Linked Data and Linked APIs: Similarities, Differences, and Challenges*

Ruben Verborgh[1], Thomas Steiner[2], Rik Van de Walle[1], and Joaquim Gabarro[2]

[1] Ghent University – IBBT, ELIS – Multimedia Lab
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
{ruben.verborgh,rik.vandewalle}@ugent.be
[2] Universitat Politècnica de Catalunya – Department LSI
08034 Barcelona, Spain
{tsteiner,gabarro}@lsi.upc.edu

Abstract. In an often retweeted Twitter post, entrepreneur and software architect Inge Henriksen described the relation of Web 1.0 to Web 3.0 as: "*Web 1.0 connected humans with machines. Web 2.0 connected humans with humans. Web 3.0 connects machines with machines.*" On the one hand, an incredible amount of valuable data is described by billions of triples, machine-accessible and interconnected thanks to the promises of Linked Data. On the other hand, REST is a scalable, resource-oriented architectural style that, like the Linked Data vision, recognizes the importance of links between resources. Hypermedia APIs are resources, too—albeit dynamic ones—and unfortunately, neither Linked Data principles, nor the REST-implied self-descriptiveness of hypermedia APIs sufficiently describe them to allow for long-envisioned realizations like automatic service discovery and composition. We argue that describing inter-resource links—similarly to what the Linked Data movement has done for data—is the key to machine-driven consumption of APIs. In this paper, we explain how the description format RESTdesc captures the functionality of APIs by explaining the effect of dynamic interactions, effectively complementing the Linked Data vision.

1 Introduction

1.1 The Web API simplification movement

The number of Web APIs has increased at a tremendous rate during the past few years. ProgrammableWeb, a major catalog of Web APIs and services, consisted of 6,000 entries as of May 2012 [13], twice the amount compared to the year before [12]. More than 4,000 of those entries carry the label "REST", meaning they are light-weight Web APIs, also called HTTP interfaces, as opposed to the more heavy-weight RPC-style Web services, often using SOAP. While there are certainly different viewpoints to take into account—especially when comparing enterprise

* This paper is an extended version of the Linked APIs for the Semantic Web (LAPIS) workshop paper titled *"The Missing Links"* [34].

SOA architects and *mash-up* developers—Web developers in general welcome this simplification movement on the dynamic side of the Web. It is common practice to intermix Web APIs from different sources and, in the sense of emergence, create something new (commonly called mashup applications), where the whole is greater than the sum of its parts. If Web application development today was compared to the world of toys, the LEGO figures would ride the Playmobil horses and fight with Star Wars collectibles swords. However, a lot of manual plumbing is required to make this work: while Web APIs bare the potential to be composed straightforwardly, they lack the semantics to do this in an automated way [26].

In the past, we have introduced the Web API description method RESTdesc [35], which aims to provide the semantics necessary to enable automated API consumption and composition, in the same way that ontologies provide the semantics to static data. In this paper, we want to look at Web APIs from a Semantic Web perspective: investigating how APIs differ from data, what they have in common, and how they could work together on the Web. An important piece of the puzzle is to realize that the resource-oriented way of looking at APIs is similar to the Linked Data vision on data.

1.2 Linked Data explains the static side, RESTdesc the dynamic side

To clarify what we mean with this statement, we need to take a step back and think about the Web in its most abstract form. What we see are *resources*, with an unparalleled variety, and an ever increasing number of *links* between them [10]. Resources and their representations make up the essence of the Web [16], while the Linked Data vision made us all realize again the crucial role that links play therein. Indeed, links have been the catalysts of the success of the human Web, and they continue to prove their strengths on the Semantic Web [8]. The representations of resources—and therefore data—are given meaning by links, corresponding to well-defined RDF predicates.

