A Framework for Semantic Description of RESTful Web APIs

Ivan Salvadori
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, SC - Brazil
ivan.salvadori@posgrad.ufsc.br

Frank Siqueira
Department of Informatics and Statistics
Federal University of Santa Catarina
Florianópolis, SC - Brazil
frank@inf.ufsc.br

***Abstract*—Support from development tools and infrastructure frameworks is crucial to increase the development of Web APIs that follow the REST architectural principle, leaving the software developer free to focus on the implementation of the business core of the application. This paper introduces a framework for semantic description of RESTful Web APIs, which is based on annotations added to the application code that associate resources, properties and operations with terms semantically described by vocabularies and ontologies. The proposed framework enforces the adoption of design principles for modeling Web APIs, focusing on resource representations and targeting important features for high concurrency services, such as low coupling and high flexibility among layers.**

Keywords-REST; Web API; Semantic Web Services; Hypermedia Control; Linked Data.

I. Introduction

The availability and use of RESTful Web APIs are in continued expansion, putting the principles that drive their implementation into focus [1]. Frameworks that provide support for building RESTful APIs are highly necessary in order to improve development productivity and software quality. Representations with support for hypermedia controls [2] [3], relationships between resources through hyperlinks, support for collections [4] and semantic description [5] are some improvements towards the construction of well-structured REST APIs.

Adamczyk [6] compares the level of support for REST principles of the 10 most used frameworks for Web API development. Just a few provide some level of support for hyperlinks. Most of them only provide the basic REST functionality, i.e., handle HTTP requests, return responses according to protocol semantics, negotiate content type using the most common formats, expose classes as REST resources and the corresponding mappings as URI templates.

The framework introduced in this paper proposes a set of annotations that allow the semantic description of resources and the generation of representations containing hypermedia controls. Resource collections should enable access to their items through hyperlinks [6]. Therefore, the proposed framework provides means to expose each resource of the collection through a URI, allowing the client to obtain information on each resource individually. Another important functionality made available by the framework is the semantic enrichment of properties and operations of resources through the creation of a binding between shared concepts and meanings defined by ontologies and vocabularies.

The proposed framework enables the development of Web APIs that are more strictly aligned with REST principles. Resource representations may contain not only data, but also hypermedia controls that can be applied to the current resource state. Web APIs with this feature are called RESTful Hypermedia Driven, which is a fundamental factor for developing clients that are more implementation-independent and resilient to server-side changes. Furthermore, this kind of Web API simplifies the construction and integration of autonomous clients.

This paper is organized as follows: Section II presents the main principles of REST and the standard API for developing RESTful services in Java. The available technologies for hypermedia support are discussed in section III. Section IV describes the proposed framework in high level of detail. An illustrative example of application developed using the framework is given in Section V. Section VI describes related research efforts found in the literature and compares them with the proposed framework. Finally, the conclusions and plans for future work in this research field are presented in Section VII.

II. Background

This section introduces the main concepts required to understand the framework described in this paper.

A. REST

REST (Representational State Transfer) was proposed by Fielding [1] and consists in a series of architectural principles and constraints for developing distributed applications on the Web. The "resource" is the central concept in the REST architectural principle, in which it represents a piece of information that is identified and addressed through the HTTP protocol. Every single resource has at least one representation, which is a data set associated with a resource and represented using a given format, with XML and JSON being the most frequently adopted.

HATEOAS (Hypermedia as the Engine of Application State) is one of the main principles behind REST [1], which aims to associate data elements with a set of possible actions (hypermedia controls). One example of hypermedia control is a resource representation with a hyperlink to another resource,

978-1-4799-5054-6/14 $31.00 © 2014 IEEE
DOI 10.1109/ICWS.2014.93

allowing the application consumer to browse from one resource to another.

B. JAX-RS Specification

The JAX-RS (Java API for RESTful Services) specification defines a group of APIs for developing Web Services that follow the REST architectural principles [7]. These APIs provide a set of annotations that allows regular Java classes (POJOs - Plain Old Java Objects) to be exposed as resources. HTTP methods (such as GET, POST and so on) can be associated with Java methods, and a variety of content types can be used to represent input (consumed) and output (produced) data. Fig. 1 presents an example of Java class with JAX-RS annotations.

```
@Path("webResourceSample")
public class WebResourceSample {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response handleHttpGet() {
        //do something
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response handleHttpPost(Obj obj) {
        //do something
    }

    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response handleHttpPut(Obj obj) {
        //do something
    }

    @DELETE
    @Produces(MediaType.TEXT_PLAIN)
    public Response handleHttpDelete() {
        //do something
    }
}
```

