Enabling HATEOAS Through HTTP OPTIONS, Link Headers, And The HTTP Vocabulary In RDF

Thomas Steiner
Universitat Politécnica de Catalunya
Department LSI
08034 Barcelona, Spain
tsteiner@lsi.upc.edu

Jan Algermissen
NORD Software Consulting
Kriemhildstrasse 7
22559 Hamburg
info@nordsc.com

ABSTRACT

Categories and Subject Descriptors
H.3 [**Information Storage and Retrieval**]: On-line Information Services

General Terms
Experimentation

Keywords
REST, HATEOAS, HTTP

1. INTRODUCTION
In one of his blog posts[1] Roy T. Fielding complains about the common practice to call any and all HTTP-based interface a REST API. He names a concrete example and writes that it actually "screams RPC" [sic]. Fielding continues that REST APIs must be hypertext driven. He defines[2] *hypertext* (and compares it to the term *hypermedia*) as follows:

> When I say hypertext, I mean the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions. Hypermedia is just an expansion on what text means to include temporal anchors within a media stream; most researchers have dropped the distinction.
>
> Hypertext does not need to be HTML on a browser. Machines can follow links when they understand the data format and relationship types.

[1] `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`
[2] `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven#comment-718`

For REST purists Hypermedia As the Engine Of Application State (HATEOAS) is the last and the hardest step towards the full glory of REST.

1.1 Richardson Maturity Model
In the Richardson Maturity Model[3] Leonard Richardson describes four levels towards true REST. Level zero is about tunneling all data through HTTP with only one HTTP method (usually POST, sometimes also GET) to one and the same endpoint usually using Remote Procedure Calls (RPC) and neglecting any mechanisms of the Web. Level one introduces resources, so rather than talking to just one endpoint, several endpoints are used. Level two switches from just one HTTP method to more adequate methods, oftentimes aligned to the four functions of persistent storage, CREATE, READ, UPDATE, DELETE (CRUD). In addition to different HTTP methods, also different HTTP status codes are used in order to signal failures, conflicts, or success. Finally level three introduces hypermedia controls that give an answer to the question "*where* can one go next" and "*what* can one do next" after each request in form of links. A Web service or API designed along this principles can be autodiscovered by a user agent by simply following her nose. To a certain extent an API can be self-documenting.

The remainder of this paper is structured as follows:

2. ON HATEOAS
We have cited Fielding's to-the-point definition of *hypermedia/hypertext* above. The next big component in the acronym HATEOAS is *application state*. The problem, however, with *application state* is that it is understood differently by different people. We tend to a definition that is explained best with the example of pagination on a search engine results page. Assume each page contained ten results and a link to its direct successor and predecessor. If the current page has a link to page seven and page nine, it can be directly implied that the current page must be page eight, if, and only if, the relation and/or meaning of the links is well-known beforehand. Hence the application is in state eight, without the explicit need to serialize this state somehow.

2.1 Present Ways To Represent HATEOAS
There is no defined standard on how to represent HATEOAS. There are, however, common practices that we present in the

[3] `http://martinfowler.com/articles/richardsonMaturityModel.html`

following (not in any particular order).

2.1.1 Atom (RFC4287)

In the Atom Syndication Format [6] there is the `atom:link` element that defines a reference from an entry or feed to a Web resource. The structure of the element is as follows:

```
atomLink =
  element atom:link {
    atomCommonAttributes,
    attribute href { atomUri },
    attribute rel { atomNCName | atomUri }?,
    attribute type { atomMediaType }?,
    attribute hreflang { atomLanguageTag }?,
    attribute title { text }?,
    attribute length { text }?,
    undefinedContent
  }
```

The `@href` attribute has to contain the link's IRI (the response to the question "*where* can one go next"). The response to the question "*what* can one do next" can (not must) be given in the link's `@rel` attribute. Its value can be a predefined value[4], or an IRI for custom link relations.

2.1.2 Google Data Protocol

The Google Data Protocol[2] extends the Atom Publishing Protocol[3] for processing queries, authentication, and batch requests. The Atom Publishing is an application-level protocol for publishing and editing Web resources. It is based on HTTP transfer of Atom-formatted representations. There are two serializations available: XML and JSON[5]. The structure of the XML serialization is the same as in 2.1.1, the structure of the JSON serialization can be seen below:

```
"link": [{
  "rel": "...",
  "type": "...",
  "href": "..."
}]
```

The elements and attributes of the JSON serialization are a straight-forward mapping of the XML serialization. The main advantage of JSON is that it is directly usable in JavaScript.