Given the importance of links, one can wonder why they seem absent on the service-side of the Web, where interactions are mostly driven by static controls such as message templates and URI construction rules. These controls have to be known in advance, unlike Linked Data controls (*i.e.*, the links between resources), which are consumed at runtime. When Fielding redesigned the HTTP specification [15], he had a resource-oriented model in mind where hypermedia drives Web applications: Representational State Transfer (REST, [16]). He later clarified that the hypermedia constraint imposed by REST demands that representations of a resource should contain controls that guide hypermedia consumers to possible next steps or resources [14]. Consequently, modeling Web APIs the REST way leads to the same resources-and-links paradigm that is at the core of the human Web, which has HTML links and forms, and the Semantic Web, which has RDF links between resources.

In all fairness, REST APIs—as defined by Fielding—are scarce. While many APIs carry the "REST" label, few actually obey the hypermedia constraint, and, even worse, some of them do not correctly adhere to the defined HTTP semantics [26]. Hypermedia-driven APIs are vastly outnumbered by plain HTTP and RPC

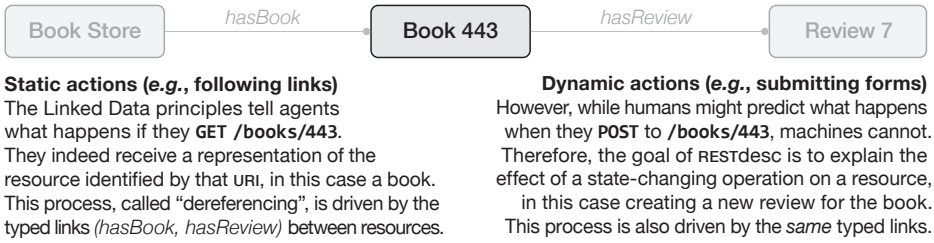

Fig. 1. RESTdesc complements Linked Data by explaining a hyperlink's *dynamic* functionality in machine-readable form. For instance, we can express with RESTdesc what happens when agents use `POST` on a linked resource instead of `GET`.

interfaces. However, this can be compared to the larger presence of unstructured and unlinked data on the Web compared to Linked Data. Therefore, the scarceness doesn't change the status of resource- and link-orientedness as well-suited model for automated agents to perform static *and* dynamic interactions.

Currently, the main obstacle for automated agents that want to consume Web APIs is that they cannot predict what effect a *state-changing* operation will have. Linked Data gives the answer for *information-retrieving* operations, known as *dereferencing*. Performing a `GET` operation on a resource's URI will provide the agent with information about that resource. But what happens when the agent performs a `POST` operation on the *same* resource? [35] Since Fielding suggests the controls (*e.g.,* links and forms) should point to possible next steps or resources, it is obvious *how* the state change happens. However, *what* this state change will bring might be obvious to humans, but is still unknown to machines. Therefore, in this paper, we zoom in on how the description format RESTdesc explains to agents what will happen if *state-changing* operations are performed on a resource, complementary to the Linked Data principles that explain the same for static operations.

This complementary nature is illustrated in Fig. 1, which positions Linked Data and RESTdesc. The example shows a book store that offers several books, each of which can have several reviews. The resources might be available on the Web for human visitors as HTML representations with (possibly typed) hyperlinks in between. To make the store machine-accessible, the server might additionally serve RDF representations, in which the relations are RDF predicates, which eventually can lead to Linked Data. However, the Linked Data principles only explain how to *browse* books and reviews, whereas the HTML representations provide the controls to *add* reviews. RESTdesc bridges this gap by explaining the functionality of this Web API, in a representation-independent way.

This paper starts with a description of related work in Section 2, then highlights the differences and similarities of Linked Data and hypermedia APIs in Section 3, zooming in on the gaps that need to be bridged. Section 4 continues with an illustration of the role RESTdesc can play herein by formally expressing the relationship between resources in a hypermedia API. Finally, Section 5 looks back on the discussed topics and ends by indicating the importance of hypermedia-driven APIs on the Web for autonomous agents.

