YQL as a Platform for Linked-Data Wrapper Development

Jon Iturrioz, Iker Azpeitia(✉), and Oscar Díaz

University of the Basque Country (UPV/EHU), San Sebastián, Spain
{jon.iturrioz,iker.azpeitia,oscar.diaz}@ehu.eus

Abstract. Linked-Data Wrappers (LDWs) have been proposed to integrate Open APIs into the linked-data cloud. A main stumbling block is maintenance: LDWs need to be kept in sync with the APIs they wrap. Hence, LDWs are not single-shot efforts, but sustained endeavors that developers might not always afford. As a result, it is not uncommon for third-party LDWs to stop working when their underlying APIs upgrade. Collaborative development might offer a way out. This requires a common platform and a community to tap into. This work investigates the suitability of the YQL platform for this job. Specifically, we look into two main properties for LDW success: effectiveness (i.e. the capability of YQL to enable users to develop LDWs) and scalability (i.e. graceful time degradation on URI dereferencing). The aim: moving LDW development from in-house development to collaborative development as promoted by YQL, on the hope of increasing LDWs' lifespan.

Keywords: Linked data wrappers · YQL · Open APIs

1 Introduction

The shortage of RDF end-points is hindering the development of the Web of Data. Turning existing data into RDF triplets might be conducted by data providers or data consumers. For example, the former is illustrated by R2RML [8] (for database sources) or OWL-S (for Web Service) [15]. Unfortunately, this approach has not caught on, suffering from the chicken-and-egg "cold-start" problem: providers (i.e. *eBay*, *Amazon*) lack a clear demand for RDF while consumers stick to JSON/XML because the learning curve and lifting effort to move to RDF. Alternatively, data consumers can construct themselves linked-data wrappers (LDWs) [4,5]. Third parties can develop LDWs at their own expense. In this case, maintenance becomes the main issue. Wrappers in general, and LDWs in particular, hold a tight coupling on the underlying platform (i.e. the Open API). If the API changes then, LDWs might need to be rewritten. The severity of this problem for LDWs stems from the frequency of change (APIs are reckoned to evolve regularly) and the developers' profile (in general research groups which might lack the resources in keeping LDWs on and running). The question is how to promote LDW development in a setting characterized by limited rewards and heavy maintenance. Collaborative development emerges as a possible answer.

© Springer International Publishing Switzerland 2015
P. Cimiano et al. (Eds.): ICWE 2015, LNCS 9114, pp. 355–373, 2015.
DOI: 10.1007/978-3-319-19890-3_23

Collaborative development rests on public availability and communication, usually via the Internet. This involves the existence of a platform. We look at Yahoo's YQL (*Yahoo Query Language*) platform [17] as an appropriate venue for LDW development. So far, this platform is being used for mashup developers to query, filter, and combine data across the Web through a single SQL-like interface. Turning YQL into a LDW platform, introduces two challenges:

- attracting producers. LDW development is programming intensive. Broadening the range of producers might be achieved by reducing the programming effort as well as embracing sibling communities. YQL facilitates both aims. First, YQL abstract the level at which API programming is conducted. Rather than facing API specifics, YQL hides this complexity through a table-like view. This speeds API programming in general, and LDW development in particular. Second, there already exists a YQL community. By adopting their own tools, we hope to tap into this community.
- attracting consumers. LDW continuous effort pays off if benefits go beyond breakout developers. So far, most LDWs are seldom used outside their research projects. If LDWs are to evolve beyond proof-of-concept, scalability issues should be considered. Graceful degradation of elapsed times promotes linkage with your LDW-supported linked data. YQL can help by providing load balancing that outperforms small-scale attempts to host LDW services.

Hence, this paper presents a case for YQL as an LDW platform by conducting two evaluations on effectiveness (i.e. the capability of YQL to enable users to develop LDWs with accuracy and completeness) (Section 4) and scalability (Section 5). So far, we focus on read-only API methods. We start by introducing a running example.

2 Running Example

The lack of services exposing linked-data is due to heterogeneous circumstances: technical (i.e. for complicated domains, mapping the underlying data representation to linked data formats is nontrivial), social (i.e. no demand on linked-data representation by the service community) or financial (i.e. no clear business model). Fortunately, in case the data is available under a liberal license[1], wrapping the data into a service separate from the original website, might be possible.

As an example, consider *Last.fm*. This is a music website that among other data, provides information about musical events. An Open API facilitates programmatic access to this data[2]. Output formats include XML and JSON but not linked data. As most current APIs, *Last.fm* behaves as a data silo with no interlinkage with other sources. Hence, moving *Last.fm* to the Linked Data (LD) cloud requires not only a change in the output format (e.g. JSON-LD [3]) but also

[1] See http://www.w3.org/TR/void/#license.
[2] http://www.last.fm/api

```
{
  "@context":{
    "rdfs":"http://www.w3.org/2000/01/rdf-schema#",
    "dc":"http://purl.org/dc/elements/1.1/",
    "mo":"http://purl.org/ontology/mo/",
    "event":"http://purl.org/NET/c4dm/event.owl#",
    "wth":"http://www.scs.ryerson.ca/~bgajdero/msc_thesis/code/ontologies/weather-ont-t2.owl",
    "dbpprop":"http://dbpedia.org/property/"
  },
  "@id":"http://rdf.onekin.org/lastfm/event/3986264",
  "@type":"mo:Performance",
  "dc:date":"Thu, 06 Nov 2014 20:00:00",
  "rdfs:label":"Guerrera",
  "event:place":"http://www.dbpedia.org/resource/Donostia",
  "mo:performer":"http://rdf.onekin.org/musicbrainz/artist/Guerrera",
  "dbpprop:hasPhotoCollection":"http://rdf.onekin.org/flickr/location/Donostia",
  "wth:hasWeather":"http://rdf.onekin.org/weather/location/Donostia"
}
```


