Deutsche Nationalbibliothek
Adickesallee 1
60322 Frankfurt am Main

An http Header for Metadata Schema Negotiation

Lars G. Svensson, Deutsche Nationalbibliothek

1. Introduction

In many cases, there are several ways to describe a resource using a structured format such as XML or one of the RDF serialisations. In the case of XML documents, for instance, the same content can be encoded using one of several DTDs or XML Schemas, whereas in RDF there is a wide choice of RDF vocabularies (classes and properties) available to describe resources of the same type. E. g. do all three of foaf, the BBC Core Ontology and the DBPedia ontology contain classes and properties to describe persons. When a User Agent (UA) initiates a request, e. g. a GET request to retrieve or a PUT request to create or replace a resource, neither the UA nor the server have any possibility to exchange information on how the transmitted resource will be structured. One solution could be to define two new http headers: `Accept-Schema` and `Schema`.

2. Motivation

Since 2010, the German National Library (Deutsche Nationalbibliothek, DNB) operates a linked data service offering access to the bibliographic descriptions from the German National Bibliography (publication data) and to the descriptions of persons, places, things etc. from the Integrated Authority File (Gemeinsame Normdatie, GND)[1]. The data is available in HTML, MARC 21 and several RDF serialisations. In addition to this, the DNB runs a service named EntityFacts[2] that provides machine-readable "fact sheets" on entities in the GND. At first, the EntityFacts format was based on JSON, but in early 2016 this was changed to be JSON-LD, which effectively means that the DNB is running two services that both make GND data available as RDF but generate different sets of triples. Currently, that is not a problem since the two services use different base URIs (http://d-nb.info/ vs. http://hub.culturegraph.org/entityfacts/). The goal is, however, to have an entity-centric access to the data, serving all possible descriptions of a specific entity from one single URI thus making it easier for data consumers to reference those entities. In order to do that, however, we need a method to let the User Agent specify which of the descriptions it prefers.

Another case when a client and a server might need to agree on how syntactic and semantic format of the content is when a client issues a PUT or a POST request in order to create or update

[1] http://www.dnb.de/EN/gnd

[2] http://www.dnb.de/EN/Wir/Projekte/Abgeschlossen/entityFacts.html

a resource. In this case it is the server that need to specify if it could process the content and – if it could not – how the representation must be structured in order for the server to be able to process it correctly.

One way to allow this to happen is for client and server to implement Schema Negotiation.

3. Terminology and Implementation Options[3]

3.1. A Note on Terminology

In the context of this proposal, a "schema" is a document that expresses the syntactical and/or semantic constraints of other documents. Examples of "schema" in this context include, but are not limited to, Dublin Core application profiles – formally expressed in Dublin Core Description Set Profiles (DSP)[4] –, XML Schema documents and RDF Shapes expressed in SHACL. How those schemata are used by consuming applications is beyond the scope of this proposal, but typical use cases are validation of data received from another system and the automated creation of objects from received data as in Java XMLBeans. The choice of the term "schema"was derived from its use in XML Schema and from the information that RDF Shapes are often described as XML Schema for RDF (RDF Schema was already taken). Another option would have been "profile", as used in e. g. the Link header field.[5] None of the terms is right or wrong, we just need to agree on what term to use.

3.2. Other Implementation Options Considered

A number of options were considered when specifying how schema negotiation could be implemented. Besides the registration of an appropriate accept header, the options included the use of profiles with the http "Accept" or "Link" header fields as in RFC 6906.

1. According to RFC 6906, profiles are one possibility "to include additional information about the nature of the resource. This would allow a client understanding this additional information to react in a way specific to that specialization of the resource, where the specialization of the resource, where the specialization can be about constraints, conventions, extensions or any other aspects that do not alter the basic media type semantics." This is one possible implementation, but with the disadvantage that there is no way to specify a weight for the profile which limits the possibilities to perform proper content negotiation.
2. Another way to convey profile information is through the http Accept header field as in the following example:
   `Accept: application/rdf+xml; profile=<urn:example:profiles:e-commerce-payment>`
   If this is possible depends on the media type. Of the media types commonly used for linked data, only two registrations in the IANA Media Type Registry foresee the use of profiles:

---

[3] Most of the text in sections three and four is taken verbatim from a not-yet-submitted Internet Draft at https://github.com/larsgsvensson/I-D-Accept--Schema/ with some changes due to the feedback of the reviewers. Comments on the I-D are warmly welcome!

[4] http://dublincore.org/documents/dc-dsp/

[5] http://www.rfc-editor.org/info/rfc6906

`application/xhtml+xml` and `application/ld+json`. E. g. neither `application/rdf+xml` nor `text/turtle` mention the use of profiles.