Fig. 1. JAX-RS Sample

III. Hypermedia and RESTful Web APIs

Hypermedia is a strategy to allow the information consumer to control the navigation through a REST API. This navigation may assume the form of hyperlinks provided by a given resource, which lead the consumer to other pieces of information or provides ways to execute CRUD (Create, Read, Update and Delete) operations. To allow the specification of hypermedia information, the resource must be described using a data format that supports these semantic controls. Given that JSON is the most common data format for describing REST resources, the lack of support for hypermedia controls is a serious limitation. Fig. 2 shows an example of a JSON object in which the "photo" attribute, despite being able to assume any text value, has an URI as its value. This is called a "fake hypermedia control" [8], because the attribute does not have any semantic information associated with it. Linking shared semantic meaning with properties and operations of resources is able to increase the level of understanding and to improve the interoperability between the Web API and its clients. This allows information encapsulated by resources to be addressed, even if properties have different identifiers.

```
{
   "name":"Lucky Luke",
   "photo":"http://api.com/lucky.jpg"
}
```

Fig. 2. Fake Hypermedia Control

A. JSON-LD

JSON-LD [9] is a data representation format based on JSON, which provides support for linked data and hypermedia controls. JSON-LD is seen as the first step to standardize the semantics of RESTful Web APIs [2]. A valid JSON-LD document is also a valid JSON document, allowing the use of the existing tools for handling JSON.

Fig. 3 shows an example of HTTP response message that has information together with hypermedia control in JSON-LD format. The first relevant information is the Content-Type, located in the header of the HTTP message, which says that the object contained in the payload is serialized in JSON-LD format instead of plain JSON. The @context property of the object contained in the payload of the HTTP message adds semantic meaning to other document properties, binding them to elements described by ontologies or vocabularies. The semantic meaning of these elements can be understood by checking the ontology or vocabulary, which may be accessed through the corresponding URI and making use of Linked Data [5]. The attribute @type with value @id adds semantic meaning of hyperlink to the "photo" property of the contained object, distinguishing the value of this property from a plain text field.

```
HTTP/1.1 200 OK
Content-Type: application/ld+json
--------------------------------
{
   "context":{
      "name":"http://schema.org/name",
      "photo":{
         "@id":"http://schema.org/image",
         "@type":"@id",
      }
   },
   "name":"Lucky Luke",
   "photo":"http://api.com/lucky.jpg"
}
```

Fig. 3. JSON-LD HTTP Response

B. HYDRA

In spite of improving the support for hypermedia controls, JSON-LD only allows the addition of hyperlinks to attribute values. It does not provide a mechanism to specify operations that modify resource state [8]. Hydra [10] improves the support for hypermedia controls by widening the applicability of JSON-LD mechanisms through the addition of a vocabulary for hypermedia control. Fig. 4 shows a simplified example of resource described using Hydra, which has two operations.

Hydra enables the development of generic RESTful Web APIs that can be interpreted by generic and autonomous clients. Integration is allowed by building a documentation that describes data representation with the corresponding attributes and operations, which are linked to a shared vocabulary that

```
{
   "context":{...},
   "id":"/fasterThanHisShadow",
   "name":"Lucky Luke",
   "photo":"http://api.com/lucky.jpg",
   "operations":[
      {
         "@type":"ReplaceResourceOperation",
         "method":"PUT"
      },
      {
         "@type":"DeleteResourceOperation",
         "method":"DELETE"
      }
   ]
}
```

Fig. 4. Hydra Sample

defines their respective semantic meaning. Resource collections may also be represented by a list of hyperlinks to the contained resources [4], providing a very useful feature for Web API clients.

IV. Hypermedia Web API Support Framework

The Hypermedia Web API Support Framework, which is based on the JAX-RS 2.0 specification, aims to provide support for developing RESTful Web APIs with hypermedia controls and linked data. The framework allows regular Java classes to be enriched with semantic information, which is linked to the existing attributes and operations and is employed to automatically generate human-readable documentation for the Web API.

A. Framework Structure

The framework adopts the principle of Web APIs based on representations instead of resources. The resource-based design results in exposing domain objects as REST resources, without interfering in the way they are modeled. Despite the fact that Lanthaler [11] proposes to expose the domain layer of the application, the representation-based approach results in a more loosely coupling between the business core and the data integration layer. Fig. 5 illustrates the representation layer, which performs the role of the data integration layer through a REST API.