Fig. 1. A *Last.fm* instance in JSON-LD format

setting links with other URI-addressable related sources. Figure 1 illustrates how an *event* instance might look like. Its URI is http://rdf.onekin.org/lastfm/event/3986264. Its description holds references to other resources such as the event's place (through the property *event:place*) or the event's performers (through the property *mo:performer*). This view is provided by a LDW which publicizes *Last.fm*'s event instances as URI-addressable resources by means of both lifting API-recovered data and interlinkaging it with other resources in the LD cloud. These issues are already covered in the literature [18][21,22].

Here, we rise a different concern. Our dismay is not so much about development but maintenance, i.e. how to increase the chances of our *Last.fm* LDW to be up and active in the medium run. The endeavor should not be underestimated. Open API upgrades are not uncommon, and development teams might need to change focus. No wonder distinct LDWs that properly worked at the time they were launched, they are no longer up at the time of this writing.

It is not odd for LDW efforts to stop working at some point: *Flickr wrappr* [4], *GoogleArt project to RDF*[3], *DBTune.org Artists*[4] (a wrapper on top of *Last.fm*), or *Twitter wrapper*[5]. Therefore, anecdotal evidence highlights the importance of maintenance for LDWs to be a sustainable foundation for the Web of Data. Collaborative code-development platforms come to the rescue.

[3] http://linkeddata.few.vu.nl/

[4] http://dbtune.org/

[5] http://km.aifb.kit.edu/services/twitterwrap/

3 Collaborative Code-Development Platforms and The YQL Web Service

A collaboration platform offers broad social networking capabilities to work processes. If this work is aimed at code development then, the platform provides repository hosting service, including distributed revision control and source code management. A main exponent of this kind of platforms is *GitHub*[6]. *GitHub* permits users to browse public repositories on the site. It provides social networking functions such as feeds, followers, wikis and a social network graph to display how developers work on their versions ("forks") of a repository. On top, services can be built that use *GitHub* as a repository. This is the case of YQL. YQL specializes on providing a framework for abstracting developers from the heterogeneity of API requests and its optimization. YQL is not thought for application development in general, but for abstracting from API heterogeneity. This specialization is what makes YQL attractive for LDW development. LDWs also aim at abstracting from Open APIs in terms of RDF end-points. The YQL Guide states "YQL Web Service enables applications to query, filter, and combine data from different sources across the Internet" [17]. LDWs pursuit similar goals. The difference stems from the abstraction at which these aims are conducted. YQL envisions the Web in term of tables (known as Open Data Tables (ODT)), and returns XML documents. By contrasts, LDWs support the linked-data view, and returns RDF triplets. This work is based on this likeness.

YQL statements have a SQL-like syntax, familiar to any developer with database experience. It aims at hiding APIs' specifics into a uniform table-like metaphor. The following YQL statement, for example, retrieves data about the event whose ID is *3986264:*

*select * from lastfm.event.getinfo where event="3986264"*

To access the YQL Web Service, a Web application can call HTTP GET, passing the YQL statement as a URL parameter, for example:

http://query.yahooapis.com/v1/public/yql?q=*select * from l*astfm.event.getinfo where event="3986264"

This setting is achieved through three mechanisms: *Open Data Tables* (ODT), the *Yahoo Query Language* (YQL), and the YQL console.

Open Data Tables. Broadly, ODTs are syntactic sugar for API parameters. Figure 2 shows the *lastfm.event.getInfo* ODT[7]. Main tags include <*meta*> and <*bindings*>. The former contains descriptive information about the ODT such as author, description or documentation link (3-7). The bindings (8-16) indicate how SQL operations are mapped into API calls. An entry exists for each operation (e.g. <*select*>, <*insert*>). The sample case illustrates the SELECT

