasingly popular. According to ProgrammableWeb's statistics [1], 2010 has seen a twofold increase in new APIs per month compared to the year before. At the same time, semantic annotations start to gain acceptance across the community. Facebook's OpenGraph protocol, e.g., was implemented in over 50,000 Web sites within the first week of its launch [2]. Since REST's principles align well with the Linked Data principles, there is an increasing interest in the relationship of RESTful Web services and the Semantic Web respectively Linked Data. Nevertheless, the two still remain largely separated in practice. Instead of providing Linked Data via RESTful Web services, current efforts deploy centralistic SPARQL endpoints or Tabulator-like interfaces, or simply upload static dumps of the data in order to provide access to the data. This rarely reflects the nature of the provided data, i.e., descriptions about interlinked resources.

Another problem of current Semantic Web approaches is that they usually just provide read-only interfaces to the underlying data. This clearly inhibits networking effects and engagement of the crowd. RESTful data services could prove to be a viable solution to these problems. But there are still some shortcomings to be resolved in order to enable their automatic invocation and composition.

The major problem of RESTful services is that no agreed machine-readable description format exists to document them. All the required information of how to invoke them and how to interpret the various resource representations is communicated out-of-band by human-readable documentations. This usually does not cause any problems in the "human Web" since humans inherently understand the representations and are thus able to quickly adapt to new control flows (e.g. a change in the order sequence or a new login page to access the service). Machines on the other hand have huge problems to understand such representations; just as disabled users sometimes have. A blind user for instance cannot make any use of information contained in graphics. Machines suffer even more from such usability and accessibility problems.

Currently machine-to-machine communication is often based on static knowledge and tight coupling to resolve those issues. The challenge is thus to bring some of the human Web's adaptivity to the Web of 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.

To address these issues we introduce a new approach to semantically describe RESTful data services. We have placed strong emphasis on simplicity and on not requiring any changes on the Web service itself. This lowers the entry barrier for future Web developers and provides a viable upgrade path for existing infrastructure. The approach is extensible and flexible enough to be applicable in a wide application domain.

The reminder of the paper is organized as follows. First, in section II, we briefly discuss the lack of acceptance of the Semantic Web so far. Then, in section III, we give an overview of related work and section IV compares the use of XML and JSON for the use in typical data service scenarios. To foster the adoption of our approach, in section V different use cases are

ISBN: 978-1-4577-0872-5 (c) 2011 IEEE

described from which, then in section VI, the requirements for a semantic description language for RESTful data services are distilled. Our approach, SEREDASj, is presented in section VII and the algorithm to translate the service's data to RDF is presented in section VIII. Finally, section IX concludes the paper and gives an overview of future work.

II. SEMAPHOBIA!?

While the vision of a Semantic Web has been around for more than fifteen years it still has a long way to go before mainstream adoption will be achieved. One of the reasons for that is, in our opinion, the fear of average Web developers to use Semantic Web technologies. The reasons for this are manifold and should be further researched. Some developers are overwhelmed by the (perceived) complexity or think they have to be AI experts to make use of the Semantic Web and simply shut down when they hear the word *Ontology*. Others are still waiting for a killer application making it a classical chicken-and-egg problem. A common perception is that the Semantic Web is a disruptive technology which makes it a show-stopper for enterprises needing to evolve their systems and build upon existing infrastructure investments. Obviously some developers are also just reluctant to use new technologies.

Nevertheless, we think most Web developers fear to use Semantic Web technologies for some reason or another; we call this *Semaphobia*. To help developers get past this fear, and to show them that they have nothing to fear but fear itself, clear incentives along with simple specifications and guidelines are necessary. Wherever possible, upgrade paths for existing systems should be provided to build upon existing investments.

III. RELATED WORK

In contrast to traditional SOAP-based services, which have agreed standards in the form of WSDL and SAWSDL to be described, both, syntactically and semantically, no standards exist for RESTful services. In consequence, REST-based services are almost exclusively described by human-readable documentation describing the URLs and the data expected as input and output. SA-REST [3], hRESTS [4], and WADL [5] are probably the best-known proposals for the description of RESTful services.

