Creating 3rd Generation Web APIs with Hydra

Markus Lanthaler

Institute for Information Systems and Computer Media
Graz University of Technology
Graz, Austria

mail@markus-lanthaler.com

ABSTRACT

In this paper we describe a novel approach to build hypermedia-driven Web APIs based on Linked Data technologies such as JSON-LD. We also present the result of implementing a first prototype featuring both a RESTful Web API and a generic API client. To the best of our knowledge, no comparable integrated system to develop Linked Data-based APIs exists.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *Linked Data, Web 2.0, World Wide Web (WWW).*

Keywords

Web; Web services; REST; Linked Data; JSON-LD; Hydra

1. INTRODUCTION

Hyperlinks form the foundation of the World Wide Web. While developers use them intuitively when building traditional Web sites to guide visitors through their sites, they are often completely missing from Web APIs. This results in tightly coupled and thus brittle systems which cannot be evolved over time. Just as in first generation SOAP-based Web Services all the possible interactions are typically hardcoded into the clients instead of being communicating the legal state transitions at runtime. Even though such approaches might work in the short term, they are condemned to break in the long term as assumptions about server resources will break as resources evolve over time.

Second generation Web APIs go a step further and do use hypermedia. Unfortunately most of these APIs rely on out-of-band contracts to represent links. The reason is that they are build, almost without exception, on formats that have no inherent support for hypermedia; mostly XML and JSON. Clients thus rely on specific structures to recognize links as such and the result is almost the same as in first generation Web APIs: tightly coupled systems that easily break.

In this paper we present a technology stack to create third generation Web APIs that do not suffer from the issues their ancestors expose. This allows the creation of truly RESTful services with all its benefits in terms of scalability, maintainability, and evolvability. Furthermore we will demonstrate how such an approach can be used to create generic API consoles and clients.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

2. USING JSON-LD AND HYDRA TO BUILD TRULY RESTFUL APIS

To build hypermedia-driven Web APIs the used serialization has to have built-in support for hyperlinks. XML can be extended by using the XML Linking Language (XLink) [1] to achieve that, but in JSON, the prevalent format used in current Web APIs, no similar, accepted extension exists. JSON-LD [2] is a format that addresses this issue and, as we have shown in previous work [3], is well-suited for RESTful services.

JSON-LD has been designed to provide a simple and convenient serialization format for Linked Data based on JSON. Instead of the triple-centric approach that other common serialization formats for Linked Data use, an entity-centric approach was chosen. The rationale was to resemble the programming models most developers are familiar with and to reflect the way JSON is used. This and the fact that JSON-LD is 100% compatible with traditional JSON allow developers to build on existing infrastructure investments.

While JSON-LD represents a generic serialization format, also a shared vocabulary, understood by both the server exposing the API and the client consuming it, is needed to implement a concrete Web API. Hydra ([4], [5]) is an attempt to define a minimal vocabulary to achieve just that. It defines a number of concepts commonly used in Web APIs and provides a vocabulary to describe the domain application protocol of an API. Operations build the core of the vocabulary as they allow the description of the functionality provided by the API. Simply speaking, operations map high-level business functionality to low-level HTTP interactions. To be discoverable, operations can either be bound to specific entity classes or properties thereof or be used directly in representations.

In most cases, the responses returned by an API based on Hydra and JSON-LD are almost indistinguishable from traditional JSON-based services. Listing 1 shows the representation of an issue. Apart from `@context`, the few JSON-LD specific

```
{
  "@context": "/demo/ctx/Issue.jsonld",
  "@id": "/demo/issues/1",
  "@type": "Issue",
  "title": "WWW2013 Paper",
  "description": "Write paper for WWW2013",
  "is_open": true,
  "raised_by": "/demo/users/1",
  "created_at": "2012-11-26T04:49:44Z",
  "comments": "/demo/issues/1/comments/"
}
```

Listing 1. An exemplary API response

```
{
  "@id": "/demo/vocab#Issue/comments",
  "@type": "hydra:Link",
  "label": "Comments",
  "domain": "/demo/vocab#Issue",
  "range": "hydra:Collection",
  "operations": [
    {
      "@type": "/demo/vocab#CreateComment",
      "method": "POST",
      "label": "Creates a new comment",
      "expects": "/demo/vocab#Comment",
      "returns": "/demo/vocab#Comment"
    }
  ]
}
```

Listing 2. Definition of the `comments` property (excerpt)

keywords could be aliased to any desired string. The context specifies how values in such a representation can be interpreted. While `created_at` would be typed as a date/time string in this example, `comments` would be marked as an IRI. Furthermore, the context contains the mappings necessary to expand properties to IRIs which makes it possible to retrieve more information.