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

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <table xmlns="http://query.yahooapis.com/v1/schema/table.xsd">
3  <meta>
4   <author>Jamie Matthews</author>
5   <description>YQL table for Last.fm Event.getInfo API method.</description>
6   <documentationURL>http://www.last.fm/api/show?service=292</documentationURL>
7  </meta>
8  <bindings>
9   <select itemPath="" produces="XML">
10   <urls><url>http://ws.audioscrobbler.com/2.0/?method=event.getinfo</url></urls>
11   <inputs>
12    <key id="event" type="xs:string" paramType="query" required="true" />
13    <key id="api_key" type="xs:string" paramType="query" required="true" />
14   </inputs>
15  </select>
16 </bindings>
17 </table>
```

Fig. 2. *lastfm.event.getinfo* ODT

case (9-15): *<url>* accounts for the URL pattern to invoke whereas *<input>* denotes the possible YQL statement input field. Each field (e.g. event) accounts for variables to be instantiated when SELECT is enacted.

The YQL Language. YQL includes *SELECT*, *INSERT* and *DELETE* statements that, behind the curtains, invoke the corresponding API methods by means of ODTs. ODTs hold all the intricacies of the underlying APIs. Specifically, benefits can be obtained from reusing of the authorization and authentication code from YQL, given the many access control mechanisms with APIs. In this way, YQL offloads processing that programmers would normally do on the client/server side to the *YQL* engine. Besides those provided by YQL itself (known as "built-in tables"), YQL permits the community to provide their own ODTs[8]. Once loaded in the YQL repository, an API's ODT makes this APIs YQL accessible.

***The* YQL Console**[9] (see Figure 3). The YQL Console enables to run YQL statements interactively from your browser. Through this website, developers can check out their queries, look at the answers, and if satisfied, obtain the query's REST query counterpart. This URL can next be included in the user's programs.

The bottom line is that YQL provides ODTs as an alternative to direct API calls through optimization and abstraction. This is certainly of interest for LDW producers.

4 The LDW Community: The Producer Perspective

We believe a main challenge about LDW definition is not so much about "the how" but "the who". Good practices about LDW development already

[8] Community ODTs can be found at http://www.datatables.org/.
[9] http://developer.yahoo.com/yql/console/

Fig. 3. The YQL console includes three main windows: the upper window contains the YQL statement; the middle window displays the statement's output (e.g. an XML document); the bottom window contains the URL counterpart of the YQL statement

exists [19]. The challenge is widen the programmer base. Two strategies: reducing LDW development effort & embracing other communities.

Reducing LDW Development Effort. So far, LDWs tend to start from scratch [18][21,22]. YQL promotes reuse through ODTs. ODTs encapsulate lowering and formatting concerns for a given API. If ODTs are shared, programs start development from ODTs (i.e. issuing SQL-like statements) rather than going down to the API services. As any other API programming, LDWs can also benefit from this approach. This entails a double wrapping: *YQL wraps APIs as tables while LDWs wrap tables as linked data.*

Benefits are twofold. 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 YQL tables (1068 at the time of this writing).

Embracing Other Communities. At the time of this writing (March 2015), the page https://github.com/yql/yql-tables characterizes the YQL community along the following figures: 153 contributors, 3287 commits, 18 open issues, 14 closed, 14 open pull requests, 399 closed, 620 stars, and 425 forks. We believe LDW concerns are not so alien to API programmers. Broadly, wrapper definition pivots around two main issues: lowering (i.e. from URI dereferencing to API calls) and lifting (from API-call outputs to RDF resources). By moving to YQL, our hopes is to tap into this sibling community. However, evidences are needed that YQL is expressive enough to cater for LDW concerns.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <table xmlns="http://query.yahooapis.com/v1/schema/table.xsd">
3  <meta>
4   <author>Iker Azpeitia</author>
5   <description> ODT for Last.fm event.getInfo API method. </description>
6   <documentationURL> http://www.last.fm/api/show?service=292 </documentationURL>
7   <sampleQuery>URIPattern: lastfm/event/{event}</sampleQuery>
8   <sampleQuery>URIexample: lastfm/event/3986264</sampleQuery>          LOWERING
9  </meta>
10 <bindings>
11  <select itemPath="" produces="XML">
12   <urls><url>http://ws.audioscrobbler.com/2.0/?method=event.getinfo</url></urls>
13   <inputs>
14    <key id="event" type="xs:string" paramType="query" required="true" />
15    <key id="api_key" type="xs:string" paramType="query" required="true" />
16   </inputs>
17  </select>
18  <function name="lifting">                                              LIFTING
19   <inputs>
20    <pipe id="oneEventXML" paramType="variable" />
21    <key id="URI" paramType="variable" required="true"/>
22   </inputs>
23   <execute> <![CDATA[
24    var oneEventJSON= y.xmlToJson(oneEventXML);
25    var oneEventJSONLD={'@context':{rdfs:'http://www.w3.org/2000/01/rdf-schema#',
26    dc:'http://purl.org/dc/elements/1.1/', mo:'http://purl.org/ontology/mo/',
27    event:'http://purl.org/NET/c4dm/event.owl#', dbpprop:'http://dbpedia.org/property/',
28    wth:'http://www.scs.ryerson.ca/~bgajdero/msc_thesis/code/ontologies/weather-ont-t2.owl'}};
29    oneEventJSONLD['@id']= URI;
30    oneEventJSONLD['@type']='mo:Performance';
31    oneEventJSONLD['dc:date']=oneEventJSON.lfm.event.startDate;
32    oneEventJSONLD['rdfs:label']=oneEventJSON.lfm.event.title;
33    oneEventJSONLD['event:place']='http://dbpedia.org/resource/'
      _                                      +oneEventJSON.lfm.event.venue.location.city;
34    oneEventJSONLD['mo:performer']='http://rdf.onekin.org/musicbrainz/artist/'
      _                                          +oneEventJSON.lfm.event.artists.artist;
35    oneEventJSONLD['dbpprop:hasPhotoCollection']='http://rdf.onekin.org/flickr/location/'
      _                                      +oneEventJSON.lfm.event.venue.location.city;
36    oneEventJSONLD['wth:hasWeather']='http://rdf.onekin.org/weather/location/'
      _                                      +oneEventJSON.lfm.event.venue.location.city;
37    response.object = oneEventJSONLD;]]>
38   </execute>
39  </function>
40 </bindings>
41 </table>
```

Fig. 4. lastfm.event.getinfo LDW

The rest of this Section delves into these issues along three aspects: LDW development, LDW deployment and LDW adaptive maintenance.

4.1 LDW Development

This subsection builds upon YQL expressiveness to specify LDWs. As an example of reuse, we tap into the ODT in Figure 2, and turn it into a LDW (see Figure 4). The process includes two main steps.

