biblio.ugent.be

The UGent Institutional Repository is the electronic archiving and dissemination platform for all UGent research publications. Ghent University has implemented a mandate stipulating that all academic publications of UGent researchers should be deposited and archived in this repository. Except for items where current copyright restrictions apply, these papers are available in Open Access.

This item is the archived peer-reviewed author-version of:

Bottom-up Web APIs with self-descriptive responses

Ruben Verborgh, Erik Mannens, and Rik Van de Walle

In: Proceedings of the First Karlsruhe Service Summit Research Workshop, 2015.

http://service-summit.ksri.kit.edu/downloads/Verborgh2015-_Bottom-up_Web_APIs_with_self-descriptive_responses.pdf

To refer to or to cite this work, please use the citation to the published version:

Verborgh, R., Mannens, E., and Van de Walle, R. (2015). Bottom-up Web APIs with self-descriptive responses. *Proceedings of the First Karlsruhe Service Summit Research Workshop*

VISION PAPER

Bottom-up Web APIs with self-descriptive responses

Ruben Verborgh, Erik Mannnens, Rik Van de Walle
{firstname.lastname}@ugent.be – Ghent University – iMinds, Belgium

The success or failure of Semantic Web services is non-measurable: many different formats exist, none of them standardized, and few to no services actually use them. Instead of trying to retrofit Web APIs to our models, building APIs in a different way makes them usable by generic clients. This paper argues why we should create Web APIs out of reusable building blocks whose functionality is self-descriptive through hypermedia controls. The non-functional aspects of such APIs can be measured on the server and client side, bringing us to a more scientific vision of agents on the Web.

1 The failed promise of Semantic Web services

The initial Semantic Web vision talks quite expressively about how *intelligent agents* will improve our lives by doing things on the Web for us (Berners-Lee, Hendler, & Lassila, 2001). Unfortunately, after roughly 15 years of Semantic Web research, nothing even vaguely resembling the envisioned agents has come out of our research community. An important reason seems the apparent failure of Semantic Web services (Pedrinaci, Domingue, & Sheth, 2011) to gain any traction or usage. Even if there were agents, they would not have any services at their disposal. Companies such as Apple and Google have released software that seemingly acts as a remarkably clever agent; however, each binding with a Web service appears hand-made and is thus still far away from agents that dynamically interact with automatically discovered services. It seems that one of the only successes of the Semantic Web is—paradoxically—intelligent *servers*, with SPARQL endpoints (Feigenbaum, Williams, Clark, & Torres, 2013) providing ad-hoc queryable semantic knowledge. Sadly, even that can hardly be called a success: we found out the hard way that such omnipotent Web services result in an unacceptably low availability (Buil-Aranda, Hogan, Umbrich, & Vandenbussche, 2013), making them one of the likely contributors to the rather negative external perception of Semantic Web technologies. After all, if the average "intelligent" server is unavailable for more than 1.5 days each month, how much remains for the intelligent agents?
Somewhere, something went horribly wrong. The main culprit might well be the way we have been building Web services so far. Oddly enough, "Web" services have surprisingly little to do with the Web. The Web, at its core, is a distributed hypermedia system (Fielding, 2000), a collection of three separate inventions that closely connect together:

- Resources on the Web are identified by a URL.
- Through the HTTP protocol, we can retrieve a representation of a resource via its URL.
- Representations in a hypermedia format such as HTML contain the URLs of related resources.

This recursive mechanism, characteristically driven by hypermedia, sets the Web apart from all other information systems. Traditional Semantic Web services, on the other hand, use the Web's core mechanisms in unrelated ways:

- A URL identifies an endpoint (i.e., a purely technical artefact of a process or implementation, instead of information).
- An HTTP message serves as an envelope to call procedures on that endpoint.
- Hyperlinks are not relevant at all; URLs are hard-coded, sometimes in separate description documents.

Indeed, such services treat the Web simply as a black box to perform remote-procedure calling (RPC); the usage of HTTP and URLs is merely an artefact to have things working through TCP port 80. The same functionality can be (and has been) recreated with, for instance, the SMTP e-mail protocol (Cunnings, Fell, & Kulchenko, 2001). Therefore, traditional Web services are by no means native Web citizens at all, sharing none of their principles with the rest of the Web.

Web-compliance by itself could be seen as simply a matter of technical purity, but the opposite is true. If we create such an artificial world within the Web, it is quite meaningless to talk about Semantic *Web* agents. After all, to what extent do they belong to the field of Web research if they simply execute pre-programmed remote procedures over TCP?