2 Related work

Description of Web services or APIs for automated use has been on the Web since before the advent of the Semantic Web (notably WSDL [11]), and played an important part during the beginning of the Semantic Web's inception. Several of the first initiatives are well-known: OWL-S [29], which evolved from DAML-S [3], and the conceptually different WSMO [24,32]. These formats target what are called "Big" Web services [31], which function in a message-passing or Remote Procedure Call (RPC) paradigm. While these models use Semantic Web elements such as ontologies, they predate the Linked Data vision and the recent revaluation of REST APIs. Neither OWL-S nor WSMO have stood the test of time, as extensive Web searches did not reveal substantial real-world usage. We therefore focus on more recent research projects that have design goals similar to RESTdesc, *e.g.*, a focus on functionality and/or hypermedia APIs.

Several methods aim to enhance existing technologies to deliver annotations of Web APIs. HTML for RESTful Services (hRESTS, [21]) is a microformat to annotate HTML descriptions of Web APIs in a machine-processable way. SA-REST [18] provides an extension of hRESTS that describes other facets such as data formats and programming language bindings. MicroWSMO [22,25], an extension to SAWSDL that enables the annotation of RESTful services, supports the discovery, composition, and invocation of Web APIs. The Semantic Web sErvices Editing Tool (SWEET, [27]) is an editor that supports the creation of mashups through semantic annotations with MicroWSMO and other technologies. A shared API description model, providing common grounds for enhancing APIs with semantic annotations to overcome the current heterogeneity, has been proposed in the context of the SOA4All project [28].

The Resource Linking Language (ReLL, [1]) features media types, resource types, and link types as first class citizens for descriptions. It offers a metamodel and an associated XML Schema to capture these aspects formally. The RESTler crawler [1] finds RESTful services based on ReLL descriptions. The authors also propose a method for ReLL API composition [2] using Petri nets to describe the machine-client navigation.

Linked Open Services (LOS, [23]) have an HTTP API approach, in which SPARQL graph patterns identify the offered functionality. Part of the project's scope concerns the lifting and lowering of existing services, since many of them do not expose their data in a semantic format yet. A difference with RESTdesc is that LOS APIs are not committed to the hypermedia constraint, whereas the hypermedia-driven consumption of APIs is a central concept in RESTdesc.

Linked Data Services (LIDS, [33]) have a similar notion of input and output graphs. They use the input data to construct a resource's URI, as opposed to LOS, which sends input data in the request body. The result is an API whose interactions are thus in a sense solely *form*-based—the form structure being defined by the unbound variables in the input graph pattern. In addition to forms (not discussed in this paper), RESTdesc also aims to support the *link* part of the hypermedia control set.

3 A joint future for Linked Data and hypermedia APIs

We start this section with an essential definition to avoid misunderstandings on the thin ice of REST, RESTlike, and unRESTful APIs:

Hypermedia Web APIs are interfaces to retrieve and manipulate *resources* according to the HTTP method semantics, serving *representations* of these resources along with the *controls* to advance through the interface [14].[1]

Striking parallels between Linked Data and hypermedia APIs exist—and this is not a coincidence, since both are closely tied to the original visions and architecture of the Web. One of the common elements are **resources**: concepts in Linked Data are identified by one or multiple URIs, which, when requested through HTTP GET, lead to information about that concept. Hypermedia APIs are similarly structured as concepts or resources, with the constraints that every URI should identify a resource and that the HTTP methods should be used conform to the HTTP specification [15]. The semantics of the GET method have therein been defined as "obtaining the information identified by the URI", which, unsurprisingly, matches the Linked Data purpose [19].

The other common element are **links**: as the name implies, they play a vital role in Linked Data, and they are at the heart of the Semantic Web. Links give a concept's data meaning beyond its own context. More concretely, if an agent does not understand what a data property means, it can look up that property because its link is an HTTP URI. The same applies to hypermedia APIs: the controls, telling us how other resources relate to the current resource, can be links. Details on the nature of the relation are conveyed by link types, which can have the same URIs as Linked Data properties [30].

In essence, one could see the whole Linking Open Data Cloud [9] as a large, distributed hypermedia application. This is in fact how its usage is encouraged: an agent starts from one resource and can make its way through the whole cloud, just by "following its nose", thanks to the links. However, it only provides a subset of the possibilities of what we expect from a hypermedia API: merely *retrieval* operations are supported. Yet, the role of links here remains important: browsing billions of triples in billions of resources would otherwise prove difficult.