First. YQL's *sampleQuery* tag is used to describe the URI pattern (line 7) and the URI grounding (line 8). When a linked wrapper server receives a URI (e.g. http://rdf.onekin.org/lastfm/event/3986264), it identifies the ODT at hand through pattern matching against the registered *URIPatterns* at deployment time. This pattern is attached to the linked wrapper server base URL (e.g. http://rdf.onekin.org/). The binding (lowering mapping) from URI pattern to

ODT input parameters is realized through pattern matching parameters (i.e. line 7 to line 14 {*event*} binding).

Second. 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 20) and <*key*> (i.e. to cast the ID for the returned RDF individual) (line 21). As for <*execute*>, it holds the JavaScript code that obtains JSON-LD out of the XML tuple (line 24-37). Interlinkage is also described here by constructing URIs out of existing parameters, e.g. *event:place* links to the *DBpedia* resource about the event's city (line 33)[10]. More interestingly, interlinkage can also be set with RDF individual counterparts of other API resources, e.g. *mo:performer* holds the URI for the event's artist kept at the *musicbrainz* API (line 34). Noteworthy, this interlinkage is realized through another LDW!

Let's check this out. Go to http://developer.yahoo.com/yql/console/ and paste the following snippet:

> *use 'https://raw.githubusercontent.com/onekin/ldw/master/events/lastfm.event.getinfo.xml' as* ***lastfm****; select * from* ***lastfm*** *where event = '3986264' and api_key = '20b0801ddb96b112ce4db2dbae134e10'*
> *|* ***lastfm****.lifting ('http://rdf.onekin.org/lastfm/event/3986264');*

The sample LDW is held in *GitHub*. The snippet starts by keeping in *lastfm* the path to locate this LDW. Sentence *select* retrieves XML documents associated with event *"3986264"*. For each tuple, *lastfm.lifting* returns its JSON-LD counterpart. Once the LDW is checked out, it is time to deploy it. More examples of LDW's can be found at https://github.com/onekin/ldw.

4.2 LDW Deployment

A main goal of LDW Servers is to make LDWs dereferenceable datasets. A dataset is a set of RDF resources that are published, maintained or aggregated by a single provider. The term dataset has a social dimension: we think of a dataset as a meaningful collection of triples, that deal with a certain topic, originate from a certain source or process, are hosted on a certain server, or are aggregated by a certain custodian. This process comprises:

1. *Register*. LDWs need to be registered to be considered datasets. This process is realized in a LDW Server (e.g. http://rdf.onekin.org/ldw/). Once registered, the server starts listening URIs that conform to the LDW *URIPattern*.
2. *URI Dereference* (see Figure 5 steps 1..8). When one URI is invoked (e.g. http://rdf.onekin.org/lastfm/event/3986264) (step 1), the LDW server (i.e.

[10] The approach of linking to other datasets by appending something (e.g. the name of a city) to some other dataset's namespace URI works often, but in the most general case, it's a bit naive. In this case, the developer should resort to JavaScript to set the mapping.

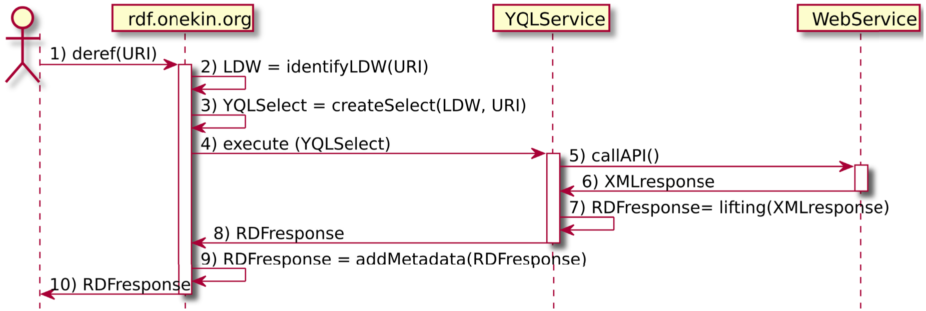

Fig. 5. Simplified sequence diagram

http://rdf.onekin.org) finds the LDW that matches the URI (step 2); lowers the URI to the corresponding YQL query (see the select in the previous section) (step 3); and sends the query to the YQL server (step 4). The YQL engine calls to the Web Service's API (steps 5 and 6) and applies the lifting function (step 7) resulting in a RDF resource description (step 8).

3. *Dataset Metadata Linkage.* Each resource pertains to a dataset. Each dataset contains metadata information that can be used in many situations, ranging from data discovery to cataloging and archiving. The Vocabulary of Interlinked Datasets (VoID) [2] is concerned with metadata about RDF datasets, and it is intended as a bridge between the publishers and users of RDF data. The server adds to the results a *void:inDataset* property that link to its VoID (Figure 5 step 9). This URI is also dereferenceable.
4. *Provenance Publication.* The ability to track the origin of data is a key component in building trustworthy, reliable applications [6]. A widely deployed vocabulary for representing such data is Provenance Ontology[11]. Provenance properties are automatically generated by the LDW server (Figure 5 step 9, resulting in the bottom part of the Figure 6[12]). The *prv:usedGuideline* indicates how the data has been created (i.e. points to the LDW URL), the *prv:performedBy* indicates who has performed the wrapping process (i.e. rdf.onekin.org), finally the source data is described in the *prv:usedData* property.

So far, we have addressed YQL's *modus operandi* for readers to judge its adjustment to LDW development practices. Next subsection conducts a quality-in-use evaluation on using YQL for LDW. We want to measure how effective are ODTs to define LDW along the lines of the previous description.

[11] Full description at http://purl.org/net/provenance/ns#.