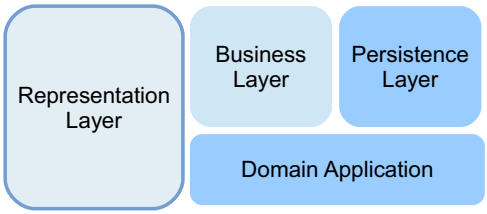

Fig. 5. Framework Structure

The framework represents resources using JSON-LD media type with Hydra extensions. The former allows the addition of semantic information to properties of a representation through relationships with elements described by vocabularies and ontologies, while the latter expands the semantic description towards operations supported by the corresponding representation.

Fig. 6 presents the JSON-based technologies employed by the proposed framework. One result obtained with the adoption of the framework is the automatic generation of the Web API profile. This profile explains the meaning of all properties and operations supported by a given representation. Based on the annotations provided by the framework, which are placed on classes of the Web API to describe the semantic meaning of properties and operations of REST representations, the framework generates a single profile document with the description of the whole API in JSON-LD format.

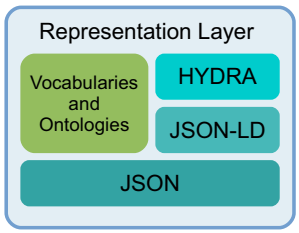

Fig. 6. Representation Layer Elements

B. Annotations

The framework has a set of annotations that are inserted in different parts of the code of a resource representation to add semantic information, as shown in Fig. 7. Annotations can be added to classes, attributes or methods. The main annotations that can be added to classes are:

- @SemanticClass: associates the class with a REST representation;
- @SemanticCollection: associates the class with a collection of REST resources;
- @Vocabulary and @Vocabularies: specify one or more vocabularies or ontologies that describe elements of the REST representation;
- @EntryPoint: defines the initial access point of the Web API.

Two other annotations - @SemanticProperty and @SemanticHyperlink - are applied on attributes in order to bind properties of the REST representation with vocabulary items associated with the respective class. The main difference between these two annotations is the way the property value is serialized. @SemanticProperty indicates that the property value is directly contained in the representation, while @SemanticHyperlink implies that the property value is a hyperlink to the property value.

There is still a group of annotations that are applied directly on methods of the representation class. They allow operations supported by the representation to be semantically described through the Hydra vocabulary. These annotations - @RetrieveResourceOperation, @CreateResourceOperation, @ReplaceResourceOperation and @DeleteResourceOperation - indicate the operations responsible for reading, creating, replacing and removing resources, respectively. These annotations are required because HTTP verbs are not able to describe the semantics of complex business requests, which have more

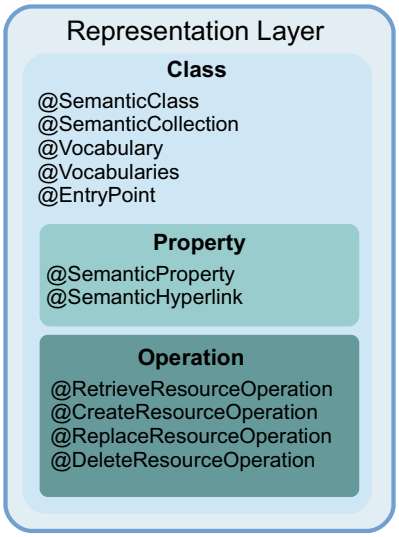

Fig. 7. Annotations

elaborate meanings. Annotations can also be extended in order to specify in more detail the semantic meaning of business operations.

C. Framework Design

The framework enforces a clear division between classes belonging to the application domain and classes that are part of the representation layer. Classes from the application domain are seen as resources, which are exposed by the RESTful Web API through their association with representation classes. As shown by Fig. 8, the same resource may have more than one representation, providing support for different data formats that can be negotiated between the server and its clients [6]. Resources may also have different representations to allow representing information with different levels of granularity. A fine grained granularity level results in lower data coupling and higher parallelism. Despite the fact that resources with different representation will more regularly differ only in data format, the framework also encourages the construction of representations with different structure, i.e., one resource may have different representations, and each may have a different subset of its state.