2 The false hope of service descriptions

From their inception, Semantic Web services had been associated with verbose descriptions. With Web services in essence being firewall-friendly abstraction layers over RPC, a mechanism to describe the *semantics* of such procedures was necessary. Both OWL-S and WSMO claimed technical superiority (Lara, Roman, Polleres, & Fensel, 2004), but none of the proposals ever achieved W3C standardisation, let alone adoption or usage. Furthermore, the manual work to generate such descriptions by far outweighs their benefits, given the non-existence of agents that could use them.

In an effort to reduce the verbosity surrounding Web services, the service landscape moved towards services with a smaller payload, abandoning the XML domination that had reigned so far. Instead of implementing a protocol such as SOAP *on top* of HTTP, those Web APIs directly operate through the HTTP protocol. This gave rise to a new generation of service descriptions such as Linked Open Services (Krummenacher, Norton, & Marte, 2010), and Linked Data Services (Speiser & Harth, 2011). Such descriptions accept the heterogeneity of HTTP interfaces and employ so-called *lifting* and *lowering* to convert between the non-RDF responses of the interfaces to the RDF model and vice-versa. The descriptions themselves rely on graph patterns to capture input and output patterns, indicating that an RPC way of thinking (a method call) still underpins their design. While this is not technological burden, it does not bring us closer to a vision of agents on the *Web*.

To realize this, we have to look at the Web's architectural properties and the exact point where it currently fails for machines. Contrary to what current practice seems to imply, machines have no inherent need for RPC interfaces; they are certainly capable of consuming hypermedia interfaces as offered by the Web. What they cannot do at present, is parsing the natural language found in the majority of Web documents. Hence, machines need information in a machine-interpretable model such as RDF, but this does not warrant an entirely different interface. Instead, through the content negotiation feature of HTTP, the same conceptual HTTP resources and URLS can be shared between machine and non-machine interfaces, with only the representation having a different format (Verborgh et al., 2015). Clients consume such hypermedia APIS by navigating the links inside of their representations. However, since links only allow to look ahead one step at a time, automated clients have difficulties performing multi-step tasks. To mitigate this, functional hypermedia API description formats such as RESTdesc (Verborgh et al., 2012) explain a hypermedia API's resources by detailing the affordances offered by its links. Data-Fu (Stadtmüller, Speiser, Harth, & Studer, 2013) follows the example of RESTdesc by using rules and the HTTP vocabulary to deal with remote functionality. However, its rules live on the client side, so it is unclear to what extent it can withstand change on the server side—something that is a frequent practice on the Web.

There are three major issues with all of the above approaches, regardless of whether they deal with RPC or hypermedia:

1. Creating descriptions or rules remains manual work, which will realistically not be undertaken given the lack of a single standard and actually implemented use cases.
2. They all rely on the premise that *reasoning* will fix all remaining gaps. If there are ontological differences between the desired goal and/or different services or descriptions used, such issues are implicitly assumed to be trivial and entirely solvable by existing reasoning technologies. As a result, pressing problems remain unresolved.
3. No scientific methodology exists to compare the different description techniques. Most of the aforementioned technologies have claimed to be better than one another, but such claims lack scientific evidence. No metrics have been defined, and existing evaluations (e.g., Stadtmüller et al., 2013) examine the performance of a resulting system, but fail to give a quantifiable measure of the appropriateness of any Web interface as such. Yet the Semantic Web research community needs quantifiable results upon which we can build (Bernstein & Noy, 2014).

3 Fostering reusability through a self-descriptive bottom-up approach

Lacking better measurements, the Web API community has been heading the same quantity-over-quality course that has characterized the first years of the Linked Data initiative. An often-quoted fact in Web API papers and articles is the ever increasing number of Web APIs (Figure 1), which is supposed to be an indicator of the ecosystem's excellent health. However, as Linked Data researchers have become painfully aware, quantity only loosely correlates with quality or usefulness. Perhaps for Web APIs, the correlation between quantity and utility could even be negative. Few other communities would pride themselves on the existence of more than 12.000 different micro-protocols to achieve essentially the same thing: communicating between clients and servers over HTTP. Of course, each application has its own domain and domain-specific vocabulary, but does that also warrant an entirely different way of exposing this, especially when we have RDF as a uniform data model? Each different API currently requires a different client, given the lack of a uniform API description format to explain the API's response structure and functionality. Clearly, this approach to Web APIs is a dead end.

