Enabling the Hypermedia Constraint 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
In this position paper we present a way how the hypermedia constraint from REST can be enabled using a combination of Link headers, the HTTP OPTIONS method, and the HTTP Vocabulary in RDF. We have implemented our approach in a Web service of ours, and discuss the learnings in the second part of the paper.

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

General Terms
Experimentation

Keywords
REST, Hypermedia Constraint, 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.

The essential constraint of REST is the hypermedia constraint, which in the Richardson Maturity Model (see the following section 1.1) is described as the last step towards the full glory of REST.

1.1 Richardson Maturity Model (RMM)
In the Richardson Maturity Model[3] (RMM) 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.

The authors, however, prefer not to use the RMM because it implies that the three lower levels induce a researched set of properties that imply that there is a known/assessable value in applying only a subset of the REST constraints.

1.2 On the Hypermedia Constraint
We have cited Fielding's to-the-point definition of *hypermedia/hypertext* above. Next, we need to define *application state*. The problem with *application state*, however, 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

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

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. The state machine of an application is not defined by the service, but by the user agent. In other words, the application comes into being by the choices the user agent makes, not by what the service intended.

2. PRESENT WAYS TO (PARTLY) FULFILL THE HYPERMEDIA CONSTRAINT

The hypermedia constraint is enabled by embedding controls (e.g., links, forms) in the representations made available to the client. There is no standard way to represent hypermedia controls, however, there are common practices that we present in the following section (not in any particular order).

2.1 Atom Syndication Format (RFC4287)

In the Atom Syndication Format [9] 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.

2.2 Google Data Protocol

The Google Data Protocol[4] extends the Atom Publishing Protocol[5] for processing queries, authentication, and batch requests. The Atom Publishing Protocol 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, the structure of the JSON serialization can be seen below:

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

[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`

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.

2.3 (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.

2.4 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. It is to be noted that WSDL 2.0 is a design-time description language, whereas WADL at least *can* be used as a run-time description language. We still decided to mention it because it can be used[11] to describe REST Web services.

2.5 Media Types

Media type give detailed insights into *how* to process a representation. They outline which parts of the representation are hypermedia controls. If media types are defined to be extensible (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. In order to register a new custom media type, a registration template can be submitted for review to the IANA[3]. Specific, i.e., exactly not `application/json`, `application/xml` or `text/xml`) are an essential aspect of achieving self descriptiveness and key for implementing truely RESTful APIs.

2.6 Link Headers

In the Web Linking[8] proposed standard document Mark Nottingham specifies relation types for Web links and how such links can be used with a Link header field in HTTP. In addition to that the document also defines a registry for link relations in section 6.2.2, therewith updating the relations defined in the Atom Syndication Format. Quoting from [8]:

> The Link entity-header field provides a means for serialising one or more links in HTTP headers. It is semantically equivalent to the <LINK>

[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]`http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/`

element in HTML, as well as the atom:link feed-level element in Atom [RFC4287].

A concrete example is shown below:

```
Link: <http://search.example.org/results/page7>;
    rel="previous"; title="previous results"
Link: <http://search.example.org/results/page9>;
    rel="next"; title="next results"
```

As outlined in the same example in section 1.2, the implication is that the current page is page 8, as the link relations `previous` and `next` refer to an ordered series of resources. We want to highlight that link relations are not limited to the set of registered relations, but can be any IRI. If we build the bridge to the world of Linked Data[1] where Tim Berners-Lee defines the four rules for Linked Data, we can see that there, too, URIs play a central role:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs, so that they can discover more things.

The idea is thus to combine link relations with meaningful Linked Data URIs. How powerful this is can be shown using an example: assume http://example.org/video.webm was a reference to a video file. We can then deliver a Link header along with the video data such as:

```
Link: <http://example.org/video.txt>;
    rel="http://dbpedia.org/resource/-
        Transcription_%28linguistics%29";
    title="transcription of the video";
    type="text/plain"
```

3. HTTP OPTIONS

The HTTP OPTIONS is one of the most basic ways to discover a resource. According to section 5.1.1 of the HTTP/1.1 specification[2] only the methods GET and HEAD must be supported by all general-purpose servers, all other methods are optional. The specification says about 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.

HTTP OPTIONS is currently not widely supported. While we have no statistically relevant data, a quick round-check (OPTIONS on the main domain, following potential redirects) of the search engines Yahoo!, Bing, and Google resulted in two 405 Method Not Allowed errors for Bing and Google, whereas Yahoo! delivered the Web site content as if the method was GET. The expected behavior can be seen on our university domain `http://www.upc.edu/`, including out-of-HTTP extensions:

```
$ curl  -i -X OPTIONS http://www.upc.edu/
HTTP/1.1 200 OK
Allow: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE,
    PROPFIND, PROPPATCH, MKCOL, COPY, [...]
Content-Length: 0
```

We could not find a server that returned a response body, however, using the Node.js[12] server with the Express framework[13] we succeeded in fully controlling both the Allow header and the response body:

```
$ curl -i -X OPTIONS http://localhost:3000
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Allow: GET, HEAD, OPTIONS
Content-Length: 14

I'M THE BODY!
```

4. THE HTTP VOCABULARY IN RDF

HTTP Vocabulary in RDF[7] defines a representation of HTTP in Resource Description Framework (RDF)[6]. It is intended to record HTTP or HTTPS request and response messages, including the various headers. Consider the following HTTP request:

```
$ curl -i http://dbpedia.org/resource/Barcelona
    -H "Accept: application/rdf+xml"
```