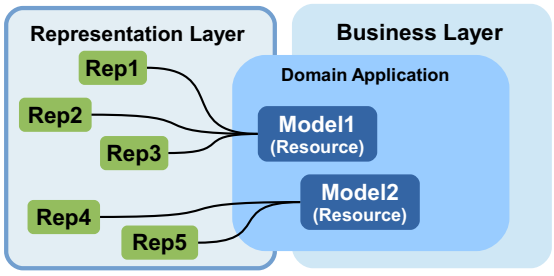

Fig. 8. Representations and Resources

The example shown in Fig. 9 presents a resource with three different representations. The associations between a resource and its multiple representations allow each single representation to have a different subset of resource state and/or to represent the resource using a different data format. Each Java class annotated with *@SemanticClass* is a representation of a resource. By separating resources from their representations, the framework allows exposing resources with different views and formats. It is possible to manipulate smaller portions of information using a fine-grained representation of the resource and, consequently, simplify data manipulation in high concurrency systems.

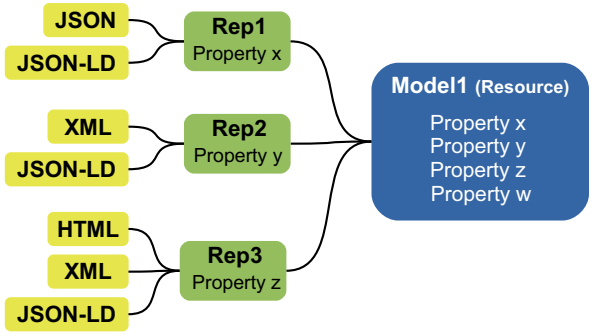

Fig. 9. Data Types and Representation

V. USE CASE: FBI WANTED LIST

The FBI Wanted List RESTful Web API was developed aiming to show how the proposed framework can be employed in a real-world application scenario. All required implementation steps for developing this use case will be detailed along this section. The use case is a hypothetical implementation for a RESTful Web API that provides information concerning people wanted by the FBI [1]. Information on wanted people is made available on the FBI Web site. The web pages were the source for a structural analysis of information, in which REST representations of resources were identified.

A. Design

Richardson [8] recommends some steps for developing WEB APIs, and the first of them is identifying the elements representing resource information. Each identified representation is mapped to a distinct class, which together compose the Representation Layer (Fig. 10). The WantedList class is a collection of hyperlinks that points out to several instances of the Wanted class, which describes a wanted person. This class has several properties such as a list of photos, aliases, a list of committed crimes, scars and body marks, and both detailed and summarized descriptions. These properties are mapped to different REST representations, which were modeled aiming to provide information with different levels of granularity. In a resource represented with fine granularity, most properties are represented through hyperlinks, and the linked content must be retrieved to obtain the actual value of the property.