SA-REST and hRESTS enrich the mostly already existing human readable HTML documentation with annotations to make it machine-processable. While hRESTS uses microformats to do so, SA-REST uses RDFa. The biggest difference between them is that SA-REST has some built in support for semantic annotations whereas hRESTS provides nothing more than a label for the inputs and outputs. SA-REST uses the concept of lifting and lowering schema mappings to translate the data structures in the inputs and outputs to the data structure of an ontology, the grounding schema, to facilitate data integration. MicroWSMO [6] is an extension for hRESTS adding similar semantic annotations. The Web Application Description Language (WADL), on the other hand, is closely related to WSDL. WADL describes a service by creating a monolithic XML file containing all the information about the service interface. This syntactic description of the service can then be semantically annotated by, for instance, SBWS (Semantic Bridge for Web Services) [7].

In principle, a RESTful service could even be described by using WSDL 2.0 [8] with SAWSDL [9] and an ontology like OWL-S [10] or WSMO-Lite [11]. A complete description of these ontologies is beyond the scope of this paper and thus we would like to refer you to [12].

The problem with all of the above described approaches is that they heavily rely on RPC's (Remote Procedure Call) flawed [13] operation-based model ignoring the fundamental architectural properties of REST. Instead of describing the resource representations, and thus allowing a client to understand them, they adhere to the RPC-like model of describing the inputs and outputs as well as the supported operations which results in tight coupling. The obvious consequence is that these approaches do not align well with clear RESTful service design.

One of the approaches that avoid the RPC-orientation and thus more suitable for RESTful services is EXPRESS [14]. Instead of describing the service, it describes the service's domain ontology in OWL [15]. An EXPRESS deployment engine will then use the domain ontology to create the RESTful service interface. Obviously such disruptive approach cannot be used to upgrade existing services, which makes it impossible to build upon existing infrastructure investments.

ReLL [16], the Resource Linking Language, does exactly the opposite. ReLL is a language to describe RESTful services with the aim to transform their exposed data to RDF and thus allowing harvesting already existing Web resources. Currently ReLL does not support any modification of the described resources, i.e., at the moment it supports only HTTP GET operations. This clearly restricts the possible use cases of ReLL at this point in time.

To circumvent the problem of describing their services, some service providers are building them on top of well-established standards like Atom [17] and OpenSearch [18], but unfortunately this is not always feasible or desirable. It is also just a solution for the service interface description; the problem of describing the exchanged data, i.e., the feed entries, still remains open. Sometimes this approach also yields to strange results, e.g. when a service provider just serializes an Atom feed into a JSON representation. Google's Data Protocol [19] is one of those inglorious examples. At least its subsidiary YouTube recognized the problem and is now offering an alternative JSON serialization [20].

IV. XML VS. JSON

Traditional Web services use XML for the data interchange with XML Schemas (XSD) as description format, but the inherent impedance mismatch between XML and object oriented programming constructs (O/X impedance mismatch) often results in severe interoperability problems. The fundamental problem is that 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. 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. Recent extensions for common languages such as Cω or LINQ (Language

ISBN: 978-1-4577-0872-5 (c) 2011 IEEE

Integrated Query) for C# or E4X (ECMAScript for XML) for JavaScript ease the data handling enormously and avoid the inherent O/X impedance mismatch.

Nevertheless, in most use cases addressed by Web services, all a developer wants to do is to interchange data—and here we are distinguishing between *data interchange* and *document interchange*. JSON was specifically designed for this: it is a lightweight, language-independent data-interchange format which is easy to parse and easy to generate. Furthermore, it is much easier for developers to understand and use. JSON's whole specification [21] consists of 10 pages (with the actual content being a mere 4 pages) compared to XML where the XML Core Working group alone lists *XML*, *XML Namespaces*, *XML Inclusions*, *XML Information Set*, *xml:id*, *XML Base*, and *Associating Stylesheets with XML* as standards [22]; not even including *XML Schema Part 1* and *XML Schema Part 2*.