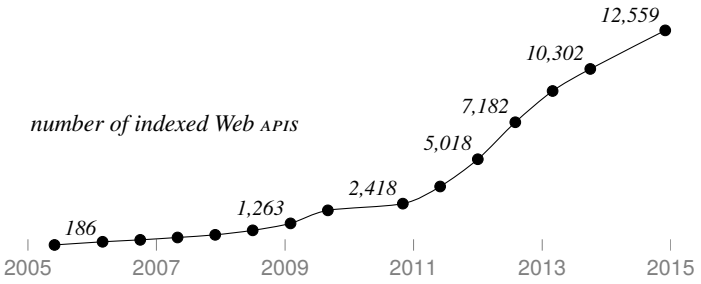


Figure 1: The increasing number of Web APIs is often named an indicator of their success, while the overgrowth of such *custom* micro-protocols is unnecessary—and detrimental to the development of *generic* Web API clients. *(data: programmableweb.com)*

In order for machines to use information autonomously, it has to be composed out of pieces they can recognize and interpret. The RDF model achieves this by identifying each of the triple components by reusable IRIs, which have a meaning beyond the scope that mentions them. Furthermore, the Linked Data principles mandate the use of HTTPURLs, which turn these components into affordances toward relevant information. For instance, given the following RDF triple:

```
<http://dbpedia.org/resource/Bill_Clinton> <http://xmlns.com/foaf/0.1/knows>
    <http://dbpedia.org/resource/Al_Gore>.
```

the knowledge of the `foaf:knows` predicate is sufficient for a machine to determine that this relation is symmetric, and that `dbpedia:Bill_Clinton` and `dbpedia:Al_Gore` are instances of `foaf:Person`—even though it might have never encountered any of those IRIs before. Furthermore, should the `foaf:knows` property be unfamiliar, its IRI can be dereferenced to find this information expressed in ontological predicates. Knowledge of these predicates in turn allows an interpretation of `foaf:knows` and hence the aforementioned derivation. We herein recognize two characteristics in particular:

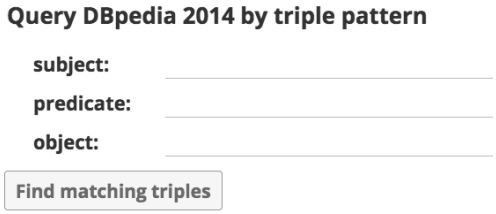
- The information is structured in a **bottom-up** way: machines interpret a larger unit of information through its pieces instead of interpreting the pieces through the whole (while humans are capable of doing both simultaneously).
- Each piece in the unit is **self-descriptive**: anything needed to interpret a piece is contained within itself, with its IRI acting as both an identifier and a direct handle towards additional interpretation mechanisms. No external resource is required beforehand, given the knowledge of a limited set of basic concepts.

This sharply contrasts with current practice for Web APIs. Machines are assumed to interpret each API operation in its entirety, as such smaller pieces do not exist, and API descriptions—if present—are external documents that must be collected and interpreted before consumption is possible. While this does not imply the inviability of such an approach, it raises serious doubt as to whether that is the most effective strategy towards automated Web API consumption by generic clients.

3.1 Making APIs self-descriptive

In order to answer the question of what bottom-up, self-descriptive Web APIs should look like, we can find inspiration in how the human Web works. Self-descriptiveness exists on two levels: on the one hand, humans understand natural language, so written texts can guide them through a webpage. On the other hand, they interact with *hypermedia controls* such as links and form elements provided through the `<a>` and `<input>` HTML tags, which avoids the need to read a separate document to find out how to craft HTTP requests against a particular server.

The first level of self-descriptiveness is provided by RDF: instead of describing content in natural languages, machines retrieve a machine-interpretable representation of the response—similar to how French-speaking people would receive a French-language representation. Analogously, we can also apply the same strategy of hypermedia controls to address the second level. Instead of providing the HTTP controls in HTML, we need a hypermedia ontology for RDF such as the Hydra Core Vocabulary (Lanthaler, 2013). Figure 2 shows how an HTML form can be represented in RDF, preserving the exact same semantics. Note in particular how each of the English field labels corresponds to an RDF property, which machines can apply like any other RDF construct. Furthermore, the semantics of "searching parts in a larger set" is expressed using the `hydra:search` predicate, so that an automated client can interpret the consequences of its actions.




```
<http://fragments.dbpedia.org/2014/en#dataset> hydra:search [
    hydra:template "http://fragments.dbpedia.org/2014/en{?s,p,o}";
    hydra:mapping
        [ hydra:variable "s"; hydra:property rdf:subject ],
        [ hydra:variable "p"; hydra:property rdf:predicate ],
        [ hydra:variable "o"; hydra:property rdf:object ]
    ].
```

Figure 2: An HTML form can be expressed in an RDF equivalent through the Hydra Core Vocabulary.