[1] http://www.fbi.gov/wanted

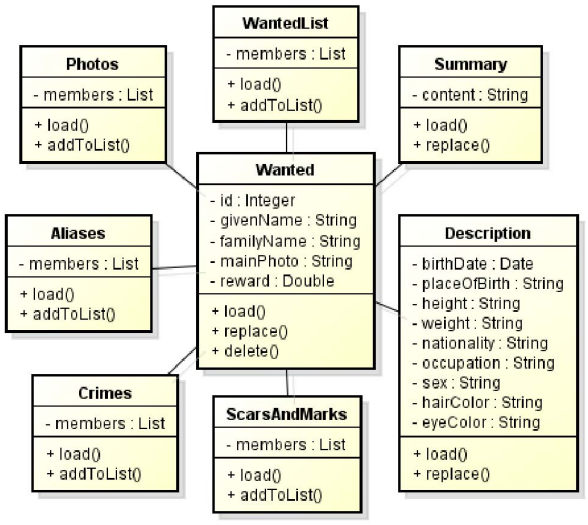

Fig. 10. Class Driagram

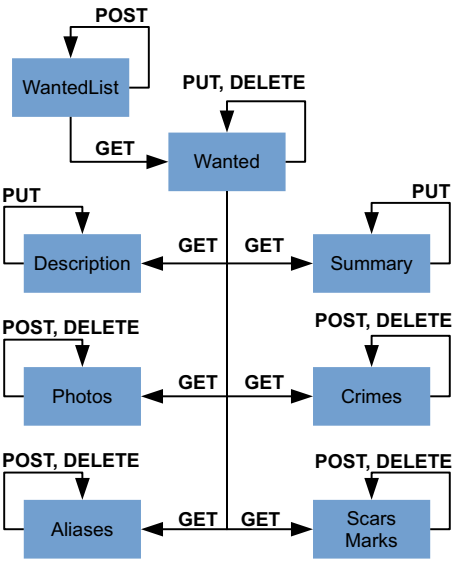

Fig. 11. Flowchart Diagram

B. Relationship between Representations

According to Richardson [8], the next step is to identify relationships among representations. One option is to draw a flowchart [12] [13], in which representations assume the role of steps, whereas hyperlinks and operations correspond to transitions. Fig.11 shows the flowchart corresponding to the FBI Wanted List Web API. The initial step corresponds to the WantedList representation, which accepts HTTP POST requests as the operation to add new elements to the collection, and HTTP GET requests to obtain information on a given wanted person. The Wanted step represents the elements stored by the collection. From the Wanted step, authenticated users can execute replace or delete operations by means of HTTP PUT and DELETE requests, respectively. The description and summary can be modified with HTTP PUT requests to the corresponding URI. The remaining properties - i.e., Photos, Crimes, Aliases and ScarsAndMarks - are obtained through HTTP GET requests, which return collections of resources that accept creation and removal operations of elements with HTTP POST and DELETE requests, respectively.

C. Implementation

The framework offers a set of annotations that semantically enrich the representations, its properties and operations. The WantedList (Fig. 12) class has 4 annotations: @Path is a JAX-RS annotation that exposes the class as a REST resource; @EntryPoint means that this representation is the starting point of the Web API; @SemanticCollection specializes the representation as a collection of resources whose semantic meaning is specified by the URI contained in the "id" attribute; and @Vocabulary defines the URI of the vocabulary that semantically describes this collection and its members. Other annotations are added to the attributes and methods of this class, such as @CollectionMembers, which defines that the annotated attribute represent a list of collection items; @RetrieveResourceOperation, which specifies the operation responsible for obtaining the representation data; and @CreateResourceOperation, which is responsible for resource creation. It is also possible to specify the type of input and output parameters of methods in the corresponding annotations. Fig.13 presents the document resulting from the serialization of the WantedList instance.

```
@Path("wantedList")
@EntryPoint
@SemanticCollection(id = "vocab:wantedList")
@Vocabulary(prefix="vocab", url="http://api.com/doc/")
public class WantedList {

    @CollectionMembers(id="vocab:wantedListMembers")
    private final List<Wanted> members;

    @GET
    @RetrieveResourceOperation
    public Response loadList() {
        //code to load
    }

    @POST
    @CreateResourceOperation(
      expects = Wanted.class, returns = Wanted.class)
    public Response addToList(Wanted wanted) {
        //code to add a new item to list
    }
}
```

Fig. 12. WantedList.java

Another example that shows how annotations provided by the framework are employed is given in Fig. 14. The @SemanticClass annotation added to the Wanted class indicates that this class is a representation of the semantic concept "dbpedia-owl:Criminal" specified by the "id" attribute. The developer can use several vocabularies by means of the @Vocabularies annotation. Properties whose value is directly specified in the representation receive the @SemanticProperty annotation and have their sematic meaning described by the "id" attribute. Properties in which the corresponding value has to be obtained through a hyperlink have the @SemanticHyperlink annotation. Their semantic meaning is described either by the "id" attribute, if present, or by the corresponding class. Some of

```
{
   "@context":"http://api.com/context.jsonld",
   "@id":"http://api.com/doc/wantedList",
   "@type":"Collection",
   "members":[
      {
         "@id":"/wanted/1",
         "@type":"http://dbpedia.org/ontology/Criminal"
      },
      {
         "@id":"/wanted/2",
         "@type":"http://dbpedia.org/ontology/Criminal"
      },
      {...}
   ]
}
```

Fig. 13. WantedList.jsonld

the semantic concepts that were associated to properties are defined by publicly available vocabularies, such as FOAF, SCHEMA.org and DBPEDIA. According to Panziera [14], it is a good practice to associate properties with shared and known terms, given that this allows users to more easily understand their semantic meaning.

```
@Path("wanted")
@SemanticClass(id = "dbpedia-owl:Criminal")
@Vocabularies({
 @Vocabulary(prefix="dbpedia-owl", url="http://dbpedia.org/ontology/"),
 @Vocabulary(prefix="foaf", url="http://xmlns.com/foaf/0.1/"),
 @Vocabulary(prefix="vocab", url="http://api.com/doc/")})
public class Wanted {

    @SemanticProperty(id = "foaf:givenName")
    private String givenName;

    @SemanticProperty(id = "foaf:surname")
    private String familyName;

    @SemanticProperty(id = "vocab:reward")
    private double reward;

    @SemanticHyperlink(id = "foaf:img")
    private String mainPhoto;

    @SemanticHyperlink
    private Summary summary = new Summary();

    @SemanticHyperlink
    private Description description = new Description();

    @SemanticHyperlink
    private Photos photos = new Photos();

    @SemanticHyperlink
    private Crimes crimes = new Crimes();

    @SemanticHyperlink
    private ScarsAndMarks scarsAndMarks = new ScarsAnMarks();

    @SemanticHyperlink
    private Aliases aliases = new Aliases();
    ...
```