An interesting aspect of REST is that it does not matter whether the resources and triples already exist. They can either be part of documents, or be the result of a service invocation—but the agent does not have to know and does not have to care. For example, a huge dataset of natural numbers has been made available as Linked Data [38], yet the information of each number is not static, but instead generated dynamically when an agent dereferences its URI. This dataset is thus what we would traditionally consider a "service", but thanks to the REST principles, it manifests itself as just another set of linked resources.

[1] Hypermedia APIs are synonymous to "REST APIs or services, *in the sense as defined by Fielding*" [16]. This last clarification is important, since many APIs that were given a "REST" label do *not*, or only partially, adhere to Fielding's definition, which is why we use the term "hypermedia API" to distinguish the *intended* meaning [20].

Nevertheless, we often associate the concept of services additionally with action-driven behavior, for example, allowing us to post a comment or order tickets. In a REST architectural model, these actions are captured by the modification or creation of resources, linked to existing resources. While these and similar actions are very common on the human Web and on the Web of services, the Semantic Web still struggles with state-changing operations [7]. Several mechanisms are there (*e.g.,* SPARQL `UPDATE` [17]), but issues such as authorization and security still impede wide adoption. Consequently, the Linked Data vision must in the meantime assume that the publisher and consumer sides are distinct, *i.e.*, that consumers of Linked Data will not need to perform write operations. This simplifying assumption has its benefits—just look at the overwhelming amount of data—but will not be sufficient for the vision of autonomous agents that require actions in the real world. Indeed, as the comment and ticket examples indicate, many interactions we perform on a daily basis involve write actions. Therefore, in the next section, we will look at the requirements of agents for browsing full hypermedia APIs, which offer both information-retrieving and state-changing operations.

4 RESTdesc describes hypermedia links

4.1 Example scenario

As an example, let us consider the situation of Fig. 1. Starting from the book store's main URI, an agent discovers resources in a fully hypermedia-driven way. Its steps might be the following:

1. **`GET`** a representation of the index resource at `/`.
2. **Find** a `hasBook` link in this representation titled "*The Catcher in the Rye*".
3. **`GET`** a representation of this linked resource at `/books/443`.
4. **Find** a `hasReview` link in this representation.
5. **`GET`** a representation of this linked resource at `/books/443/reviews/7`.

This way of working is hypermedia-driven, because the agent only follows the representation-supplied controls (*e.g.*, links) to go from one step to the next.

4.2 Understanding the `GET` operations

As an introduction to RESTdesc, we will know discuss the RESTdesc description that is associated with the action of retrieving a book's representation. RESTdesc descriptions are expressed in Notation3 (N3, [5]), a small superset of RDF put forward by Tim Berners-Lee. N3 adds support for quantification, necessary to create statements concerning *all* resources instead of only specific ones. Without this explicit support, the quantifications should have to be expressed indirectly. One other possibility to express this is to wrap SPARQL expressions inside string literals, which is the method used by LIDS [33]. The quantification constructs in N3 enable to integrate the semantics directly, whereas for instance SPARQL expressions have to be interpreted separately.

```
@prefix ex: <http://example.org/book-store#>.
@prefix http: <http://www.w3.org/2011/http#>.

{
  ?store ex:hasBook ?book. 1
}
=>
{
  _:request http:methodName "GET"; 2
            http:requestURI ?book;
            http:resp [ http:body ?book ].

  ?book ex:hasTitle ?title; 3
        ex:hasAuthor ?author;
        ex:hasReview ?review.
}.
```

Listing 1. RESTdesc describes the act of retrieving a book by explaining the associated hypermedia link.

Listing 1 displays a description of the `GET` operation on the `hasBook` link type and serves as an illustration of several common aspects of RESTdesc descriptions. Every description is a logic implication. The logical foundations of N3 (N3Logic, [6]) define an operational semantics, *i.e.*, RESTdesc descriptions are N3 rules that can be instantiated and executed by a reasoner. As indicated in Listing 1, it is convenient to examine the description in three parts:

1 **IF** you obtain a book's URI from a `hasBook` hyperlink
2 **THEN** you can make a `POST` request to that URI
3 to retrieve a representation of this book.