Modeled in RDF (in Turtle syntax[14], prefixes omitted for the sake of brevity) the request looks like this:

```
_:req a http:Request ;
  http:httpVersion "1.1" ;
  http:methodName "GET" ;
  http:mthd <http://www.w3.org/2008/http-
      methods#GET> ;
  http:abs_path "/resource/Barcelona" ;
  http:resp _:resp ;
  http:headers (
```

[12] http://nodejs.org/
[13] http://github.com/visionmedia/express
[14] http://www.w3.org/TeamSubmission/turtle/

```
[ http:fieldName "Host";
  http:fieldValue "dbpedia.org";
  http:hdrName <http://www.w3.org/2008/-
      http-header#host> ]
[ http:fieldName "Accept";
  http:fieldValue "application/rdf+xml";
  http:hdrName <http://www.w3.org/2008/-
      http-header#accept> ]
  ) .
```

Albeit verbous, the RDF triples represent the request in a protocol-compliant way. The response to the request could be modeled in the blank node `_:resp`.

5. IMPLEMENTATION AND LEARNINGS

Bringing all the technologies mentioned above together, first, we have Link headers to transparently inject data into a response without touching the body of the HTTP message. Second, we have HTTP OPTIONS as a means of discovering a resource on HTTP level. Third, we have the HTTP Vocabulary in RDF. which allows for HTTP communication to be modeled.

In order to test what can be done with regards to the hypermedia constraint with these technologies, we took an existing Web application of ours[15], and converted it into an API with the functionality to automatically annotate YouTube videos in RDF, where the input is a YouTube video ID, and the output an RDF document. In the course of implementing this API, a set of secondary wrapper APIs were implemented as well, one for YouTube video search based on a plaintext query, one for Named Entity Extraction (NEE) based on a text fragment, and one for URI Lookup (UL) based on a plaintext query. See Figure 1 for an overview of the structure of the services and the link relations between them. We serve the output of these wrapping services with the pseudo media types `application/prs.atom+json` for YouTube API-based data, with `application/prs.nee-entity+json` for NEE-based data, and with `application/prs.ul-entity+json` for UL-based data. The main problem of the Web service are the output constraints of our API: the video annotation service is designed to return RDF in one of its serializations, in consequence the media types are, e.g., `text/turtle`, or `application/rdf+xml`. The RDF/XML media type[16] describes the format, however, no concrete actions. With our Web service it is, however, possible to, e.g., modify an existing automatically generated RDF description of a video in order to correct things for example (via a PUT, or PATCH once implemented, with the new data). One can also DELETE a description, or retrieve the related audio transcript to a video, short, there are actions as potential next steps that are not described by the generic RDF/XML media type. This is where Link headers come into play. For the video annotation API, these are the Link headers that get sent when one accesses `http://localhost:3000/youtube/rdf/{video_id}`, where {video_id} must be a valid YouTube video id, e.g. `5irhWbQrFTE`:

```
Link: <http://localhost:3000/youtube/search/{query}>;
      rel="search"; title="search",
<>; rel="edit"; title="delete, modify",
<>; rel="alternate"; type="application/rdf+xml";
    title="application/rdf+xml",
<>; rel="alternate"; type="text/turtle";
    title="text/turtle",
<http://localhost:3000/youtube/videos/5irhWbQrFTE>;
    rel="related"; type="application/prs.atom+json";
    title="video metadata"
```

[15] `http://tomayac.com/semwebvid/`
[16] `http://tools.ietf.org/html/rfc3870`

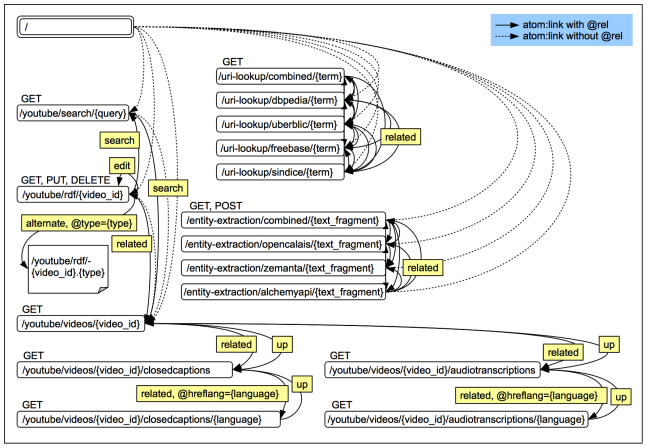

Figure 1: Overview of the Web service structure with link relations, and allowed HTTP methods.

6. RELATED WORK AND CONCLUSION

[10]

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] T. Berners-Lee. Linked Data, Juli 27, 2006. `http://www.w3.org/DesignIssues/LinkedData.html`.

[2] 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`.

[3] N. Freed and J. Klensin. Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, December 2005. `http://tools.ietf.org/html/rfc4289`.

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

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

[6] G. Klyne, J. J. Carroll, and B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, February 10, 2004. `http://www.w3.org/TR/rdf-concepts/`.

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

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

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

[10] J. Rees. New opportunities for linked data nose-following. W3C Blog, Juli 27, 2006. http://www.w3.org/QA/2010/07/new_opportunities_for_linked_d.html.