Fig. 14. Wanted.java - Properties

Besides these properties, the Wanted class has three operations, which are shown in Fig.15. The first operation has the @RetrieveResourceOperation annotation and is responsible for obtaining the REST representation of the resource. The other two operations are responsible for modifying and removing the resource, and are annotated with @ReplaceResourceOperation and @DeleteResourceOperation respectively. These three operations are mapped to URI templates that follow the *wanted/{id}* template specified by the @Path annotation. When the URI template of operations has variables, such as the {id} in this example, they must necessarily be class properties and their values must be associated with the URI path. The Hydra serialization of the Wanted representation, which is produced by method load, annotated with @RetrieveResourceOperation, is shown by Fig. 16.

```
private Integer id;

@GET
@Path("/{id}")
@RetrieveResourceOperation
public Response load(@PathParam("id") Integer id){
    //code to load
}

@PUT
@Path("/{id}")
@ReplaceResourceOperation
public Response replace(@PathParam("id") Integer id){
    //code to replace
}

@DELETE
@Path("/{id}")
@DeleteResourceOperation
public Response delete(@PathParam("id") Integer id){
    //code to delete
}
```

Fig. 15. Wanted.java - Operations

```
{
   "@context":"http://api.com/context.jsonld",
   "@id":"/wanted/1",
   "@type":"Wanted",
   "givenName":"Albert",
   "familyName":"DeSalvo",
   "reward":100000.00,
   "mainPhoto":"http://gallery.com/ads.jpg",
   "summary":"/wanted/1/summary",
   "description":"/wanted/1/description",
   "scarsAndMarks":"/wanted/1/scarsAndMarks",
   "photos":"/wanted/1/photos",
   "crimes":"/wanted/1/crimes",
   "aliases":"/wanted/1/eliases"
}
```

Fig. 16. Wanted.jsonld

D. Documenting the Web API

Based on the annotated source code, the framework generates Web pages containing the description of the representation classes, resulting in a complete documentation of the Web API enriched with semantic meaning. Fig. 17 shows the main page of the documentation generated for the FBI Wanted List Web API, with all supported classes listed and the entry point clearly indicated.

Fig. 18 shows the documentation of the WantedList class in more detail. This page describes all supported properties and operations. The @id property represents the URI that identifies the resource. The members property is a Collection that contains resources with semantic meaning described by the link to term dbpedia-owl:Criminal. There is only one supported operation, which adds new resources to the collection. The description also specifies the input and output values expected by the operation. Fig. 19 shows the documentation for class Wanted, in which three different kinds of property exist. The ones labeled as property have their value contained in the representation itself, while the ones labeled with IRI and Collection are, respectively, hyperlinks to another representation or to a list of representations.