[12] Data provided by the API does not conform to *xsd:date*. For simplification sake, we do not include here its cast.

```
"@id":"http://rdf.onekin.org/lastfm/event/3986264",
"@type":"mo:Performance",
"dc:date":"Thu, 06 Nov 2014 20:00:00",
"rdfs:label":"Guerrera",
"event:place":"http://www.dbpedia.org/resource/Donostia",
"mo:performer":"http://rdf.onekin.org/musicbrainz/artist/Guerrera",
"dbpprop:hasPhotoCollection":"http://rdf.onekin.org/flickr/location/Donostia",
"wth:hasWeather":"http://rdf.onekin.org/weather/location/Donostia",
"void:inDataset":"http://rdf.onekin.org/lastfm/event/event",
"prv:createdBy":{ ⊟
  "prv:usedGuideline":{ ⊟
    "@type":"prv:CreationGuideline",
    "foaf:homepage":"https://raw.githubusercontent.com/onekin/ldw/master/events/lastfm.ever
  },
  "prv:completedAt":"2015-03-04T15:53:03",
  "@type":"prv:DataCreation",
  "prv:usedData":{ ⊟
    "@type":"prv:DataItem",
    "foaf:homepage":"http://ws.audioscrobbler.com"
  },
  "prv:performedBy":"http://rdf.onekin.org"
}
```


Fig. 6. Resource dereference with provenance. Namespaces omitted.

4.3 LDW Adaptive Maintenance

Adaptive maintenance refers to the modification of a software product performed after delivery to keep it usable in a changed or changing environment. For LDWs, this changing environment include Open APIs and the Linked-Data cloud.

API Upgrades. This is not only a LDW problem. Applications using third-party APIs are well-aware of these problems. For example, Zibran et al. [23] found that among 1,513 bug reports related to various components of Eclipse, GNOME, MySQL, Python 3.1, and Android projects, 562 bug-reports were related to API usability issues. Likewise, mobile apps suffer also from similar problems [13]. Linares-Vásquez et al. provide empirical evidence about the relation between the success of apps (in terms of user ratings), and the change- and fault-proneness of the underlying APIs. Similar problems will likely emerge for LDWs.

Overcoming this problem requires mechanisms for change detection and change propagation. YQL can help. First, YQL provides a one-shot view of the health of ODTs (see Figure 7) so that willing-full members of the community can quickly spot where their work is required. Second, change propagation can be better localized by splitting LDW development in two stages: from API-to-table and from table-to-rdf. The former refers to traditional ODTs. In the best scenario, ODTs can be already available (i.e. not being part of the wrapping effort), and hence, its maintenance can be conducted by a different set of developers. As for the table-to-rdf stage, it involves leveraging the ODT to deliver

Fig. 7. YQL's health checker. ODTs are monitored for problems. A color code is used to indicate the outcome together with an automatically-generated symptom message.

JSON-LD. This part is less likely to be affected as long as the API upgrade does not involve parameter removals. The ODT-as-usual should shelter the second part from changes on the format, name and order of API parameters.

Linked Open Data (LOD) Cloud Expansion. New data sources are continuously being added to the LOD cloud. One could expect this growth resulted into a more bushy cloud with a higher degree on interlinkage. However, this does not come across in a study about the dynamics of LD [11]. Käfer et al. observe that, unlike the HTML world with an estimate of 25% in the number of new hyperlinks in a week period, LD seemed much more static. The authors indicate that "this seems counter-intuitive in that LD itself is fundamentally comprised of URIs and thus links". In the same vein, a recent study about the LD cloud concludes that only 56% of the 1014 datasets studied have external links [19]. This might be partially due to most linked-data sources be in-house wrappers. Here, adding links to brand-new LOD nodes is taken by the few wrapper's developers.

Here, distinct benefits can be hypothesized from collaborative LDW development. First, the larger the number of people, the greater the chances of spotting an interesting LOD node. Also, more people tend to imply more heterogeneous data needs, hence increasing the pressure for interlinkages. Finally, the burden of adding new links can be shared with people other than the initial developers.

4.4 Measuring Effectiveness

ISO-9126 provides a framework to evaluate quality in use which includes effectiveness (i.e. the capability of the software product to enable users to achieve specified goals with accuracy and completeness) and productivity (i.e.the relation between the capability of the software product to enable users to expend

appropriate amounts of resources in relation to the effectiveness) [9]. A main indicator of effectiveness is the *"quality of solution"*, i.e. a measure of the outcome of the user's interaction with the system. As for productivity, indicators include task completion time and learning time. In this study, we use *"task completion time"* as the primary indicator of productivity.

Setting. In order to eliminate differences in the perception of LDW due to hardware or bandwidth differences, the study was conducted in a laboratory of the Computer Science Faculty of San Sebastián. All participants used computers with the same features (i.e., Intel Core 2 1.86 GHz, 3 GB RAM and Windows XP Professional SP3) and a clean installation of Firefox.

Subjects. The experiment was conducted among 15 graduate students applying in a Master in Web Engineering. The majority of participants were male (73.3%). Regarding age, 86.7% were in the 22-30 age range and all participants were below 35 years old. This experiment was realized at the end of 10 hours course in Web Programmable issues, where students familiarize with the YQL console, the YQL language and ODT specifications. As part of the Master degree, students followed a 30 hour Semantic Web course, where Linked Data concepts and RDF syntax were introduced. All of them were acquainted with JSON, but no JSON-LD. 5 students were expert JavaScript programmers, 5 had basic skills, and 5 knew the language but never code with it. All the students knew the existence of GitHub but only six had used it assiduously in previous projects.