Below, we discuss some important aspects of this description.

Firstly, the explicit quantification makes agent understand that the book in the antecedent and the conclusion are the same. The `?book` variable can be instantiated with a concrete instance. For example, if an agent finds a `hasBook` link from the store `/` to the book `/books/443/`, it can instantiate the description of Listing 1 into the RDF fragment in Listing 2. This fragment details the instructions an agent needs to execute. Since this request in these instructions has not been executed, the resulting values are not known yet. However, the reasoner has instantiated them with blank nodes (`title1`, `author1`, and `review1`). After a successful execution of the request, these blank nodes can be substituted by the actual data received from the server.

Secondly, it might seem strange at first sight that the request 2 is part of the consequent, and not of the antecedent. After all, it is the existence of the link 1 *and* the execution of the request 2 that lead to obtaining information about the book 3. However, RESTdesc adopts a different view here. In fact, it is because of the existence of the link 1, that a request exists 2 which will lead to the information 3. The word *exists* is important here: the description indeed states that a request *exists* that will deliver the information, not that *all* requests with

```
@prefix ex: <http://example.org/book-store#>.
@prefix http: <http://www.w3.org/2011/http#>.

</> ex:hasBook </books/443>.
_:request1 http:methodName "GET";
           http:requestURI </books/443>;
           http:resp [ http:body </books/443> ].

</books/443> ex:hasTitle _:title1;
             ex:hasauthor _:author1;
             ex:hasreview _:review1.
```

Listing 2. The instantiation of the RESTdesc description in Listing 1 yields an RDF fragment with instructions.

the given parameters will lead there. In other words: the request is existentially quantified, not universally. This notion is important, because it models the world more accurately. For example, a given request could fail because of connection issues or might require additional authentication.

Thirdly, RESTdesc does not specify what representations should look like. This is a central part of the REST philosophy. While RESTdesc describes the relations between resources and the result of actions performed on them, the selection of the right representation should happen at runtime by making use of the content negotiation mechanism of the HTTP protocol. For example, Listing 1 states that the retrieved resource will have a title and an author. The description does, however, *not* imply that this information will be supplied in RDF or any other format. While it seems logical that an agent would ask for an RDF representation (since the agent uses Semantic Web technologies internally), this is by no means a requirement. The actual representation could be in XML, JSON, HTML, or any other format. This opens possibilities to work with non-textual data, such as images and video [36]. However, the major benefit of RDF representations is that their contents are self-describing and can therefore be automatically interpreted by machines.

The final and most crucial remark is that the necessity of the description in Listing 1 can be questioned. After all, why would we want to describe a `GET` request? It seems unnecessary, because of the following two reasons: first, the Linked Data principles already tell us what happens with `GET` request—receiving a representation of the resource with the corresponding URI (which is called "dereferencing"). Second, even if these principles did not apply, an agent could safely execute the request, since the HTTP specification indicates `GET` should not change application state [15]. We fully agree here: RESTdesc is designed to describe state-changing operations whose result is resource-dependent, the primary verb being `POST`. Therefore, the next subsection illustrates a `POST` request, which fully illustrates RESTdesc's capabilities. RESTdesc *can* however be used for `GET`, which is interesting *a)* for situations where ontological constructs are insufficient to describe a complex resource relationship and *b)* to convey an expectation of what properties a representation will contain (*e.g.*, `hasTitle`, `hasAuthor`, ...).

```
@prefix ex: <http://example.org/book-store#>.
@prefix http: <http://www.w3.org/2011/http#>.

{
  ?store ex:hasBook ?book. 1
  ?review ex:author _:author;
          ex:rating _:rating;
          ex:contents _:text.
}
=>
{
  _:request http:methodName "POST"; 2
            http:requestURI ?book;
            http:body ?review;
            http:resp [ http:body ?book ].

  ?book ex:hasReview ?review. 3
}.
```

Listing 3. RESTdesc describes the act of posting a review by explaining the associated hypermedia link.