Summarized, JSON's simplicity, ease of integration, and its raising adoption across the Web community [1] make it a first choice to bring the RESTful service ecosystem forward. Joe Gregorio [23] put it in a nutshell by saying that "there are still plenty of use cases for 'documents' of XML, but APIs on the web is not one of them". Given that we address data services, we thus base our approach on JSON instead of XML but would highlight that its principles are applicable to any serialization format.

V. USE CASES

One of the critiques of the Semantic Web is that it is a solution looking for a problem and that there are no clear use cases for it. The W3C answered this by publishing a list of case studies and use cases [24], nonetheless, the impression remains. Aware of these critiques, we would like to describe three potential use cases for SEREDASj.

The first use case SEREDASj addresses is the documentation of a service respectively its used data formats. Currently this is mostly done manually by creating a human-readable HTML documentation which is not machine-processable and thus, e.g., not directly usable within integrated development environments (IDEs). SEREDASj will not only create a machine-processable documentation but it could also automatically translate it into a nicely formatted human-readable format. This works similar to widely accepted documentation generators such as *Doxygen* [25] which automatically generate documentations for commented Java, C++, or Python code. A strong incentive for using a semantic approach as SEREDASj does, and a motivation to reuse existing ontologies, is that documentation fragments can be shared between different systems. All the developer has to do is to annotate a field with a concept; the human-readable documentation can then be generated by using the `rdfs:label` and `rdfs:comment` properties in the ontology. In case there is no domain ontology available yet, the developer could be supported to create one by techniques similar to the ones used by tools that map data from relational databases to RDF [26].

The second big use case for the proposed semantic description language for RESTful data services, especially for JSON-based ones, is the ability to create more flexible and dynamic service consumers or clients. JSON, for instance, has no built-in support for hyperlinks which makes it impossible to build services following the hypertext as the engine of application state (HATEOAS) constraint [27] without additional out-of-band information. In consequence, this leads to tighter coupled services. By using semantic annotations, the client will not only be able to figure out which elements in a JSON representation represent URIs, but also what these URIs and all the other elements mean. Using the uniform interface REST provides, a client might not only retrieve representations but manipulate them or create new ones as well.

Last but not least, data integration is another important use case. By having semantically annotated data it is possible to integrate it (semi-)automatically with other data sources. For instance, a typical mashup combining and showing data from different sources on a map could be created automatically. The widget would be able to automatically figure out which fields in the representation represent the needed coordinates and in consequence render the data on the map. This would render the creation of dashboards, an important business use case, much simpler and eliminate a lot of the usually needed data mediation code. Furthermore, the semantic descriptions allow the translation of the data representations to RDF and thus to integrate them into the linked data cloud.

VI. REQUIREMENTS

Analyzing the related work and the use cases, we derived a set of requirements for a semantic description language for RESTful data services.

Given that the description language is targeting RESTful services, it clearly has to adhere to REST's architectural constraints [27] which can be summarized as follows: 1) stateless interaction, 2) uniform interface, 3) identification of resources, 4) manipulation of resources through representations, 5) self-descriptive messages, and 6) hypermedia as the engine of application state. Stateless interaction means that all the session state is kept entirely on the client and that each request from the client to the server has to contain all the necessary information for the server to understand the request; this makes interactions with the server independent of each other and decouples the client from the server. All the interactions in a RESTful system are performed via a uniform interface which decouples the implementations from the services they provide. To obtain such a uniform interface every resource is accessible through a representation and has to have an identifier (whether the representation is in the same format as the raw source, or is derived from the source, remains hidden behind the interface). All resource representations should be self-descriptive, i.e., they are somehow labeled with their type. Finally, the hypermedia as the engine of application state (HATEOAS) constraint refers to the use of hyperlinks in resource representations as a way of navigating the state machine of an application.