Listing 2 shows the definition of the `comments` property. It defines the property as a Hydra `Link` which means that its values represent dereferenceable resources a client can interact with. The range declaration further refines it by saying that the resources will be instances of Hydra's `Collection` class. The interesting part, from the point of view of a client, is the `operations` property. In this example, a single operation to create a new comment is associated with the property. The operation declares the expected data and the data returned on success. This allows a client to construct a valid HTTP request if all required information is available or to render a form to request the necessary data from a user.

While this example uses proprietary properties and operations, it would work exactly the same way if definitions from another API or a centrally defined standard would be used instead. This unique feature paves the way for a completely new breed of interoperable Web APIs using decentralized, reusable, and composable contracts. It allows developers to create clients that work with various different APIs instead of having to create a specialized client for each of them. It also simplifies standardization since complex problems can easily be divided into smaller sub-problems. Concrete implementations can then choose from a variety of options and combine them to a new application. Given that a lot of overlapping functionality exists in Web APIs, not only within but also across verticals, we are convinced that this is a crucial feature that every proposed solution should address.

3. CREATING GENERIC CLIENTS

The combination of JSON-LD and Hydra enables the creation of machine-processable contracts that can be discovered at runtime. This allows the implementation of completely generic clients, such as API consoles or client libraries. To demonstrate the feasibility of the principles and technologies presented in this paper, we built a simple Web API featuring an issue tracker and described it using Hydra. This illustrates how easily the proposed approach can be integrated in real-world systems. Furthermore, we developed two generic clients to access Hydra-powered Web APIs. One client represents an API console allowing users to interact with the API in a similar fashion they interact with normal websites. The other client, which, due to space constraints is not presented in this paper, is a small PHP library that can be used for programmatic access. All components are open source and available on Hydra's homepage [4].

The server component is based on Symfony2, a Web development framework implemented in PHP. We extended Symfony2 by a implementing a custom bundle, i.e., a plugin, which serializes entities into JSON-LD representations. Furthermore, the bundle generates machine-readable documentation using Hydra to describe the entity types and their properties as well as the affordances the system supports. The bundle relies on code annotations to control the serialization of entities and the documentation of their types. While this is more complex than simply serializing and documenting all public members of an entity, it provides much more flexibility. Compared to other configuration mechanisms, annotations have the advantage that the information is kept close to the source code it documents, which makes it much easier to keep the two in sync. Once the entities have been annotated, the code for the controllers implementing the basic CRUD functionality can be generated can completely automatically.

Listing 3 shows how the PHP class representing an issue can be annotated. Elements that are exposed by the API are marked with the `@Hydra\Expose()` annotation. Without parameters, the system automatically generates an IRI for such elements as shown in Listing 2. To allow reuse, it is also possible to map an element to an existing IRI using the same mechanism. These annotations are mainly used by the serializer to create representations and contexts. The `@Hydra\Operations` annotation on the other hand, is used to document the valid HTTP operations for an element. The strings used in the example identify Symfony2 routes which

```
/**
 * An Issue tracked by the system.
 *
 * @Hydra\Expose()
 * @Hydra\Operations( {
 *   "issue_replace",
 *   "issue_delete"
 * } )
 */
class Issue
{
  /**
   * The comments associated with this issue
   *
   * @Hydra\Expose()
   * @Hydra\Collection("issue_comments")
   * @Hydra\Operations("issue_comment_create")
   */
  private $comments;

  // ... other members and methods ...
}
```

Listing 3. The `Issue` class (excerpt)