Procedure. Before starting, a 45-minute talk was given, introducing the purpose, some practical examples of JSON-LD, one implemented LDW example[13] and the registration process on the LDW server. A user-guide sheet were distributed among participants with all this information. Next, subjects were faced with two scenarios, namely,

- from ODT to LDW. Here, students were given an existing ODT (i.e. *lastfm .events.getinfo*). The aim was to leverage this ODT to become a LDW. Tasks ahead include: URL pattern specification (i.e. lowering process) and lifting function definition. The latter involves ontologies identification, namespace handling, URI resource construction, URI class identification, properties XPath specification, multivalued attribute management and linkage pattern construction.
- from API to LDW. Here, students started from scratch, i.e. the API (in this case, the *authenticjobs.search* API[14]). This method returns the actual jobs that fulfill some input conditions. Besides the previously mentioned tasks, this scenario's demands include: API key obtention, API Endpoint localization, API parameter identification, and finally, ODT construction.

In order to measure productivity, participants had to annotate the start time and the finishing time. Finally, the subjects were directed to a *GoogleDocs* questionnaire to gather their opinion.

[13] https://github.com/onekin/ldw/blob/master/flickr/flickr.photos.getinfo.xml
[14] http://www.authenticjobs.com/api/documentation/

Table 1. Effectiveness. (a): from ODT to LDW. (b) from API to LDW

Task	#Students scenario (a)	#Students scenario (b)
API key obtention		12
API endpoint localization		15
API parameter localization		15
ODT construction		12
URL pattern specification	15	15
Ontologies identification	7	11
URI resource construction	13	12
Resource class identification	10	12
Properties XPath specification	15	11
Multivalue property management	2	5
Linkage pattern construction	11	12

Effectiveness Results. Table 1(a) shows the results for the first task: 13 out of 15 students completed the *LDW*. The criterium for success was the dereferenced of *Last.fm* events' URIs. During LDW development, none had problems to identify the *URL Pattern* that describes the lowering mapping. However, three had problems in specifying the <*function*> parameters that describe the lifting process. As expected, the *lifting* function caused most problems: all students lifted at least two attributes and created linked URI's to one resource; 13 correctly identified the URI of the resource (@id); 10 properly identified the type of the resource (@type: *mo:performance*); 7 provided appropriate namespaces (@context); finally, only 2 successfully processed multivalued attributes. The latter can be alleviated through a JavaScript library that helps managing multivalued attributes. Finally, interlinkage to other resources task was properly fulfilled by 11 students.

Table 1(b) depicts the outcome for the second endeavor: the development of the ODT plus the LDW. Compared with the first LDW, this task requires students to be familiarized with the *authenticjobs* API, identifying the required method and its input parameters. Additionally, students must register to *authenticjobs* to obtain the applications API Keys. Three students had problems to obtain this API keys. This API follows a standard REST query protocol similar to the *Last.fm* API, so students follow a clone-and-own approach by starting from the *flickr.photos.getinfo* ODT, and next, adapt it to the *authenticjobs'* specifics. This accounts for a new way of LDW development that collaborative development fosters. In the last step, that is, the ODT construction, 4 students had problems to identify the XPath where the result tuples were located. Once the ODT was created, moving to the LDW didn't involve any significant setback for most students (mainly due to the first lab being resolved some few hours before). Nevertheless, 3 students had problems to identify the ontology while 4 had difficulties to identify some complex XPaths from a service data (nested elements, attribute obtention, array position access). Once again, the main stumbling block stemmed from property multivalued attributes. Linkage

to other services accounted for 0 links (3 students), 1 link (7 students), 2 link (4 students) and 3 links (1 student).

Productivity Results. We appreciate a considerable dispersion on the time involved in LDW development. The first LDW involved 20' on average while the second took 50' on average. Spend time was proportional to the student's JavaScript experience.

5 The LDW Community: The Consumer Perspective

LDW continuous effort pays off if beneficiaries go beyond breakout developers. So far, most LDWs are seldom used outside their research projects. If LDWs are to evolve beyond proof-of-concept, scalability issues should be considered. Graceful degradation of elapsed times should be obtained to ensure appropriate quality of service. YQL can help by providing load balancing that outperforms small-scale attempts to host LDW services. This section evaluates two scenarios:

- LDW overhead, i.e. additional latency introduced by the wrapping w.r.t direct API access, and
- LDW load balancing gains, i.e. difference between running a LDW in a server with and without load balancing.

Both studies were conducted over a AMD Turion 64 X2 2 GHz CPU with 4GB of memory, with a domestic 6Mbps WIFI LAN bandwidth. Measurements were realized through JMeter[15]. The experiment pivots around the *Flickr* website. The goal was to dereference a URI that contains a position (i.e. http://$flickrservice/location/52.453056/13.290556/) together with photos at this position. The wrapper was implemented in two ways:

- as an ad-hoc program (i.e. ***Flickrwrappr***). This accounts for the traditional scenario, and it is based on a wrapper service provided by the University of Mannheim[16]. The implementation accounts for 250 lines of PHP code.
- as an YQL's ODT (i. e. ***FlickrODT***). Here, the wrapper was developed and deployed using YQL infrastructure (available at https://github.com/onekin/ldw/blob/master/flickr/flickr.photos.getinfo.xml).