3.2 Building APIs from the bottom up

While humans adapt much better to unknown conditions than machines, we often also resort to recognizable patterns in unfamiliar environments such as new Web pages. They include frequently reused types of forms, such as search or contact forms, and usability conventions such as underlined hyperlinks. To appreciate the strength of such factors, consider how we have become accustomed to the fairly recent trend of clicking the main logo on any subpage to navigate to the website's homepage (Cappel & Huang, 2007).

Given the Web API ecosystem's total lack of reuse on the macro level, it is not surprising that reuse is lacking on the micro level as well. For instance, despite being highly similar in intent, each social media site offers a radically different API to post an updated status message—even though the human counterparts in HTML look and behave fairly similar. This is unintuitive, as machines actually need *more* structure than humans, precisely because they experience increasing difficulty when adapting to new situations.

The need for reusable building blocks can also be addressed by hypermedia controls, given that they are carefully designed in a progressive way. Again, the structure of webpages can inspire us here: we should provide smaller groups of hypermedia controls than can be reused in different combinations, with a preference of giving more hypermedia controls to users (and certainly not less). For instance, a common practice on current Web APIs is to have a dedicated entry point, which exposes significantly more functionality than other resources. However, this contrasts with websites: while there is indeed an index page (often located at `/`), usually *all* of the pages carry navigational controls that provide access to the entire website. In that sense, no page is more special than any other. Another issue is that people often omit "trivial" metadata which they would include for humans: how many result pages there are, how these results can be filtered and sorted, and so on. If we want to see intelligent generic clients, server have to *enable* such intelligent behavior by providing the necessary affordances instead of making use-case-specific assumptions about their necessity.

4 Transforming Web API engineering into measurable research

Such self-describing, bottom-up APIs bring us to the question on how a particular approach can be evaluated from a research perspective. In other words, we need to define objectively quantifiable metrics such that two or more Web APIs used for the same task can be compared in a reproducible manner. If such metrics do not exist, as has been the case in the majority of Web API papers and articles so far, no matter how good the engineering contributions are, there is no way for others to analyze or improve existing solutions (Bernstein & Noy, 2014).

As applied in our previous work on a self-describing, bottom-up Web API (Verborgh et al., 2014), we propose to measure Web APIs as follows. First, since *backward compatibility* (defined here as the ability of a client to interact with future versions of an API) cannot be quantified generically, we require full backward compatibility if a modified version of a Web API is proposed. Furthermore, the amount of out-of-band knowledge needed to consume the same part of an API cannot increase. Should these constraints not be satisfied, the new and old APIs must be considered different and any improvements are not characteristics of the original API. Second, APIs cannot be evaluated in isolation, as their non-functional characteristics depend on a particular use case. As such, when discussing the results of API evaluations, *external validity* should be thoroughly argued. Finally, the measurements must take into account quantifiable parameters, such as:

- The **average number of HTTP requests** it takes for a client to solve (a particular task of) a use case.
- The **average response size**.
- The **average response time** of the server.
- The **amount of processor and memory usage** on the client and server in function of parameters such as the concurrent number of clients and the complexity of use case tasks.
- The **cache hit rate** of responses.
- The **total completion time** of (particular tasks of) a use case.

Backward compatibility is an especially important criterion to avoid artificial manipulation of the above measurements. For instance, removing hypermedia controls from responses would certainly improve the average response size and, consequently, also the response time of the server. Yet this would mean an increase in out-of-band knowledge and/or breakage of previous clients. In order to progress toward an objectively verifiable and improvable scientific discipline, the utilization of such idiosyncratic shortcuts must be ruled out.

Figure 2 depicts the hypermedia form of a real-world hypermedia API that provides domain-agnostic access to triples. Following the principles above, we have analyzed the concrete use case of SPARQL query execution through this triple pattern fragments API (Verborgh et al., 2014) and the SPARQL protocol (Feigenbaum et al., 2013). Backwards compatibility is ensured by self-descriptiveness and the API's bottom-up structure. For instance, the current API only allows lookups with exact matching of triple components. Should the API be extended with full-text search, an additional hypermedia control would be added to facilitate this. That would guarantee compatibility with existing clients (since the triple-pattern "building block" continues to exist), while at the same time, clients that can interpret the full-text control (the new "building block") can realize a faster execution time for several queries. In this particular case, we have shown that the choice of interface influences the trade-offs between some of the parameters listed above: SPARQL query answering through a triple pattern fragments interface requires significantly more HTTP requests compared to when the server offers a SPARQL interface, but responses are more cacheable and thus withstand a growing number of clients much better.

