Rapid Deployment of a RESTful Service for Data Collected by Oceanographic Research Cruises

Linyun Fu[1] and Robert A. Arko[2]

[1]Tetherless World Constellation, Rensselaer Polytechnic Institute, Troy, NY, United States
[2]Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY, United States
ful2@rpi.edu, arko@ldeo.columbia.edu

Abstract. The Rolling Deck to Repository (R2R) program has the mission to capture, catalog, and describe the underway environmental sensor data from US oceanographic research vessels and submit the data to public long-term archives. Information about vessels, sensors, cruises, datasets, people, organizations, funding awards, logs, reports, etc. is published online as Linked Open Data, accessible through a SPARQL endpoint. In response to user feedback, we are developing a RESTful service based on the Elda open-source Java package to facilitate data access. Our experience shows that constructing a simple portal with limited schema elements in this way can significantly reduce development time and maintenance complexity compared to PHP or Servlet based approaches.

Introduction. The Rolling Deck to Repository (R2R) program addresses the need for consistent preservation and dissemination of environmental sensor data routinely acquired by oceanographic research vessels in the U.S. academic fleet. R2R collects information about each expedition that includes vessel identifier and operator; cruise identifier, project title, and research program; port stops and dates; science party names, institutions, and roles; funding agency and award identifier; sensor identifier, classification, manufacturer, and model; cruise reports and event logs; and file manifests.

R2R publishes content as Linked Data [1] using the D2RQ[1] software package, which transforms content from a SQL database to RDF resources in a virtual triple store. Content is mapped to community-standard controlled vocabularies where these are available online as RDF with stable URIs, such as the NERC Vocabulary Server[2].

For example, one vessel called "Atlantis" is assigned the identifier <http://data.rvdata.us/id/vessel/33AT> and is described with the following triples, encoded in Turtle format for the sake of simplicity.

```
<http://data.rvdata.us/id/vessel/33AT>
  rdf:type <http://data.rvdata.us/vocab/id/class/Vessel> ;
  dcterms:identifier "33AT" ;
  rdfs:label "Atlantis" ;
  r2r:Operator <http://data.rvdata.us/id/organization/edu.whoi> ;
```

[1] D2RQ. http://d2rq.org/d2r-server
[2] NERC Vocabulary Server (NVS), Version 2.0. http://vocab.nerc.ac.uk/

adfa, p. 1, 2011.

```
r2r:Owner <http://data.rvdata.us/id/organization/mil.navy> ;
skos:exactMatch <http://vocab.nerc.ac.uk/collection/C17/current/33AT/> .
```

The Linked Data API[3] was developed in response to a requirement from Web developers that Linked Data in the RDF data structure should be accessible in a way that is familiar to them, namely through RESTful services [2], in addition to through SPARQL endpoints. The Linked Data API achieves this goal by defining a proxy layer on top of existing SPARQL endpoints that 1) translates HTTP requests into SPARQL queries, and 2) renders the returned results as required by the request sender using content negotiation, suffixes and parameters.

The first part, HTTP-request-to-SPARQL-query translation, is done by modules called *selectors*, whereas the second part, rendering, is done by *viewers* and *formatters*. Selectors, viewers and formatters are usually grouped together to form *endpoints*. Unlike SPARQL endpoints which accept SPARQL queries, Linked Data API endpoints accept HTTP requests. Figure 1 shows some important classes along with relationships among them in Linked Data API, created with COE [3].

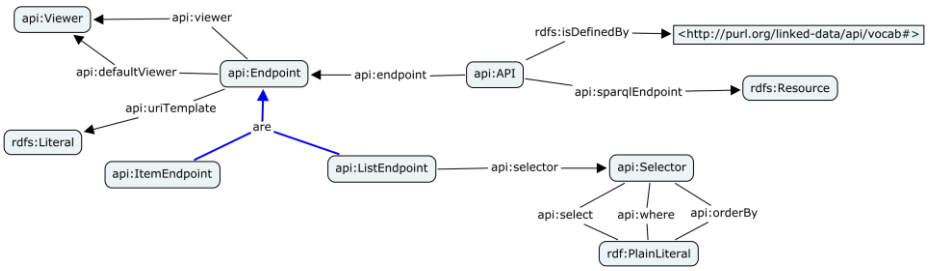

Fig. 1. Part of Linked Data API ontology[4]. Prefixes used:
rdfs: http://www.w3.org/2000/01/rdf-schema#
api: http://purl.org/linked-data/api/vocab#

Note that only part of the whole Linked Data API ontology is shown in Figure 1 for the sake of simplicity. For example, we do not show the formatter class in the Figure because it deals with the detailed representation of the returned Web page in response to an HTTP request and is not the focus of this paper. We also omit the `rdfs:Literal` valued `api:base` property of the `api:API` class here because it is best illustrated, and will be shown through a detailed example in the next section.

The HTTP request pattern that an endpoint accepts is encoded in its `api:uriTemplate` field, and specification for selecting resources is encoded in terms of `api:select`, `api:where`, `api:orderBy`, etc.

Once implemented as a software package, this API enables Linked Data publishers or proxy builders to create their RESTful service by writing a configuration file containing only the definition of an `api:API` instance. The `api:API` instance is recognized upon invocation of the HTTP request, and the RESTful service backed by the

[3] Linked Data API Specification. https://code.google.com/p/linked-data-api/wiki/Specification
[4] The full ontology is online at http://purl.org/linked-data/api/vocab#

software performs the translation and rendering jobs according to the `api:API` instance definition. We will show in the next section how we define this instance along with one of its associated endpoints and the endpoint's selector to create our RESTful service for the R2R dataset.