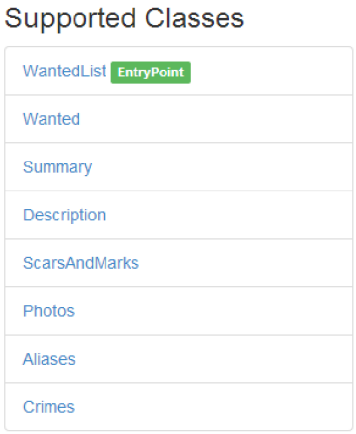

Fig. 17. Supported Classes

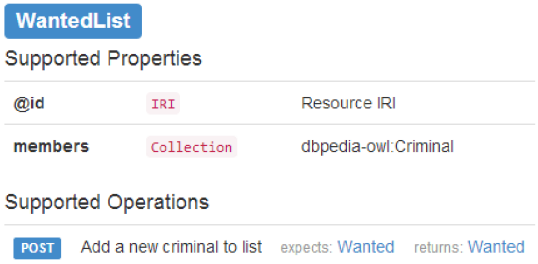

Fig. 18. WantedList - Documentation

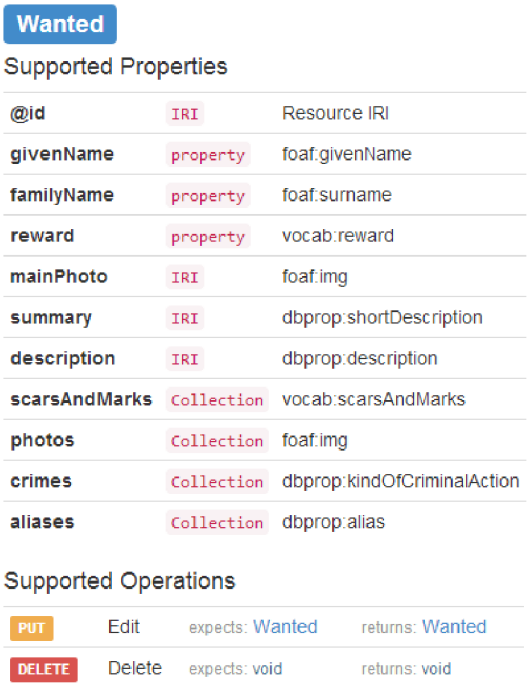

Fig. 19. Wanted - Documentation

VI. Related Work

RESTful Objects [15] is a specification targeted at RESTful application development, which provides access to domain objects through HTTP and produces resource representations in JSON format. Domain classes can expose their properties, operations and collections containing other entities. This work considers that each class in the application domain is a resource that has a serialized representation in JSON. Resources may have sub-resources, which also expose their properties. The authors introduce the concept of an Object Action, which represents an operation that can be invoked through the REST interface. A profile added to the JSON media type adds the required semantics to handle hypermedia controls and to bind the serialized domain object to a semantic concept or type.

Spring-HATEOAS [16] is a framework aimed at the development of hypermedia-driven RESTful Web Services with the Java programming language. This framework allows creating representation classes for REST resources with hypermedia controls. These representations are made available in JSON format semantically enriched using the Hypertext Application Language (HAL), which allows hyperlinks to be added to a JSON object. The design principles proposed by the framework introduce the Resource Representation Class, which describes only the properties of the corresponding resource. Operations are available through a Resource Controller, which handles representation classes and adds the required hyperlinks.

RESTful Objects, Spring-HATEOAS and the framework proposed in this paper employ the JSON format to serialize resource representations. However, different approaches are adopted for describing protocol and application semantics. REsTful Objects employs the 'profile' property to specify the representation semantics, while Spring-HATEOAS adopts the HAL language for the same purpose. The proposed framework, on the other hand, is based on JSON-LD, which is an official W3C recommendation. Apache Isis, which is a Java implementation of the RESTful Objects specification, employs annotations to expose domain objects, but does not allow defining if properties will be represented directly by their value or by hyperlinks. Spring-HATEOAS reuses the annotations provided by the Spring-MVC framework. Hyperlinks have to be created programmatically.

John and Rajasree [17] propose a framework for description, discovery and composition of semantic RESTful services through semantic annotations added to the Web API documentation. Despite being useful to allow the integration of automated agents, solely semantically enriching the documentation provides the client with a single source of information to guide its interaction with the Web API. In contrast, the Hypermedia Support Framework allows the documentation to be generated through code annotations, but also provides hypermedia controls in resource representations as the main guide for client interaction. This strategy reduces the impact on client software caused by eventual changes in the Web API.

Jung et al. [18] have developed a method for automated composition of RESTful Web Services. This method groups services in different categories based on their associated domain ontologies. Similarly, the Hypermedia Support Framework provides support for using ontologies and vocabularies to describe resource representations, aiming to simplify the

composition of services provided by Web APIs.

RESTdesc [19] provides an approach to semantically describe RESTful services by specifying preconditions, post-conditions and request details, which are grouped as rules. Through the semantic enrichment of URI templates, this work aims to allow the automation of the discovery and execution of Web Services. However, in hypermedia-driven Web APIs the URI template is just a side-effect of using hypermedia controls in resource representations, instead of the main mechanism for resource description, which increases coupling between client code and the service provider.

Based on a set of good practices for Web API design, Panziera and De Paoli [14] propose a framework able to semi-automate the generation of RESTful service descriptions. One of the recommended practices is the use of vocabularies to bind resource properties with well-known concepts. The proposed approach is based on description providers distributed through the Web, which provide information on services for humans and autonomous agents. Service descriptions become available to consumers as REST resources, resulting in a paradigm called by the authors "Description as a Service". This approach is an alternative to the use of hypermedia controls, but requires some level of human intervention and may result in the same limitations faced by service repositories (e.g., UDDI).

VII. Conclusion

This paper introduced a framework for developing Web APIs with the Java programming language. The proposed framework is based on the JAX-RS specification and provides support for hypermedia and linked data. This framework enforces the designer to closely follow concepts and constraints established by the REST architectural principle. By offering means to associate resources, properties and operations with vocabulary terms that have a clearly specified semantic meaning, the construction of Web API clients, service mashups and autonomous clients is simplified.