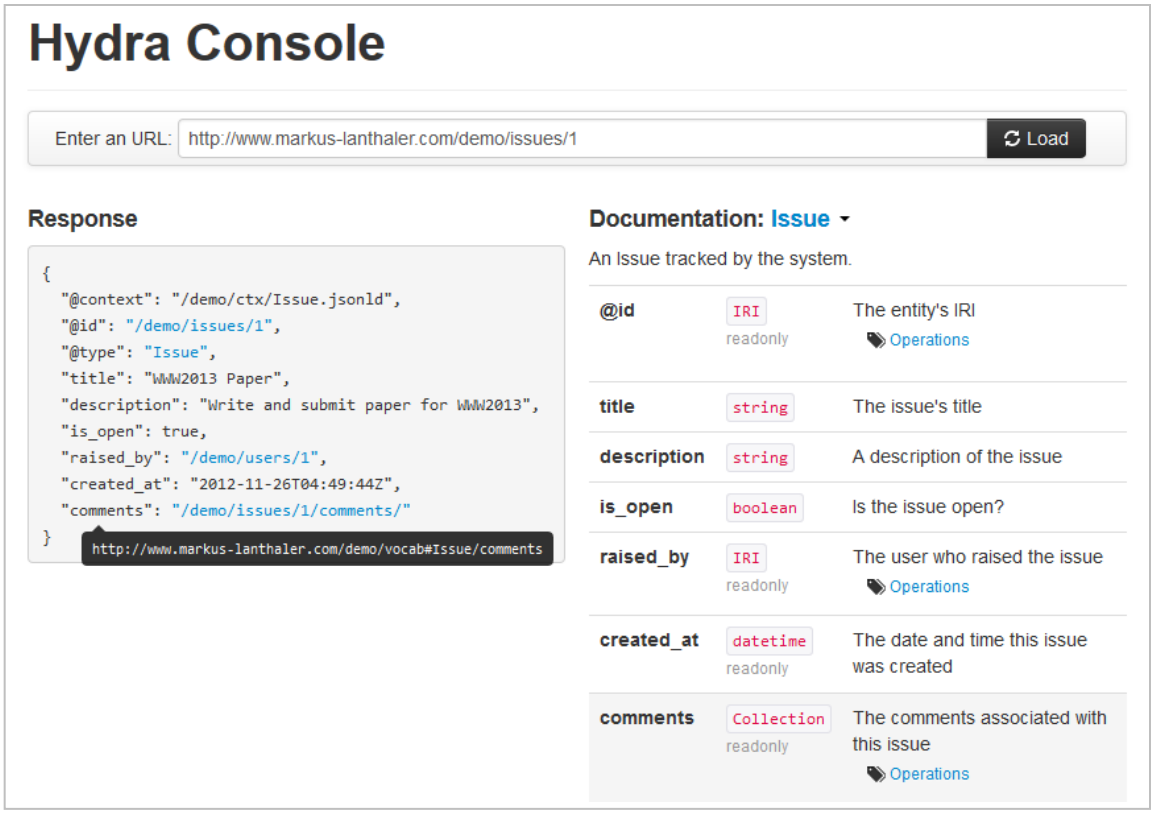

Figure 1. The API console showing a representation of an issue alongside its documentation

correspond to specific controller methods. These methods specify what data an operation expects, what data and status codes it might return, and which HTTP method has to be used. This data is used to create descriptions similar to the one shown in Listing 2.

The API console we developed as proof of concept shows the feasibility of generic clients to access APIs powered by Hydra. It is implemented as a single-page Web application using a number of well-known libraries such as Bootstrap, Backbone, Underscore, and a slightly modified JSON-LD processor. The JSON-LD processor had to be modified to include additional information in the parsed responses required by the response renderer for tooltips etc.; otherwise a standard JSON-LD processor implementing the JSON-LD API [6] could have been used as well. The functionality of the client includes the retrieval of resource representations, their parsing and rendering (which includes displaying the related documentation), as well as the invocation of various HTTP operations on embedded hyperlinks, which, in some cases, implicates the dynamic creation of forms to gather the required data to construct valid requests.

4. CONCLUSIONS AND FUTURE WORK

In this work we presented an approach to model and describe RESTful APIs using Linked Data technologies such as JSON-LD, RDF, and Hydra. We have shown that it is easy and practical to integrate such an approach in current Web frameworks. In future work we would like to refine Hydra and to implement various tools to simplify developers' lives. We would also like to experiment with autonomous agents accessing APIs documented in such a way.

5. REFERENCES

[1] S. DeRose, E. Maler, D. Orchard, and N. Walsh, "XML Linking Language (XLink) Version 1.1," *W3C Recommendation*, 2010. [Online]. Available: http://www.w3.org/TR/xlink11/.

[2] M. Sporny, G. Kellogg, and M. Lanthaler, "JSON-LD 1.0 - A JSON-based Serialization for Linked Data," *W3C Working Draft*, 2013. [Online]. Available: http://www.w3.org/TR/json-ld /.

[3] M. Lanthaler and C. Gütl, "On Using JSON-LD to Create Evolvable RESTful Services," in *Proceedings of the 3rd International Workshop on RESTful Design (WS-REST) at the 21st International World Wide Web Conference (WWW2012)*, 2012, pp. 25-32.

[4] M. Lanthaler, "Hydra Core Vocabulary Specification," 2013 (work in progress). [Online]. Available: http://www.markus-lanthaler.com/hydra/.

[5] M. Lanthaler and C. Gütl, "Hydra: A Vocabulary for Hypermedia-Driven Web APIs," in *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW2013) at the 22nd International World Wide Web Conference (WWW2013)*, 2013.

[6] M. Lanthaler, G. Kellogg, and M. Sporny, "JSON-LD 1.0 Processing Algorithms and API," *W3C Working Draft*, 2013. [Online]. Available: http://www.w3.org/TR/json-ld-api/.