Fulfilling the Hypermedia Constraint Via HTTP OPTIONS, the HTTP Vocabulary In RDF, And Link Headers

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

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

ABSTRACT
One of the main REST design principles is the focus on media types as the core of contracts on the Web. However, not always is the service designer free to select the most appropriate media type for a task, sometimes a generic media type like application/rdf+xml (or in the worst case a binary format like image/png) with no defined or possible hypermedia controls at all has to be chosen. With this position paper we present a way how the hypermedia constraint of REST can still be fulfilled using a combination of Link headers, the OPTIONS method, and the HTTP Vocabulary in RDF.

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 HTTP-based interfaces REST APIs. Fielding emphasizes that REST APIs must be hypertext driven. In a comment on this post he defines *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.

Hence, 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[2] (RMM) Leonard Richardson describes four levels towards true REST. Level zero is about tunneling all data through HTTP with only one HTTP method to just one 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 states. 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. An API designed along these 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 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 a link to its direct successor and predecessor. If the current page has a link to page seven and page nine, there can be directly implied that the current page must be page eight, if,

[1] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[2] http://martinfowler.com/articles/richardsonMaturityModel.html

and only if, the relations of the links are known beforehand. In consequence the application is in state eight, without the explicit need to serialize this state somehow (and without the application even being aware of the existance of such state eight). 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. HYPERMEDIA CONTROLS

The hypermedia constraint is enabled by embedding hypermedia controls (e.g., links, forms) in the representations made available to the client. There is no standard way to represent these controls, however, some common practices.

2.1 Atom Syndication Format (RFC4287)

In the Atom Syndication Format [8] there is the `atom:link` element that defines a reference from an entry or feed to a Web resource. The (simplified) structure of the element is as follows:

```
1.5
 atomLink = element atom:link {
     attribute href { atomUri },
     attribute rel { atomNCName | atomUri }?,
     attribute type { atomMediaType }?,
     attribute hreflang { atomLanguageTag }?,
     attribute title { text }?,
     attribute length { text }?}
```

The `@href` attribute must 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[3], or an IRI for custom link relations.

2.2 Google Data Protocol

The Atom Publishing Protocol[4] is an application-level protocol for publishing and editing Web resources. The Google Data Protocol[3] extends the Atom Publishing Protocol for processing queries, authentication, and batch requests. It is based on HTTP transfer of Atom-formatted representations. There are two serializations available: XML and JSON[4]. The structure of the XML serialization is the same as in 2.1, the structure of the JSON serialization can be seen below:

```
1.5
 "link": [{
   "rel": "...",
   "type": "...",
   "href": "..."}]
```

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

2.3 Form Technologies

A regular human-readable (X)HTML page can define hypermedia controls. Contained links and potential surrounding textual information can be understood by humans, while machines can process 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 defined.

We adopt the term "form technologies" from Leonard Richardson[5] to reference a subset of description languages and mechanisms commonly criticized as being brittle[6]. The goal of form technologies like the Web Application Description Language (WADL)[7] (RESTful only when being used at runtime, not at design-time), XForms[8], or RDF Forms[9] is to describe the HTTP methods, parameters, and allowed values during an API request. A well-adopted form technology is OpenSearch[10].

2.4 Media Types

Media type give detailed insights into *how* to process a representation. They outline which parts of the representation are hypermedia controls and their meaning. 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 media type, a registration template can be submitted for review to the IANA. More specific, i.e., exactly not generic media types like `application/json`, `application/xml` or `text/xml`) are an essential aspect of achieving self descriptiveness and are key for implementing truely RESTful APIs.

2.5 Link Headers

In the proposed standard document Web Linking[7] 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 (section 6.2.2), therewith updating the relations defined in the Atom Syndication Format. Quoting from [7]:

> The Link entity-header field provides a means for serialising one or more links in HTTP headers. It is semantically equivalent to the <LINK> element in HTML, as well as the atom:link feed-level element in Atom [RFC4287].

A concrete example is shown below:

```
1.5
 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

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

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

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

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

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

[8]`http://www.w3.org/TR/xforms/`

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

[10]`http://www.opensearch.org/`

`previous` and `next` refer to an ordered series of resources. 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.

Hence, the idea is to combine link relations with meaningful Linked Data URIs.

3. HTTP OPTIONS

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.

OPTIONS is not supported by all servers. The expected behavior can be seen for example on our university domain `http://www.upc.edu/`, including out-of-HTTP extensions:

```
1.5
 $ 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
```

4. THE HTTP VOCABULARY IN RDF

The HTTP Vocabulary in RDF[6] defines a representation of HTTP in Resource Description Framework (RDF)[5]. It is intended to record HTTP(S) request and response messages, including the various headers. Consider the following HTTP request:

```
1.5
 $ curl -i http://dbpedia.org/
     -H "Accept: application/rdf+xml"
```

Modeled in RDF (in Turtle syntax[11], prefixes omitted for the sake of brevity) the request looks like this:

```
1.5
 _:req a http:Request ;
   http:httpVersion "1.1" ;
   http:methodName "GET" ;
   http:mthd <http://www.w3.org/2008/http-
       methods#GET> ;
   http:headers (
     [ 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> ]
       ) .
```

5. COMBINING THE TECHNOLOGIES

Bringing the technologies mentioned above together, we have first Link headers to transparently inject data into a response without touching the body of the HTTP message. Second, we have 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 how to fulfill the hypermedia constraint with these technologies, we took an existing Web application of ours[12], 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 query, one for Named Entity Extraction (NEE) based on a text fragment, and one for URI Lookup (UL) based on a term. See Figure 1 for an overview of the structure of the services and the link relations between the service components. We serve the output of the 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 API is the output format: the video annotation service must return RDF in one of its serializations, in consequence the media types are `text/turtle`, or `application/rdf+xml`. The RDF/XML media type[13] describes the format, however, no media controls. With our API it is, however, also possible to modify an existing automatically generated RDF annotation of a video in order to correct (via PUT) or delete (via DELETE) it. Hence, 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 upon accessing `http://localhost/youtube/rdf/3PuHGKnboNY`:

```
1.5
 Link: <http://localhost/youtube/videos/53PuHGKnboNY
     >;
```

[11] http://www.w3.org/TeamSubmission/turtle/
[12] http://tomayac.com/semwebvid/
[13] http://tools.ietf.org/html/rfc3870

```
        rel="related";
        type="application/prs.atom+json";
        title="video metadata"
    <http://localhost/youtube/search/opensearch.
        xml>;
        rel="search"; title="search",
    <>; rel="edit"; title="delete, modify",
    <>; rel="alternate"; type="application/rdf+xml
        ",
    <>; rel="alternate"; type="text/turtle"
```

Complimentary to Link headers, and due to the extensibility of RDF links can also be added in triple form to the actual service output (here and below we have only listed the first from all the Link headers above, prefixes omitted):

```
1.5
 _:link_1 a atom:link ;
   atom:href
     <http://localhost/youtube/videos/53PuHGKnboNY> ;
   atom:rel "related" .
```

In order to explore the API an OPTIONS call against, e.g., `http://localhost/youtube/videos` can be made that returns the following headers and body:

```
1.5
 $ curl -i -X OPTIONS http://localhost/youtube/videos

 HTTP/1.1 200 OK
 Content-Type: text/turtle; charset=utf-8
 Link: <http://localhost/youtube/videos/{video_id}>;
          rel="related"
 Allow: GET, HEAD, PUT, DELETE, OPTIONS, PATCH
 Content-Length: xx

 _:req_1 a http:Request ;
   http:httpVersion "1.1" ;
   http:methodName "GET" ;
   http:mthd <http://www.w3.org/2008/http-
       methods#GET> ;
   exHttp:prefixPath "/youtube/rdf/" ;
   exHttp:suffixPath [
     a api:uriTemplate ;
     a youtube_data_api_tag_yt:videoid ;
   ] ;
   http:headers (
     [ http:fieldName "Host";
       http:fieldValue "localhost";
       http:hdrName <http://www.w3.org/2008/-
           http-header#host> ]
     ) .
```

User agents can then try to discover the constraints (in this case that the video_id URI template is a YouTube video ID). `exHttp` stands for an exemplary extension of the HTTP Vocabulary in RDF by URL query path and fragment, which is not covered by the original vocabulary. The `exHttp:prefixPath` stands for the first part of the query path, `exHttp:suffixPath` for the second. Combined, the two span the entire query path. The `exHttp:suffixPath` is defined as an `api:uriTemplate`[14], which is described as being of type YouTube Data API video ID.

6. CONCLUSION

We have shown an approach towards fulfilling the hypermedia constraint with the OPTIONS message header and body, where the header contains Link headers, and the body HTTP Vocabulary in RDF triples. The approach has been successfully applied to an API with media type constraints. A criticism of the approach might be that it seems to encourage tight coupling, however, not if API consumers interpret their OPTIONS at run-time. Given the complete Semantic Web stack, automatic reasoning over such service annotations could be possible, and, in addition to that, semantic link relations open to REST the whole Linked Data world. Future work will be to evaluate our approach on more and complexer services, and see how far a consumer can reach with only discovering OPTIONS, following links, and interpreting the results.

[14] `http://code.google.com/p/linked-data-api/wiki/API_Vocabulary`

htb!

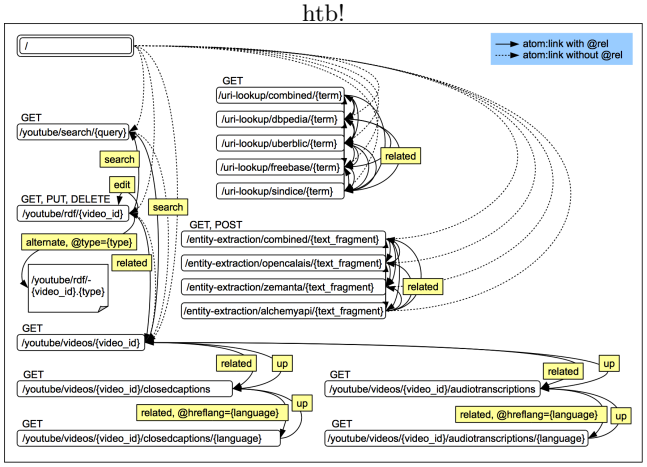

Figure 1: Overview of the API structure with link relations and allowed HTTP method names.

7. ACKNOWLEDGMENTS

Partially funded by EU FP7 I-SEARCH project (ref. 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] Google. Google Data Protocol, retrieved on February 10, 2011. `http://code.google.com/apis/gdata/`.

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

[5] 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/`.

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

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

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