While all of these constraints are important when designing a RESTful service, the most important aspects for a semantic description language are how the resources can be accessed, how they are represented and how they are interlinked. The description language should be expressive enough to describe how resource representation can be retrieved and manipulated, and what the meaning of those representations is. To integrate the service into the Semantic Web, the description language

ISBN: 978-1-4577-0872-5 (c) 2011 IEEE

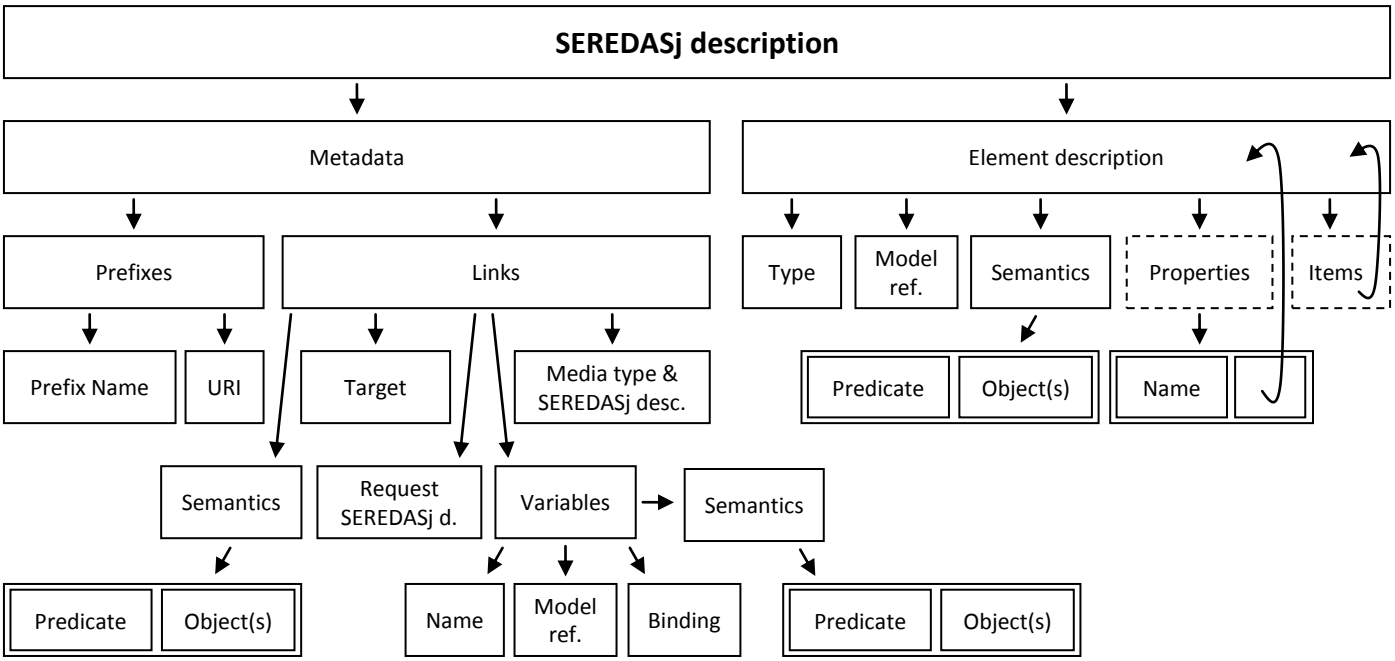

Figure 1. The SEREDASj description model

should also provide means to transform the representations in RDF triples.

To be able to evolve systems and build upon existing infrastructure, an important requirement is that no (or just minimal) changes on the existing system are required; this implies a requirement to support partial descriptions that can be completed later. Finally, in order to lower the entry barrier for developers and to foster its adoption, the approach has to be as simple as possible, specify clear use cases, and provide instant incentives.

VII. SEREDASJ

Considering the requirements described in the previous section we designed *SEREDASj*. SEREDAS stands for SEmantic REstful DAta Services, while the "j" should highlight that the approach is based on JSON (this leaves the door open for other data formats).