Configuration of the RESTful Service. Figure 2 shows how the `api:API` instance is defined in our application. It talks to the SPARQL endpoint located at `http://data.rvdata.us/sparql`, as its `api:sparqlEndpoint` value indicates. Here we just show one Linked Data API endpoint of this instance, namely `spec:listVesselsByLabelContains`. This endpoint is responsible for listing all the resources in the dataset that have a certain substring in their `rdfs:label` fields.

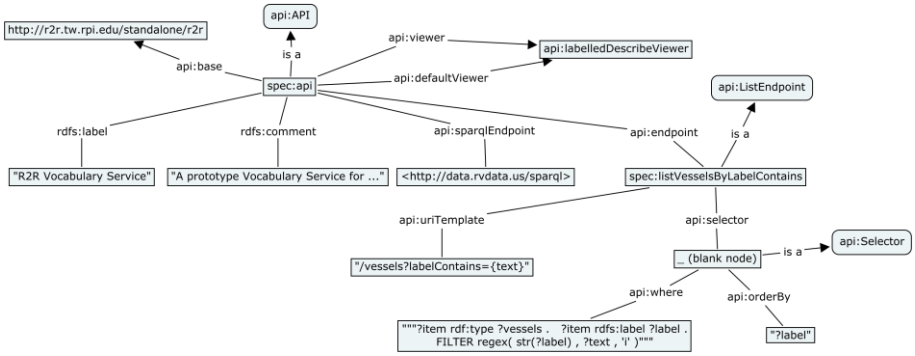

Fig. 2. Part of R2R Linked Data API Configuration.
In addition to the prefixes used in Figure 1, we have:
spec: http://r2r.tw.rpi.edu/spec/r2r#

The `spec:listVesselsByLabelContains` endpoint deals with HTTP requests matching the pattern `base:/vessels?labelContains={text}`, as its `api:uriTemplate` field indicates. The `base` prefix is the `api:base` value of the `spec:api`, so this endpoint responds to requests such as `http://r2r.tw.rpi.edu/standalone/r2r/vessels?labelContains=Atlantis`. This endpoint uses a selector to fulfill its duty in a way that is encoded as `api:where` and `api:orderBy` values. These values are simply the `WHERE` clause and the `ORDER BY` clause in a SPARQL query. The actual query submitted to the SPARQL endpoint is as follows.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?item
WHERE { ?item rdf:type <http://data.rvdata.us/vocab/id/class/Vessel> .
          ?item rdfs:label ?label .
          FILTER regex( str(?label) , "Atlantis" , 'i' ) }  ORDER BY ?label
```

The endpoint received a result for this SPARQL query, and creates a view with the `api:labelledDescribeViewer`, as indicated by the `api:defaultViewer` value of the `spec:api`. The viewer returns the graph created from a DESCRIBE query for each query result, supplemented by labels for linked resources. The final Web page rendered by Elda[5], which is one implementation of the Linked Data API, is shown in Figure 3.

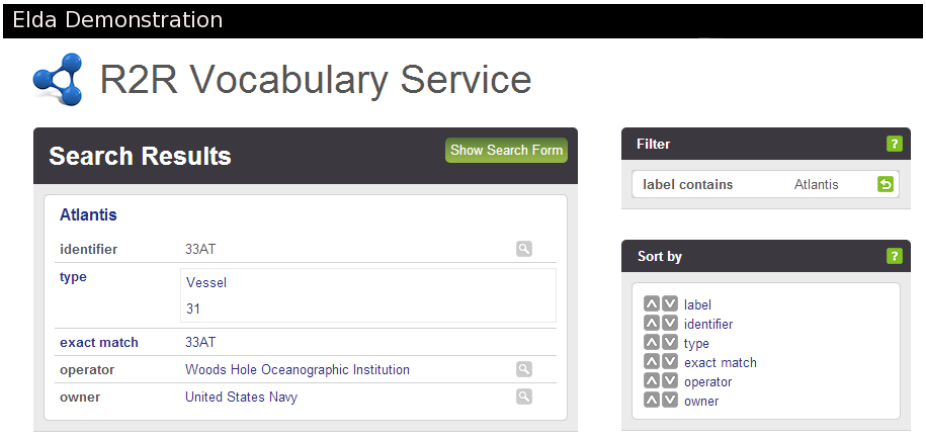

Fig. 3. HTML response to the HTTP request `base:/vessels?labelContains=Atlantis`

Conclusion. The Elda approach for creating RESTful services enables Semantic Web engineers to reach a broad community of Web developers. It requires minimal coding to write a configuration file in Turtle (or other RDF formats) and expose a triple store in a RESTful way, thus enabling construction and maintenance of a Linked Data-driven Web portal in a lightweight manner.

Acknowledgements. This research is funded by the U.S. National Science Foundation via the Rolling Deck to Repository (R2R) program and the Ocean Data Interoperability Platform (ODIP), working collaboratively with the U.S. University-National Oceanographic Laboratory System (UNOLS) Office.

References

1. Christian Bizer, Tom Heath and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems* 5.3 (2009): 1-22.
2. Leonard Richardson and Sam Ruby. *RESTful Web Services*, O'Reilly, ISBN 978-0-596-52926-0 (2007).
3. Pat Hayes, Thomas C. Eskridge, Mala Mehrotra, Dmitri Bobrovnikoff, Thomas Reichherzer, and Raul Saavedra. COE: Tools for Collaborative Ontology Development and Reuse. *Knowledge Capture Conference (KCAP)* (2005).

[5] Elda: the linked-data API in Java. http://www.epimorphics.com/web/tools/elda.html