5.1 URI-Dereferencing Latency

We want to measure the latency introduced by wrapping w.r.t. directly invoking the *Flickr*'s API. To this end, dereferencing was conducted 1000 times with one call per second. The experiment was repeated three times at different hours of

[15] http://jmeter.apache.org/

[16] http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/. Interesting enough, this wrapper stopped working on June, 2014 as a result of a change in *Flickr*'s API (refer to http://code.flickr.net/2014/04/30/flickr-api-going-ssl-only-on-june-27th-2014/). We upgrade the code and install it in our server.

the day. Table 2 (lefthand side) shows the results. Outcomes indicate that wrapping involves a three-fold overhead compared with direct API calling. In addition, *Flickrwrappr* benefits from directly invoking API whereas *FlickrODT* only accesses *Flickr* indirectly through YQL services. This indirection costs 125ms in the median. We can tentatively conclude that for sparsely used wrappers, ODT indirection might improve maintenance but introduces a time penalty.

Table 2. Latencies. Lefthand side: latency average values (ms). Righthand side: median latency values based on a number of threads (ms).

	Flickr	Flickrwrappr	FlickrODT
Mean	212	646	851
Median	194	601	726
Min	188	515	615
Max	300	990	1223

#threads	Flickr	Flickrwrappr	FlickrODT
10	202	605	726
50	204	611	739
250	210	1251	802
2000	215	2371	957

5.2 URI-Dereferencing Scalability

This second experiment looks at wrapper behavior with different loads. Here, we subject the wrappers to different dereferencing petition loads: 10, 50, 250 and 2000 threads. The process is repeated 10 times every 5 seconds. Table 2 (righthand side) depicts the results. Here, *FlickrODT* outperforms *Flickrwrappr* as a result of the load balancing performed by the YQL platform. Whereas *FlickrODT* performance gracefully degrades, *Flickrwrappr* surpasses 611ms as the median latency when handling over 50 threads in parallel. This behavior is most important to ensure quality of service on the Web of Data. For wrappers supported by data owners (e.g. *DBpedia*) this might not be a problem, since they enjoy the resources to meet these figures. However, third-party collaboratively developed wrappers require YQL-like infrastructure to thrive. Otherwise, their poor quality of service might well discourage other end-points to set interlinkage with them.

6 Related Work

This section frames YQL into other attempts to facilitate LDW development: *D2RQ* [1], *DBpedia* [12], the *SA-REST* platform [20] *Karma* [22], *SWEET* [14], *LIDS/LOS* services [18][21], *Virtuoso Sponger* [10]. These platforms are compared along the following dimension (see Table 3): the data-source being wrapped, the wrapper language, and the existence of tools. Next paragraphs describe the specifics of each approach.

Data Sources. There are several initiatives to wrap heterogeneous data sources to the Linked Data. *D2RQ* platform wraps relational databases, *DBpedia* wraps Wikipedia HTML pages, and *SA-REST* focuses on Web Services. But it is the

wrapping of REST API's where more initiatives showed up. This is inline with the tendency of publish data following this approach (a 75% of programmable web API offers this protocol). More encompassing approaches such as *Virtuoso Sponger*, offer a middleware for a variety data sources (relational, Web Service or REST). Unlike other approaches, data is not obtained on-the-fly but periodically uploaded. YQL permits wrappers upon REST services, and in general, any source that produces XML (e.g. XHTML).

Table 3. LDW Platforms

	Data sources	**Wrapper language**	**Tooling**
D2RQ	Relational DB	R2RML ontology	Code generator
Virtuoso Sponger	Relational DB and web resources	Virtuoso PL or C/C++ or Java	Clone
DBpedia	Wikipedia articles	WikiText template	Debug, clone
SA-REST platform	Web services	RDFa upon SAWSDL ontology	n/a
KARMA	REST	KARMA ontology	Programming by example
SWEET	REST	hREST upon MicroWSMO ontology	Annotation recommender
LIDS/LOS	REST	ontology + procedural	n/a
YQL	REST	YQL + JavaScript	Debug, clone cloud-deployment

Wrapper Language. Approaches attempt to find a compromise between expressiveness and learnability. This balance impacts the target audience. Domain-specific approaches focus on specific data sources (e.g. *Wikipedia* or relational databases) which permit lowering and lifting to be built-in. This accounts for more declarative wrapper specifications that easy user involvement. In the case of *DBpedia*, this specification is realized in terms of wiki templates, akin to the wiki origins of this initiative. In relational databases, wrapping is specified by R2RML ontology, where "TripleMaps" objects map tables and columns into RDF classes and properties, respectively. Departing from declarative specifications, other authors resort to general-purpose procedural languages (e.g. *YQL* and *Virtuoso Sponger*), wrapper ontologies (e.g. *Karma*), or a mixture (e.g. *SA-REST*, *SWEET* and *LIDS/LOS*), depending on the target audience (i.e. API programmers for *YQL* vs. Semantic Web community for *Karma*). Data services (e.g. *LIDS/LOS*) return data dynamically derived from supplied input parameters. The input can come in RDF format (Linked Open Services, *LOS*) or it can be a URI (Linked Data Services, LIDS). In both cases, the result is a RDF document. The wrapping is described using a RDF ontology together with some programming language (e.g. XSLT) for the lowering and the lifting. Finally, *Virtuoso Sponger* LDWs (known as Cartridges) resort to *Virtuoso PL*, a language

that somehow blends SQL and C, or it can be defined in some procedural languages (e.g. C/C++ of Java).