Similar frameworks aimed at developing RESTful Web APIs with hypermedia support have limited ways to describe resource representations, particularly in the support for description of operations, which is the main element in the HATEOAS principle. Most frameworks found in the literature focus on the description of the Web API, which, despite being a relevant benefit, encourages the development of clients based on information obtained from human-readable documentation. Documentation-driven development increases coupling among API and client code, may also increase the need for client maintenance due to changes in the Web API.

A prototype implementation of the proposed framework has been developed on the Java EE Platform. We intend to further explore its applicability and evaluate its features with the development of autonomous clients and service mashups.

References

[1] R. T. Fielding, "REST: architectural styles and the design of network-based software architectures," Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[2] M. Lanthaler and C. Gütl, "On using json-ld to create evolvable restful services," in *Proceedings of the Third International Workshop on RESTful Design*, ser. WS-REST '12. New York, NY, USA: ACM, 2012, pp. 25–32. [Online]. Available: http://doi.acm.org/10.1145/2307819.2307827

[3] M. Lanthaler, "Creating 3rd generation web apis with hydra." International World Wide Web Conferences Steering Committee / ACM, 2013, pp. 35–38.

[4] M. Amundsen. (2013, 02) Collection+json. [Online]. Available: http://amundsen.com/media-types/collection/

[5] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001.

[6] P. Adamczyk, P. Smith, R. Johnson, and M. Hafiz, "Rest and web services: In theory and in practice," in *REST: From Research to Practice*, E. Wilde and C. Pautasso, Eds. Springer New York, 2011, pp. 35–57. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8303-9_2

[7] ORACLE, "JAX-RS: Java API for RESTful Web Services - Version 2.0 Public Review (Second Edition)," JSR 399, Tech. Rep., 2012. [Online]. Available: https://jcp.org/aboutJava/communityprocess/final/jsr339/index.html

[8] L. Richardson, M. Amundsen, and S. Ruby, *Restful Web Apis*. Oreilly & Associates Incorporated, 2013.

[9] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström, "Json-ld 1.0 a json-based serialization for linked data," Tech. Rep., 10 2013. [Online]. Available: http://json-ld.org/spec/latest/json-ld/

[10] M. Lanthaler, "Creating 3rd generation web apis with hydra," in *Proceedings of the 22Nd International Conference on World Wide Web Companion*, ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 35–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487788.2487799

[11] M. Lanthaler and C. Gütl, "Model your application domain, not your json structures," in *Proceedings of the 22Nd International Conference on World Wide Web Companion*, ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 1415–1420. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487788.2488184

[12] O. Liskin, L. Singer, and K. Schneider, "Welcome to the real world: A notation for modeling rest services," *Internet Computing, IEEE*, vol. 16, no. 4, pp. 36–44, 2012.

[13] I. Zuzak, I. Budiselic, and G. Delac, "Formal modeling of restful systems using finite-state machines," in *Web Engineering*, ser. Lecture Notes in Computer Science, S. Auer, O. Díaz, and G. Papadopoulos, Eds. Springer Berlin Heidelberg, 2011, vol. 6757, pp. 346–360. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22233-7_24

[14] L. Panziera and F. De Paoli, "A framework for self-descriptive restful services," in *Proceedings of the 22Nd International Conference on World Wide Web Companion*, ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 1407–1414. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487788.2488183

[15] D. Haywood, "Restful Objects Specification(v1.0.0)," Tech. Rep., 2012. [Online]. Available: http://restfulobjects.org

[16] SPRING. (2013, 05) Spring hateoas - reference. [Online]. Available: http://projects.spring.io/spring-hateoas/

[17] D. John and M. S. Rajasree, "A framework for the description, discovery and composition of restful semantic web services," in *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, ser. CCSEIT '12. New York, NY, USA: ACM, 2012, pp. 88–93. [Online]. Available: http://doi.acm.org/10.1145/2393216.2393232

[18] W. Jung, S. I. Kim, and H. S. Kim, "Ontology modeling for rest open apis and web service mash-up method," vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2013, pp. 523–528.

[19] R. Verborgh, T. Steiner, D. Deursen, J. Roo, R. d. Walle, and J. Gabarró Vallés, "Capturing the functionality of web services with functional descriptions," *Multimedia Tools and Applications*, vol. 64, no. 2, pp. 365–387, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11042-012-1004-5