SEREDASj specifies, similar to Zyp's JSON Schema draft [28], the syntactic structure of a specific JSON representation. Additionally, it allows to reference JSON elements to concepts in an ontology and to further describe the element itself by semantic annotations. In contrast to the JSON Schema draft, SEREDASj has no validation rules as such but allows a developer instead to add arbitrary descriptions in the form of semantic annotations to a JSON element. The rationale behind this is that we believe that the data has to be understood (semantically) to be validated and used correctly; simple validation rules as the ones proposed by Zyp are not expressive enough and thus of limited use. For example, in order to illustrate this, it is impossible to define that a value has to be either between ten and twenty or forty and fifty (think of something like, e.g., frequency bands), it is just possible to define that it has to be between ten and fifty. A program understanding the ontology used in the annotations will be much more capable of validating the data. Since our use cases are fundamentally different from Zyp's this should not be understood as a critique towards Zyp's approach. It should, however, be highlighted that SEREDASj descriptions are completely compatible to Zyp's JSON Schema draft.

Figure 1 depicts the structure of a SEREDASj description. A description consists of metadata and a description of the structure of the JSON instance data representations it describes. The metadata contains information about the hyperlinks related to the instance data and prefix definitions to abbreviate long URIs in the semantic annotations to CURIEs [29]. The link descriptions contain all the necessary information for a client to retrieve and manipulate instance data. Additionally to the link's target, its media type and the target's SEREDASj description, link descriptions can contain the needed SEREDASj request description to create requests and semantic annotations to describe the link, e.g., its relation to the current representation. The link's target is expressed by link templates where the associated variables can be bound to an element in the instance data and/or linked to a conceptual model, e.g., a class or property in an ontology. The link template's variables can be further described by generic semantic annotations in the form of predicate-object pairs. The links' SEREDASj request description allows a client to construct the request bodies used in POST or PUT operations to create or update resources.

The description of the structure of instance representations (denoted as *element description* in the figure) defines the JSON data type(s) as well as links to conceptual models. Furthermore it may contain semantic annotations to describe an element further and, if the element represents either a JSON object or array, a description of its properties respectively items in term of, again, an element description. The structure of the JSON instance arises out of nested element descriptions. To allow

ISBN: 978-1-4577-0872-5 (c) 2011 IEEE

reuse, the type of an element description can be set to the URI of another model definition or another part within the current model definition. To address different parts of a model, a slash-delimited fragment resolution is used. For instance, `person.json#properties/address` refers to the `address` property defined in the SEREDASj description `person.json`.

In order to better illustrate the approach, a simple example of a JSON representation and its corresponding SEREDASj description are given in Listing 1. The example is a representation of a person and its friends from an imaginary social networking site. Without annotations the data cannot be understood by a machine and even for a human it is not evident that a friend's ID is in fact a hyperlink to a more detailed representation of that specific friend. SEREDASj solves those problems by describing all the important aspects of such a representation. In consequence, it is not only possible to extract the hyperlinks, but also to create a human-readable documentation of the data format and to translate the JSON representation to a RDF representation.

The SEREDASj description in Listing 1 contains two link definitions. The first one specifies the link to the friends' representations via their ID. It uses a URI template whose variable is bound to `#properties/knows/id`. This link definition shows also how further semantic annotations can be used; this is described in detail in section VIII. The second link specifies a search interface and is thus not bound to any element in the instance data; instead, the variable's model reference is specified. Again, this link is semantically annotated so that an agent will know that this link specifies a search interface.

The following description of the representation's structure basically maps the structure to the FOAF [30] ontology. The mapping strategy is similar to the table-to-class, column-to-predicate strategy of current relational database-to-RDF approaches [26]; JSON objects are mapped to classes, all the rest to predicates. By reusing the FOAF ontology wherever possible, the developer is able to exploit the already available, human-readable descriptions for the various elements. By just considering the FOAF ontology and its `rdfs:comment` annotations, the documentation shown in Table I, could be created completely automatically.