4.3 Understanding `POST` requests

The situation is completely different for `POST` requests because, unlike with `GET` and other safe requests, the agent cannot carelessly issue a `POST` request in one of the steps, since *a*) it cannot predict what the result will be and *b*) testing what the result is can have unwanted consequences, as `POST` is *unsafe* [15]. Furthermore, it cannot determine what body it should send along with the `POST` request. Although some representation formats provide forms (*e.g.*, HTML and Atom), others lack form functionality (*e.g.*, RDF, although proposals exist [4]), but in either case, it is unclear how the result relates to the submitted data.

Let us therefore examine the description in Listing 3, which can similarly be interpreted in three parts:

1 **IF** you obtain a book's URI from a `hasBook` hyperlink
2 **THEN** you can make a `POST` request to that URI
3 to add a review with the supplied parameters to this book.

This enables agents to understand what data they can send along with a `POST` request and how this data will influence the outcome of the request.

Note how, in this example, the precondition is more restricting: the agent needs to have access to a review before the request can be executed. Also, this review is necessary to construct the request: it should be placed inside the HTTP request's `POST` body. Again, the exact representation of this body is not detailed, because agents and servers should be able to agree on the best representation at runtime. We do, however, get a suggestion of properties that should be present in the representation: an author, a rating, and a review text.

Now that the agent understands each of the steps, it is able to chain them together and actually execute each of the requests in the process.

4.4 Executing the requests

Concretely, if the agent has been given the contents of a review *(author, rating, content)*, it can follow these hypermedia-driven steps:

1. **`GET`** the RESTdesc description of `hasBook`.[2]
2. **`GET`** a representation of the index resource at `/`.
3. **Find** a `hasBook` link in this representation titled "*The Catcher in the Rye*".
4. **Instantiate** the description with the review and found link.
5. **`POST`** the review, as instructed by the description, at `/books/443`.

Note again how only hypermedia controls are used to get from one step to the next. The added value of RESTdesc here is to explain the agent in advance what effect the `POST` request will have, so it can decide whether to execute this request. In real-world applications, RESTdesc descriptions can be used for goal-driven API compositions [37]. For instance, the user can supply the review parameters as input, and ask that it is submitted to a certain book.

5 Conclusion

The Linked Data vision strives to connect data on the Web, making it available in a machine-processable format. Hypermedia APIs similarly strive for connectedness of resources, but also consider the write side of interactions. Their goals are similar, and so are their tools: both make automated consumption of the Web available using the core principles of the HTTP architecture, featuring resources, representations, and links. However, dealing with state-changing operations requires automated agents to have expectations of what consequences their actions will have.

RESTdesc shows how existing Semantic Web technologies can be combined to explain the functionality of a Web API to those agents. It enables us to apply the Linked Data vision to hypermedia APIs by describing the meaning of links for state-changing operations. In that sense, it is a plea for more hypermedia APIs on the Web, as they beautifully incorporate the controls that future autonomous agents will need to browse the Web. Therefore, we believe it is time to transition today's services towards hypermedia APIs by adding the missing links.

Acknowledgments The described activities were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO), and the European Union.

This work was partially supported by the European Commission under Grant No. 248296 FP7 (I-SEARCH project). Joaquim Gabarró is partially supported by TIN-2007-66523 (FORMALISM), and SGR 2009-2015 (ALCOM).

[2] RESTdesc discovery, *i.e.*, how to obtain RESTdesc descriptions, has been discussed earlier [36]. The agent could for example dereference the `hasBook` link.