A further option would be to use the http “Prefer” and “Preference-Applied” headers as specified in RFC 7240.[6] This approach has two disadvantages. The first is – as with the “Link” header, that there is no possibility to work with q-values. The second one is that the only way for a server to state that it ignored the preference stated by the client is to omit sending a “Preference-Applied” header. For the client – however – it is not clear if the “Preference-Applied” header is absent because the server did not honour the preference, or if it is because the server did not understand the “Prefer” header in the first place. This could be solved by making it mandatory to send a “Link: rel=profile” header when answering to a request with a “Prefer: profile=”” header in it. This solution requires that a client evaluates two different headers in order to find a response to its request for a specific schema, which would make client implementation more complicated.

For both options, it is also not clear if it is possible to specify that an XML document must comply with several XML schemas, where the schemas are bound to individual XML namespaces used. For the those reasons, both options were discarded and left the registration of a new http header as the most viable way forward.

4. Proposed Behaviour

The "Accept-Schema" and "Schema" header fields can be sent by both the UA and the server. The "Accept-Schema" header is used to specify one or more schemas the Agent can accept, whereas the "Schema" header tells the other Agent according to which schema the payload of the message is structured. This way a UA issuing a request for a resource can specify that it prefers persons to be described using foaf, but that the BBC Core ontology is also acceptable, and that it can only accept text/turtle, by setting the "Accept" and "Accept-Schema" header fields appropriately. When the server answers, it would set the "Content-Type" and "Schema" header fields. Likewise, a UA sending an XML document to a server would set the "Content-Type" and the "Schema" header fields. If the server cannot process the specified schema, it would answer with an http 406 status code and possibly a list of acceptable schemas.

An "Accept-Schema" and "Schema" header field does not contain the actual schema but instead points to it using a URI. As long as the URI is only used to denote the schema, the URI does not need to point to an actual document but can be considered opaque. If the parties involved agree on a schema definition, the schema can be identified with e. g. a URN or an info-URI. When a protocol-based URI, such as an FTP- or an HTTP-URI is used, however, it is RECOMMENDED that it dereference to a document containing the schema definition.

5. Examples

The following examples highlight the exchange of schema information between a client and a server. For clarity, the examples only contain minimal information, i. e. only the relevant headers are included and message bodies are ignored.

5.1. Example 1

[6] http://www.rfc-editor.org/info/rfc7240

A client requests an XML document conforming to a specific XML schema. The schema is identified by <urn:example:schema:e-commerce-payment>.

Request:

```
GET /some-resource HTTP/1.1
Accept: application/xml
Accept-Schema: <urn:example:schema:e-commerce-payment>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Schema: <urn:example:schema:e-commerce-payment>
```

5.2. Example 2

A client requests an RDF/XML document conforming to one of two RDF Shapes (<http://example.com/shapes/shape-1> and <http://example.com/shapes/shape-2>). It uses q-values to express a preference for shape-1, the server, however, prefers to deliver in shape-2.

Request:

```
GET /some-resource HTTP/1.1
Accept: application/rdf+xml
Accept-Schema: <http://example.com/shapes/shape-1>; q=0.8,
       <http://example.com/shapes/shape-2>; q=0.5
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/rdf+xml
Schema: <http://example.com/shapes/shape-2>
```

5.3. Example 3

A client PUTs a turtle document conforming to the RDF Shape <http://example.com/shapes/shape-1>. The server answers that it can only process documents conforming to <http://example.com/shapes/shape-2>.

Request:

```
PUT /some-resource HTTP/1.1
Schema: <http://example.com/shapes/shape-1>
```

Response:

```
HTTP/1.1 406 Not acceptable
Content-Type: application/xhtml+xml
Accept-Schema: <http://example.com/shapes/shape-2>
```

5.4. Example 4

A client requests an XML document where the elements in namespace <urn:example:namespaces:ns1> must conform to XML schema <http://example.com/schema/schema-1> and the elements in namespace <urn:example:namespaces:ns2> must conform to XML schema <http://example.com/schema/schema-2>. The server answers that it can supply the document as requested.

Request:

```
GET /some-resource HTTP/1.1
Accept-Schema:  <urn:example:namespaces:ns1 http://example.com/schema/schema-1
      urn:example:namespaces:ns2 http://example.com/schema/schema-2>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Schema: <urn:example:namespaces:ns1 http://example.com/schema/schema-1
      urn:example:namespaces:ns2 http://example.com/schema/schema-2>
```

6. Conclusion and Next Steps

It is not enough for publishers and consumers to agree on metadata schemata, they also need a way to formally specify those and to provide machine-understandable information about which schema to use, particularly when the data is not only exchanged within closed communities.

While there are technologies in place to describe metadata schemas (e. g. SHACL, DSP and XML-Schema), there is no agreed-on technology that allows clients and servers to exchange information about which schema to use. This paper – and the cited I-D – propose a solution to solve this.

Depending on the outcome of the workshop and the feedback given, the I-D might be submitted to IETF in early 2017.