SEREDASj descriptions don't have to be complete, i.e., they do not need to describe every element in all details. If an unknown element is encountered in an instance representation

TABLE I. AN AUTOMATICALLY GENERATED DOCUMENTATION

A person. (`object`)		
id	`number`	
first_name	`string`	The first name of a person.
last_name	`string`	The surname of some person.
gender	`string`	The gender of this Agent (typically but not necessarily 'male' or 'female').
knows	`array`	A person known by this person (indicating some level of reciprocated interaction between the parties).
A person. (`object`)		
id	`number`	
name	`string`	A name for some thing.

Instance Data
http://example.com/user/556410

```
{
  "id": 556410,
  "first_name": "Markus",
  "last_name": "Lanthaler",
  "gender": "male",
  "knows": [
    { "id": 586807, "name": "Christian Gütl" },
    { "id": 790980, "name": "John Doe" } ]
}
```

SEREDASj Description
http://example.com/models/person.json

```
{
  "meta": {
    "prefixes": {
      "foaf": "http://xmlns.com/foaf/0.1/",
      "ex": " http://example.com/onto#",
      "owl": "http://www.w3.org/2002/07/owl#",
      "iana": "http://www.iana.org/link-relations/"
    },
    "links": {
      "/user/{id}": {
        "mediaType": "application/json",
        "seredasjDescription": "#",
        "semantics": {
          "owl:sameAs": "<#properties/knows>"
        },
        "variables": {
          "id": {
            "binding": "#properties/knows/id",
            "model": "[ex:id]"
          }
        },
        "requestDescription": "#"
      },
      "/user/search{?query}": {
        "mediaType": "application/json",
        "seredasjDescription": "personlist.json",
        "semantics": {
          "[iana:relation]": "[iana:search]" },
        "variables": {
          "query": { "model": "[foaf:name]" }
        }
      }
    }
  },
  "type": "object",
  "model": "[foaf:Person]",
  "properties": {
    "id": {
      "type": "number", "model": "[ex:id]" },
    "first_name": {
      "type": "string","model": "[foaf:firstName]" },
    "last_name": {
      "type": "string", "model": "[foaf:surname]" },
    "gender": {
      "type": "string", "model": "[foaf:gender]" },
    "knows": {
      "type": "array",
      "model": "[foaf:knows]",
      "items": {
        "type": "object", "model": "[foaf:Person]",
        "properties": {
          "id": {
            "type": "number", "model": "[ex:id]" },
          "name": {
            "type": "string", "model": "[foaf:name]" }
        }
      }
    }
  }
}
```

Listing 1. An exemplary JSON representation and its corresponding SEREDASj description

it is simply ignored. This way, SEREDASj allows forward compatibility as well as extensibility and diminishes the coupling. In this context it should also be emphasized that a SEREDASj description does not imply a shared data model between a service and a client. It just provides a description of the service's representations to ease the mapping to the client's data model.

VIII. FROM SEREDASJ TO RDF

Translating SEREDASj described JSON representations to RDF triples, and thus integrating it in the linked data cloud, is a straightforward process. The translation starts at the root of the JSON representation and considers all model references of JSON objects and tuple-typed arrays to be RDF classes, while all the other elements' model references are considered to be RDF predicates where the value of that element will be taken as object. If a representation contains nested objects, as the example in Listing 1, a slash-delimited URI fragment is used to identify the nested object. Semantic annotations in the form of the `semantics` property, as the one shown in the user's link in the example, contain the predicate and the object. The object might point to a specific element in the SEREDASj description and is eventually translated to a link in the instance data.

The automatic translation of the example from Listing 1 to RDF is shown in Listing 2. The person and its knows-relations are nicely mapped to the FOAF ontology. For each of the array items a new object URI is created by using a slash-delimited URI fragment. Eventually, those URIs are mapped to the user's "real" URI by the link's semantic annotation. Please note that the query link is not included in the RDF representation. The reason for this is that the query variable is not bound to any instance element and thus its value is unknown. In consequence, the translator is unable to construct the URI.