References

1. R. Alarcón and E. Wilde. RESTler: crawling RESTful services. In *Proceedings of the 19th international conference on World Wide Web*, pages 1051–1052. ACM, 2010. Available at `http://doi.acm.org/10.1145/1772690.1772799`.
2. R. Alarcón, E. Wilde, and J. Bellido. Hypermedia-driven RESTful service composition. In *Service-Oriented Computing*, volume 6568 of *Lecture Notes in Computer Science*, pages 111–120. Springer, 2011. Available at `http://dx.doi.org/10.1007/978-3-642-19394-1_12`.
3. A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the Semantic Web. 2342:348–363, 2002. Available at `http://eprints.soton.ac.uk/257342/1/ISWC2002-DAMLS.pdf`.
4. M. Baker. RDF forms, 2003–2005. Available at `http://www.markbaker.ca/2003/05/RDF-Forms/`.
5. T. Berners-Lee and D. Connolly. Notation3 (N3): A readable RDF syntax. W3C Team Submission, 2011. Available at `http://www.w3.org/TeamSubmission/n3/`.
6. T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3):249–269, 2008. Available at `http://arxiv.org/pdf/0711.1533v1.pdf`.
7. T. Berners-Lee, R. Cyganiak, M. Hausenblas, J. Presbrey, O. Seneviratne, and O. Ureche. Realising a read-write Web of Data, June 2009. Available at `http://web.mit.edu/presbrey/Public/rw-wod.pdf`.
8. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001. Available at `http://www.scientificamerican.com/article.cfm?id=the-semantic-web`.
9. C. Bizer. The emerging Web of Linked Data. *Intelligent Systems,* IEEE, 24(5):87–92, Sept. 2009. Available at `http://lpis.csd.auth.gr/mtpx/sw/material/IEEE-IS/IS-24-5.pdf`.
10. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009. Available at `http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf`.
11. F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web services Web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing,* IEEE, 6(2):86–93, Mar. 2002. Available at `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=991449`.
12. A. DuVander. 3,000 Web APIs: Trends from a quickly growing directory, Mar. 2011. Available at `http://blog.programmableweb.com/2011/03/08/3000-web-apis/`.
13. A. DuVander. 6,000 APIs: It's business, it's social and it's happening quickly, May 2012. Available at `http://blog.programmableweb.com/2012/05/22/6000-apis-its-business-its-social-and-its-happening-quickly/`.
14. R. T. Fielding. REST APIs must be hypertext-driven. Untangled – Musings of Roy T. Fielding, Oct. 2008. Available at `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`.
15. R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments: 2616, June 1999. Available at `http://tools.ietf.org/html/rfc2616`.
16. R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. *Transactions on Internet Technology*, 2(2):115–150, May 2002. Available at `http://dl.acm.org/citation.cfm?id=514185`.

