Linked Data Wrappers atop Yahoo's YQL

Jon Iturrioz, Iker Azpeitia and Oscar Díaz

Univ. of the Basque Country (EHU/UPV), Donostia, Spain. *{name.surname}@ehu.es*

The *Yahoo Query Language* (YQL) specializes on providing a framework for abstracting developers from the heterogeneity of API requests and its optimization [1]. This specialization is what makes YQL attractive for LDW development. YQL abstracts APIs through the so-called **Open Data Tables** (ODT). As an example, consider *Last.fm*. This is a music website that provides information about musical events through an Open API[1]. ODTs abstract API specifics through a table-like view. Figure 1 shows the ODT for the service *lastfm.event.getInfo*[2]. Main tags include *<meta>* and *<bindings>*. The former contains descriptive information about the ODT such as author, description or documentation link 3-5). The bindings indicate how YQL operations are mapped into API calls. An entry exists for each operation (e.g. *<select>*). The sample case illustrates the SELECT case (10-16): *<url>* accounts for the URL pattern to invoke whereas *<inputs>* denotes the possible YQL statement input field. Each field (e.g. event) accounts for variables to be instantiated when SELECT is enacted. YQL promotes reuse through ODTs so that the YQL community can tap on someone else's ODT for their own developments.

ODTs provide a good starting point for LDWs. This entails a double wrapping: *YQL wraps APIs as tables while LDWs wrap tables as linked data.* Grounding and lifting concerns are considered during the second wrapping. Back to our running example, this is achieved as follows (see Figure 1).

YQL's *sampleQuery* tag is used to describe the URI pattern (line 6) and the URI grounding (line 7). When a linked wrapper server receives a URI, it identifies the ODT at hand through pattern matching against the registered *URIPattern* at deployment time. This pattern is attached to the linked wrapper server base URL (e.g. `http://www.onekin.org/`). The binding (lowering mapping) from URI pattern to ODT input parameters is realized through pattern matching parameters (i.e. line 6 to line 13 {event} binding).

YQL's *function* tag is recast for lifting: each YQL tuple (i.e. *oneEventXML*) is to be turned into an RDF individual (i.e. *oneEventJSONLD*). The lifting function holds *<inputs>* and *<execute>*. The former indicates the function's parameters which are set to *<pipe>*, i.e. holds "a tuple" of the ODT table described à la XML (line 19), and *<key>* (i.e. to cast the ID for the returned RDF individual) (line 20). As for *<execute>*, it holds the JavaScript code that obtains JSON-LD out of the XML tuple (line 22-28). Interlinkage is also described here by constructing URIs out of existing parameters, e.g. *mo:performer*

[1] `http://www.last.fm/api`.

[2] Full description at `https://github.com/yql/yql-tables/blob/master/lastfm/lastfm.event.getinfo.xml`.

```
 1<table xmlns="http://query.yahooapis.com/v1/schema/table.xsd">
 2<meta>
 3 <author>Jamie Matthews</author><author>Iker Azpeitia</author>
 4 <description> ODT for Last.fm event.getInfo API method. </description>
 5 <documentationURL>http://www.last.fm/api/show/event.getInfo</documentationURL>
 6 <sampleQuery>URIPattern: lastfm/event/{event}</sampleQuery>
 7 <sampleQuery>URIexample: lastfm/event/3986264</sampleQuery>
 8</meta>
 9<bindings>
10 <select itemPath="" produces="XML">
11  <urls><url>http://ws.audioscrobbler.com/2.0/?method=event.getinfo</url></urls>
12  <inputs>
13   <key id="event" type="xs:string" paramType="query" required="true" />
14   <key id="api_key" type="xs:string" paramType="query" required="true" />
15  </inputs>
16 </select>
17 <function name="lifting">
18  <inputs>
19   <pipe id="oneEventXML" paramType="variable" />
20   <key id="URI" paramType="variable" required="true"/>
21  </inputs>
22  <execute> <![CDATA[var oneEventJSON= y.xmlToJson(oneEventXML);
23   var oneEventJSONLD={'@context':{dc:'http://purl.org/dc/elements/1.1/',
   _                                 mo:'http://purl.org/ontology/mo/',}};
24   oneEventJSONLD['@id']= URI;
25   oneEventJSONLD['@type']='mo:Performance';
26   oneEventJSONLD['dc:date']=oneEventJSON.lfm.event.startDate;
27   oneEventJSONLD['mo:performer']='http://rdf.onekin.org/musicbrainz/artist/'
   _                                   +oneEventJSON.lfm.event.artists.artist;
28   response.object = oneEventJSONLD;]]> </execute>
29 </function>
30</bindings>
31</table>
```

Fig. 1. LDW definition for `lastfm.event.getinfo`.

links to the *MusicBrainz* resource about the artist (line 27). Noteworthy, this interlinkage is realized through another LDW.

Benefits are three-fold. First, LDWs are sheltered from changes in the underlying APIs. API upgrades are handled at the YQL-table level without impacting the LDWs built on top. Second, reuse. LDW developers can tap into existing ODT (1068 at the time of this writing). Third, infrastructure. Yahoo supports ODT enactment by providing load balancing and graceful performance degradation. Dereferencing (which implies ODT enactment) then enjoys these pluses. See it at work by dereferencing `http://rdf.onekin.org/lastfm/event/3986264`.

References

1. Yahoo! Developer Network. Yahoo Query Language (YQL) guide. https://developer.yahoo.com/yql/guide/.