IX. CONCLUSIONS AND FUTURE WORK

In this paper we presented SEREDASj, a new approach to describe RESTful data services. In contrast to other approaches, such as OWL-S [10] or WSMO [31], we have placed strong emphasis on simplicity to lower the entry barrier. Web developers can use tools and knowledge they are mostly already familiar with. Since SEREDASj does not require any changes on the described Web service, it provides a viable upgrade path for existing infrastructure. To further foster the adoption of our approach, we presented several use cases as well as a concrete example of how our approach can be used to semantically describe a RESTful data service. By basing SEREDASj on RESTful services and making use of its uniform interface, the approach will be widely applicable.

There is still a conversely discussion in the community whether RESTful Web services need machine-readable descriptions and how such descriptions affect the coupling between a service and its clients. Our point of view is pragmatic. Descriptions inherently introduce coupling but without descriptions no interpretation of the exchanged data and thus no communication is possible. We argue that whether these descriptions are machine readable or not does not affect the degree of coupling between a service and its clients as long as the description does not include any implementation details such as too specific data types. A good example of such an over-

```
@base <http://example.com/user/556410> .
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ex: <http://example.com/onto#> .

<#> rdf:type foaf:Person .
<#> ex:id 556410 .
<#> foaf:firstName "Markus" .
<#> foaf:surname "Lanthaler" .
<#> foaf:gender "male" .

<#> foaf:knows <#knows/0> .
<#knows/0> rdf:type foaf:Person .
<#knows/0> ex:id 586807 .
<#knows/0> foaf:name "Christian Gütl" .

<#> foaf:knows <#knows/1> .
<#knows/1> rdf:type foaf:Person .
<#knows/1> ex:id 790980 .
<#knows/1> foaf:name "John Doe" .

<http://example.com/user/586807> owl:sameAs <#knows/0> .
<http://example.com/user/790980> owl:sameAs <#knows/1> .
```

Listing 2. The example in Listing 1 translated to RDF

specification and the resulting tight coupling is XML Schema which defines, e.g., a plethora of integer types (among others `byte`, `short`, `int`, `long`, `unsigned long`).

A limitation of the current proposal is that it is restricted to resources represented in JSON; no other media types are supported at the moment. In future work support should be extended to other formats such as, e.g., XML. Potentially this could be done by mapping XML representations to JSON as there are already promising approaches such as the JSON Markup Language (JsonML) [32] to do so. This would allow to transparently support XML representations without changing the current approach. Similarly, URI templates could be used to support the popular `application/x-www-form-urlencoded` media type.

In future work we would also like to create a tool suite for developers to support the creation of SEREDASj descriptions and, if needed, the automatic creation of domain ontologies with techniques similar to the ones used to create domain ontologies from relational databases [26]. Moreover, we would like to research aspects such as service discovery and composition which includes issues like authentication that might require the creation of a lightweight ontology to be described.

REFERENCES

[1] T. Vitvar and J. Musser, "ProgrammableWeb.com: statistics, trends, and best practices," Keynote of the Web APIs and Service Mashups Workshop at the European Conf. on Web Services (ECOWS), 2010.

[2] S.L. Huang, "After f8 - resources for building the personalized Web," Facebook Developer Blog, 2010. [Online]. Available: http://developers.facebook.com/blog/post/379.

[3] J. Lathem, K. Gomadam, and A.P. Sheth, "SA-REST and (S)mashups: adding semantics to RESTful services," in Proc. of the Int. Conf. on Semantic Computing 2007 (ICSC), IEEE 2007, pp. 469-476.

[4] J. Kopecký, K. Gomadam, and T. Vitvar, "hRESTS: an HTML microformat for describing RESTful Web services", in Proc. of the 2008 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology, 2008, pp. 619-625.

[5] M.J. Hadley, Web Application Description Language (WADL). 2009.