2.1.3 (X)HTML With Or Without Forms

A regular human-readable (X)HTML page can serve as a hypermedia control, too. Contained links and potential surrounding textual can be understood by humans, while machines can process just the links. Forms can give further instructions on the *how* of the next steps. With forms allowed values for parameters, like, e.g., enumerations can be given.

[4]The current list of pre-defined link relations is maintained by the IANA at `http://www.iana.org/assignments/link-relations/link-relations.xhtml`.

[5]`http://code.google.com/apis/gdata/docs/json.html`

2.1.4 Form Technologies

We adopt the term "form technologies" from Leonard Richardson[6] to reference a subset of description languages and mechanisms that are commonly criticized as being brittle[7]. The goal of such form technologies like the Web Application Description Language (WADL)[8], RDF Forms[9], or even the Web Services Description Language Version 2.0 (WSDL 2.0)[10] is to describe the HTTP methods, parameters, and allowed values that are involved during a Web service request.

2.1.5 Media Types

Media type give detailed insights into *how* to process a representation. They outline which parts of the representation are links. If media types are defined in an open way (i.e., in a way that new data can be added without breaking old user agents that did not expect this new data), they can help decouple a service from its implementation. Media types are cheap to create[11] and are seen as the method of choice for implementing truely RESTful APIs.

3. LINK HEADERS

[5]

4. HTTP OPTIONS

> The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. [...]
>
> A 200 response SHOULD include any header fields that indicate optional features implemented by the server and applicable to that resource (e.g., Allow), possibly including extensions not defined by this specification. The response body, if any, SHOULD also include information about the communication options. The format for such a body is not defined by this specification, but might be defined by future extensions to HTTP. Content negotiation MAY be used to select the appropriate response format.

[1]

5. THE HTTP VOCABULARY IN RDF

[4]

6. CONCLUSION

7. ACKNOWLEDGMENTS

This work is partially funded by the EU FP7 I-SEARCH project (project reference 248296).

[6]`http://www.crummy.com/writing/speaking/2008-QCon/act3.html`

[7]See, e.g., `http://bitworking.org/news/193/Do-we-need-WADL` for WADL.

[8]`http://www.w3.org/Submission/wadl/`

[9]`http://www.markbaker.ca/2003/05/RDF-Forms/`

[10]`http://www.w3.org/TR/wsdl20/`

[11]See `http://tools.ietf.org/html/rfc2048\#section-2.1.2` and http://tools.ietf.org/html/rfc2048#section-2.1.3 of RFC2048.

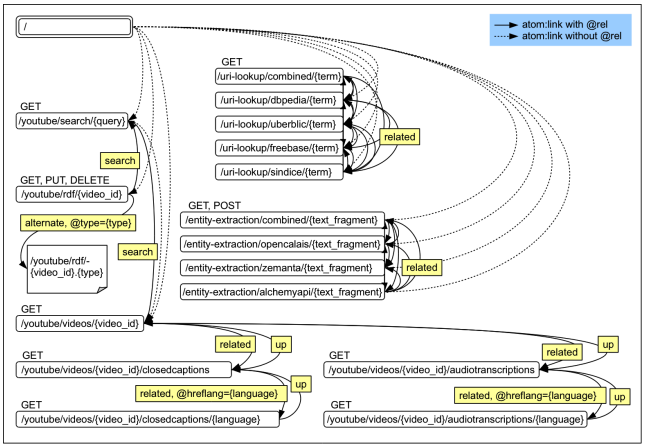

Figure 1: State machine for a Web service with link relations, and allowed HTTP methods.

8. REFERENCES

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. `http://tools.ietf.org/html/rfc2616`.

[2] Google. Google Data Protocol, retrieved on February 10, 2011. `http://code.google.com/apis/gdata/`.

[3] J. Gregorio and B. de hÓra. The Atom Publishing Protocol, October 2007. `http://tools.ietf.org/html/rfc5023`.

[4] J. Koch and C. A. Velasco. HTTP Vocabulary in RDF 1.0, 29 October 2009. `http://www.w3.org/TR/HTTP-in-RDF/`.

[5] M. Nottingham. Web Linking, October 2010. `http://tools.ietf.org/html/rfc5988`.

[6] M. Nottingham and J. Sayre. The Atom Syndication Format, December 2005. `http://tools.ietf.org/html/rfc4287`.