Tooling. Promoting LDW goes through providing appropriate tools. This includes the existence of publicly available LDW repositories that permit clone&own, code generators, assistive editing, testing and debugging capabilities as well as cloud deployment. *RBA* [16] is a tool for semi-automatically generating customised *R2RML* Mappings from a database. Both *DBpedia* and YQL enjoy a LDW repository which is realized through wiki pages and *GitHub* repositories, respectively. *SWEET* offers a ontology assisted annotation recommender based on *Watson* [7]. *Karma* illustrates the most ground-breaking stance. Through a kind of "programming-by-example" approach, *Karma* permits end users to generate LDW automatically out of a set of examples of API calls.

7 Conclusions

Linked Data Wrappers (LDWs) permit Open APIs to become part of the LD cloud. This effort can be facilitated through LDW frameworks. This paper explores the potential of YQL to become such platform with a first focus on effectiveness and scalability. Results are promising in scalability, and, to some extent, also in effectiveness. We illustrate how common LDW matters can be built on top of YQL's ODT, and how students manage to develop LDWs on an average time frame of 20' to 50', depending on whether they can tap into an existing ODT or not.

We hypothesize that the *aging* of YQL's LDWs will improve w.r.t. traditional in-house LDWs. Rationales are twofold. First, LDWs are available through GitHub so open to public scrutiny and collaboration, while ODTs a more declarative than their Java counterparts. Second, by caring about performance, the chances of increasing trust that will eventual end up in interlinkage, would increase. This in turn might well result in more pressure to keep the LDW up. This has yet to be proven. However, the benefits are worthwhile: keeping in sync Open APIs, LDWs and the linked-data cloud.

Acknowledgment. This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contract TIN2011-23839 (Scripting).

References

1. D2RQ. Accessing Relational Databases as Virtual RDF Graphs. http://d2rq.org/
2. Describing Linked Datasets with the VoID Vocabulary (2011). http://www.w3.org/TR/void/
3. A JSON-based Serialization for Linked Data. W3C Recommendation (2014). http://www.w3.org/TR/json-ld/
4. Becker, C., Bizer, C.: Flickr wrappr: precise photo association. http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/

5. Bizer, C., Cyganiak, R., Gauß, T.: The RDF book mashup: from web APIs to a web of data. In: Proc. of Scripting for the Semantic Web Workshop at the ESWC. CEUR Workshop Proceedings (2007), ISSN 1613–0073. CEUR-WS.org/Vol-248/paper4.pdf
6. Buneman, P., Khanna, S., Wang-Chiew, T.: Why and where: a characterization of data provenance. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 316–330. Springer, Heidelberg (2000)
7. d'Aquin, M., Motta, E., Sabou, M., Angeletou, S., Gridinoc, L., Lopez, V., Guidi, D.: Toward a new generation of semantic web applications. IEEE Intelligent Systems **23**(3), 20–28 (2008)
8. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. Tech. rep., W3C Recommendation (2012). http://www.w3.org/TR/r2rml/
9. Davis, I., Vitiello Jr, E.: ISO 9241–11. Ergonomic requirements for office work with visual displays terminals(VDTs) Part 11: Guidance on Usability (1998)
10. Erling, O., Mikhailov, I.: RDF support in the virtuoso DBMS. In: Pellegrini, T., Auer, S., Tochtermann, K., Schaffert, S. (eds.) Networked Knowledge - Networked Media. SCI, vol. 221, pp. 7–24. Springer, Heidelberg (2009)
11. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., Hogan, A.: Observing linked data dynamics. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 213–227. Springer, Heidelberg (2013)
12. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., et al.: DBpedia-a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web (2014)
13. Linares-Vásquez, M., Bavota, G., Bernal-Cárdenas, C., Penta, M.D., Oliveto, R., Poshyvanyk, D.: API change and fault proneness: a threat to the success of android apps. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 477–487. ACM (2013)
14. Maleshkova, M., Pedrinaci, C., Domingue, J.: Semantic annotation of web APIs with SWEET. In: 6th Workshop on Scripting and Development for the Semantic Web, Colocated with ESWC (2010)
15. Martin, D., Burstein, M., Hobbs, J.: OWL-S: Semantic Markup for Web Services. Tech. rep., W3C Member Submission (2004). http://www.w3.org/Submission/OWL-S/
16. Neto, L.E.T., Vidal, V.M.P., Casanova, M.A., Monteiro, J.M.: *R2RML by assertion*: a semi-automatic tool for generating customised R2RML mappings. In: Cimiano, P., Fernández, M., Lopez, V., Schlobach, S., Völker, J. (eds.) ESWC 2013. LNCS, vol. 7955, pp. 248–252. Springer, Heidelberg (2013)
17. Network, Y.D.: Yahoo Query Language (YQL) guide. https://developer.yahoo.com/yql/guide/
18. Norton, B., Krummenacher, R., Marte, A., Fensel, D.: Dynamic linked data via linked open services. In: Proceedings of the Workshop on Linked Data in the Future Internet at the Future Internet Assembly (2010)
19. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 245–260. Springer, Heidelberg (2014)
20. Sheth, A.P., Gomadam, K., Lathem, J.: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Computing **11**(6), 91–94 (2007)

21. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 170–184. Springer, Heidelberg (2011)
22. Taheriyan, M., Knoblock, C.A., Szekely, P., Ambite, J.L.: Rapidly integrating services into the linked data cloud. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 559–574. Springer, Heidelberg (2012)
23. Zibran, M.F., Eishita, F.Z., Roy, C.K.: Useful, but usable? factors affecting the usability of APIs. In: 2011 18th Working Conference on Reverse Engineering (WCRE), pp. 151–155. IEEE (2011)