[6] M. Maleshkova and J. Kopecký, "Adapting SAWSDL for semantic annotations of RESTful services," On the Move to Meaningful Internet Systems: OTM 2009 Workshops, Springer, 2009, pp. 917-926.

[7] R. Battle and E. Benson, "Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST)," in Web Semantics: Science, Services and Agents on the World Wide Web, vol. 6, issue 1, 2008, pp. 61-69.

[8] Web Services Description Language (WSDL) Version 2.0. W3C Recommendation, 2007.

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

[10] D. Martin, M. Burstein, D. McDermott, et al., "Bringing semantics to Web services with OWL-S," in World Wide Web, vol. 10, issue 3, 2007, pp. 243-277.

[11] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel, "WSMO-Lite annotations for Web services," in Proc. of the 5th European Semantic Web Conf. (ESWC 2008), LNCS 5021, Springer, 2008, pp. 674-689.

[12] M. Lanthaler, M. Granitzer, and C. Gütl, "Semantic Web services: state of the art," in Proc. of the IADIS Int. Conf. on Internet Technologies & Society (ITS 2010), IADIS, 2010, pp. 107-114.

[13] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, "A note on distributed computing," Mountain View, California, USA, 1994.

[14] A. Alowisheq and D.E. Millard, "EXPRESS: EXPressing REstful Semantic Services," in Proc. of the 2009 IEEE/WIC/ACM Int. Joint Conf. on Web Intelligence and Intelligent Agent Technology, IEEE, 2009, pp. 453-456.

[15] OWL 2 Web Ontology Language. W3C Recommendation, 2009.

[16] R. Alarcón and E. Wilde, "Linking data from RESTful services," in Proc. of the 3rd Workshop on Linked Data on the Web (LDOWS 2010), 2010.

[17] The Atom Syndication Format, RFC4287, Internet Engineering Task Force (IETF), 2005.

[18] Clinton, D., OpenSearch 1.1 Draft 4. 2005. [Online]. Available: http://www.opensearch.org/Specifications/OpenSearch/1.1

[19] Google Data Protocol. Google Inc., 2010. [Online]. Available: http://code.google.com/apis/gdata/.

[20] Developer's Guide: JSON-C/JavaScript, YouTube, 2010. [Online] http://code.google.com/apis/youtube/2.0/developers_guide_jsonc.html.

[21] The application/json Media Type for JavaScript Object Notation (JSON), RFC4627, Internet Engineering Task Force (IETF), 2006.

[22] XML Core Working Group Public Page – Pubblications, XML Core Working Group, 2011, http://www.w3.org/XML/Core/#Publications

[23] J. Gregorio, "JSON, XML and the Web," BitWorking, 2010. [Online]. Available: http://bitworking.org/news/2010/12/json-xml-web

[24] Semantic Web Case Studies and Use Cases, W3C, 2010. [Online]. Available: http://www.w3.org/2001/sw/sweo/public/UseCases/

[25] Generate documentation from source code, Doxygen, 2010. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/.

[26] F. Cerbah, "Learning highly structured semantic repositories from relational databases: the RDBToOnto tool," in Proc. of the 5th European Semantic Web Conf. (ESWC 2008), Springer, 2008, pp. 777-781.

[27] 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.

[28] A JSON Media Type for Describing the Structure and Meaning of JSON Documents. 2010, http://tools.ietf.org/html/draft-zyp-json-schema-03.

[29] CURIE Syntax 1.0: A syntax for expressing Compact URIs, W3C Working Group Note. W3C, 2010, http://www.w3.org/TR/curie/.

[30] D. Brickley and L. Miller, FOAF Vocabulary Specification 0.98. 2010. [Online]. Available: http://xmlns.com/foaf/spec/.

[31] D. Roman, U. Keller, H. Lausen, and J.D. Bruijn, "Web Service Modeling Ontology," in Applied Ontology, vol. 1, issue 1, 2005, pp. 77-106.

[32] JSON Markup Language (JsonML), 2011. [Online]. http://jsonml.org/.

ISBN: 978-1-4577-0872-5 (c) 2011 IEEE