17. P. Gearon, A. Passant, and A. Polleres. SPARQL 1.1 Update. W3C Working Draft, Jan. 2012. Available at http://www.w3.org/TR/sparql11-update/.
18. K. Gomadam, A. Ranabahu, and A. Sheth. SA-REST: Semantic Annotation of Web Resources. W3C Member Submission. Available at http://www.w3.org/Submission/SA-REST/.
19. O. Hartig and J. Zhao. Publishing and consuming provenance metadata on the Web of Linked Data. *Provenance and Annotation of Data and Processes*, pages 78–90, 2010. Available at http://www2.informatik.hu-berlin.de/~hartig/files/HartigZhao_Provenance_IPAW2010_Preprint.pdf.
20. S. Klabnik. REST is over, Feb. 2012. Available at http://blog.steveklabnik.com/posts/2012-02-23-rest-is-over.
21. J. Kopecký, K. Gomadam, and T. Vitvar. hRESTS: An HTML microformat for describing RESTful Web services. In *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology*, pages 619–625. IEEE Computer Society, 2008. Available at http://dx.doi.org/10.1109/WIIAT.2008.379.
22. J. Kopecký and T. Vitvar. MicroWSMO. WSMO Working Draft, Feb. 2008. Available at http://www.wsmo.org/TR/d38/v0.1/.
23. R. Krummenacher, B. Norton, and A. Marte. Towards Linked Open Services and Processes. In A. Berre, A. Gómez-Pérez, K. Tutschku, and D. Fensel, editors, *Future Internet – FIS 2010*, volume 6369 of *Lecture Notes in Computer Science*, pages 68–77. Springer Berlin / Heidelberg, 2010. Available at www.linkedopenservices.org/publications/FIS2010.pdf.
24. R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of WSMO and OWL-S. In L.-J. Zhang and M. Jeckle, editors, *Web Services*, volume 3250 of *Lecture Notes in Computer Science*, pages 254–269. Springer Berlin, 2004. Available at http://www.wsmo.org/2004/d4/d4.1/v0.1/20050106/d4.1v0.1_20050106.pdf.
25. M. Maleshkova, J. Kopecký, and C. Pedrinaci. Adapting SAWSDL for semantic annotations of RESTful services. In *Proceedings of the On the Move to Meaningful Internet Systems Workshops*, volume 5872 of *Lecture Notes in Computer Science*, pages 917–926. Springer, 2009. Available at http://dx.doi.org/10.1007/978-3-642-05290-3_110.
26. M. Maleshkova, C. Pedrinaci, and J. Domingue. Investigating Web APIs on the World Wide Web. In *Proceedings of the 8th European Conference on Web Services*, pages 107–114. IEEE, 2010. Available at http://sweet-dev.open.ac.uk/war/Papers/mmaWebAPISurvey.pdf.
27. M. Maleshkova, C. Pedrinaci, and J. Domingue. Semantic annotation of web APIs with SWEET. May 2010. Available at http://oro.open.ac.uk/23095/.
28. M. Maleshkova, C. Pedrinaci, N. Li, J. Kopecky, and J. Domingue. Lightweight semantics for automating the invocation of Web APIs. In *Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications*, Dec. 2011. Available at http://sweet.kmi.open.ac.uk/pub/SOCA.pdf.
29. D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan. Bringing semantics to Web services with OWL-S. *World Wide Web*, 10:243–277, Sept. 2007.
30. M. Nottingham. Web Linking. Request for Comments: 5988, Oct. 2010. Available at http://tools.ietf.org/html/rfc5988.
31. C. Pautasso, O. Zimmermann, and F. Leymann. RESTful Web services vs. "Big" Web services: making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, pages 805–814, New York, NY,

USA, 2008. ACM. Available at http://www.jopera.org/files/www2008-restws-pautasso-zimmermann-leymann.pdf.

32. D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1:77–106, January 2005. Available at http://dl.acm.org/citation.cfm?id=1412350.1412357.
33. S. Speiser and A. Harth. Taking the LIDS off data silos. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 44:1–44:4, New York, NY, USA, 2010. ACM. Available at http://www.aifb.kit.edu/images/4/4a/Triplify-2010-ssp-aha-taking-the-lids-off-data-silos.pdf.
34. R. Verborgh, T. Steiner, R. Van de Walle, and J. Gabarró Vallés. The missing links – how the description format RESTdesc applies the linked data vision to connect hypermedia APIs. In *Proceedings of the First Linked APIs workshop at the Ninth Extended Semantic Web Conference*, May 2012. Available at http://lapis2012.linkedservices.org/papers/3.pdf.
35. R. Verborgh, T. Steiner, D. Van Deursen, S. Coppens, J. Gabarró Vallés, and R. Van de Walle. Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In *Proceedings of the Third International Workshop on RESTful Design*. ACM, Apr. 2012. Available at http://www.ws-rest.org/2012/proc/a5-9-verborgh.pdf.
36. R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J. Gabarro. Capturing the functionality of Web services with functional descriptions. *Multimedia Tools and Applications*, 2012. Available at http://rd.springer.com/article/10.1007/s11042-012-1004-5.
37. R. Verborgh, D. Van Deursen, E. Mannens, C. Poppe, and R. Van de Walle. Enabling context-aware multimedia annotation by a novel generic semantic problem-solving platform. *Multimedia Tools and Applications*, 2012. Available at http://rd.springer.com/article/10.1007/s11042-010-0709-6.
38. D. Vrandečić, M. Krötzsch, S. Rudolph, and U. Lösch. Leveraging non-lexical knowledge for the Linked Open Data Web. *5th Review of April Fool's day Transactions*, pages 18–27, 2010. Available at http://km.aifb.kit.edu/projects/numbers/linked_open_numbers.pdf.