The example indicates how self-describing, bottom-up APIs allow a gradual evolution, with different clients using different parts of the interface—just like humans do on the Web. Reusing such building blocks ensures that similar use cases can be tackled with recurring strategies. Furthermore, given the evaluation strategy introduced above, the effectiveness or fitness of a Web API for a certain use case can be quantified. This enables a more meaningful debate in the Web API community, beyond a hollow description format discussion that only pushes the vision of autonomous agents on the Web further away.

References

Berners-Lee, T., Hendler, J., & Lassila, O. (2001, May). The Semantic Web. *Scientific American*, *284*(5), 34–43. Retrieved from http://www.scientificamerican.com/article.cfm?id=the-semantic-web

Bernstein, A., & Noy, N. (2014). *Is this really science? The Semantic Webber's guide to evaluating research contributions* (Tech. Rep. No. IFI-2014.02). Department of Informatics, University of Zurich. Retrieved from https://www.merlin.uzh.ch/contributionDocument/download/6915

Buil-Aranda, C., Hogan, A., Umbrich, J., & Vandenbussche, P.-Y. (2013, November). SPARQL Web-querying infrastructure: Ready for action? In *Proceedings of the 12th international semantic web conference.* Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-41338-4_18

Cappel, J. J., & Huang, Z. (2007). A usability analysis of company websites. *Journal of Computer Information Systems*, *48*(1), 117–123.

Cunnings, R., Fell, S., & Kulchenko, P. (2001, November). *SMTP transport binding for SOAP 1.1* (Tech. Rep.). Retrieved from http://www.pocketsoap.com/specs/smtpbinding/

Feigenbaum, L., Williams, G. T., Clark, K. G., & Torres, E. (2013, March 21). *SPARQL 1.1 protocol* (Recommendation). W3C. Retrieved from http://www.w3.org/TR/sparql11-protocol/

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation, University of California). Retrieved from http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Krummenacher, R., Norton, B., & Marte, A. (2010). Towards Linked Open Services and processes. In *Future internet symposium* (Vol. 6369, pp. 68–77). Springer. doi: 10.1007/978-3-642-15877-3_8

Lanthaler, M. (2013). Creating 3rd generation Web APIs with Hydra. In *Proceedings of the 22nd international conference on world wide web companion* (pp. 35–38). Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee. Retrieved from http://dl.acm.org/citation.cfm?id=2487788.2487799

Lara, R., Roman, D., Polleres, A., & Fensel, D. (2004). A conceptual comparison of WSMO and OWL-S. In L.-J. Zhang & M. Jeckle (Eds.), *Web services* (Vol. 3250, pp. 254–269). Springer.

Pedrinaci, C., Domingue, J., & Sheth, A. (2011). Semantic Web services. In J. Domingue, D. Fensel, & J. Hendler (Eds.), *Handbook of Semantic Web technologies* (pp. 977–1035). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-92913-0_22 doi: 10.1007/978-3-540-92913-0_22

Speiser, S., & Harth, A. (2011). Integrating Linked Data and services with Linked Data Services. In *The Semantic Web: Research and applications* (Vol. 6643, pp. 170–184). Springer. doi: 10.1007/978-3-642-21034-1_12

Stadtmüller, S., Speiser, S., Harth, A., & Studer, R. (2013). Data-Fu: a language and an interpreter for interaction with read/write linked data. In *Proceedings of the 22nd international conference on world wide web* (pp. 1225–1236). Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee. Retrieved from http://dl.acm.org/citation.cfm?id=2488388.2488495

Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., … Van de Walle, R. (2014, October). Querying datasets on the Web with high availability. In P. Mika et al. (Eds.), *Proceedings of the 13th international semantic web conference* (Vol. 8796, pp. 180–196). Springer. Retrieved from http://linkeddatafragments.org/publications/iswc2014.pdf doi: 10.1007/978-3-319-11964-9_12

Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Gabarró Vallés, J., & Van de Walle, R. (2012, April). Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In *Proceedings of the third international workshop on RESTful design* (pp. 33–40). Retrieved from http://ws-rest.org/2012/proc/a5-9-verborgh.pdf

Verborgh, R., van Hooland, S., Cope, A. S., Chan, S., Mannens, E., & Van de Walle, R. (2015, March). The fallacy of the multi-API culture: Conceptual and practical benefits of representational state transfer (REST). *Journal of Documentation*, *71*(2). Retrieved from http://freeyourmetadata.org/publications/named-entity-recognition.pdf