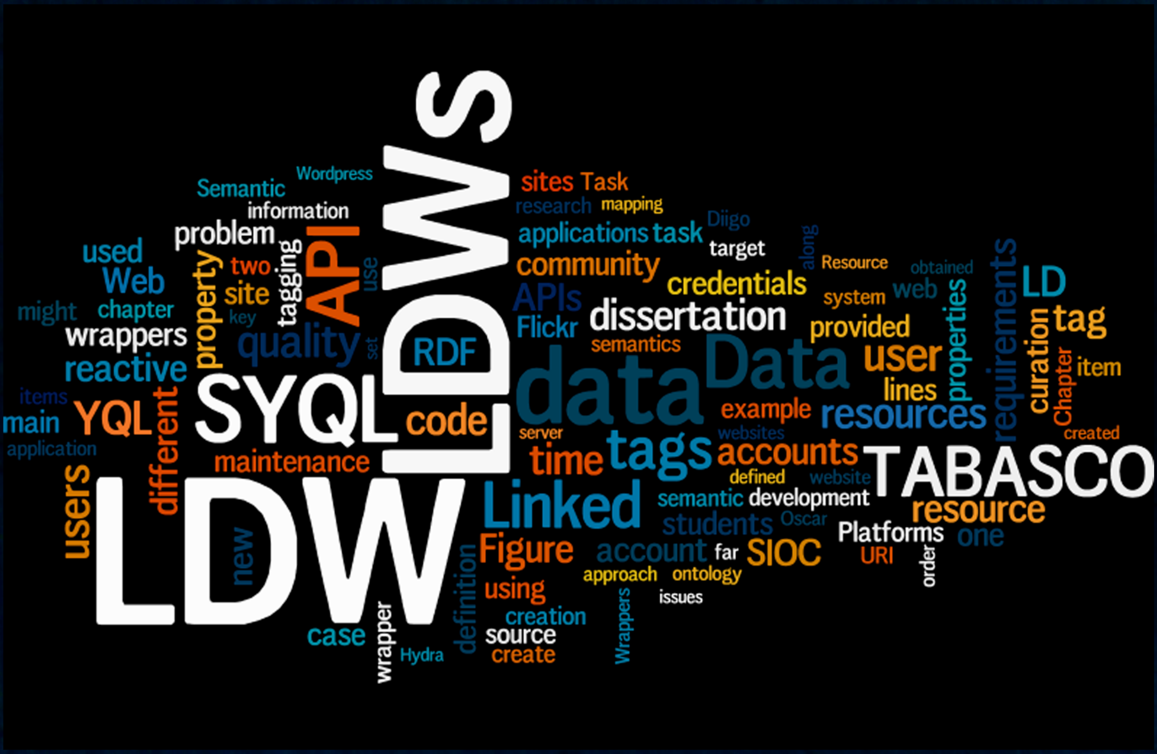


Ph.D. Thesis

Linked Data Wrapper Curation: A Platform Perspective

Iker Aitor Azpeitia Lakuntza

2017

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

onekin
Web Engineering Research Group
Supervisor: Oscar Díaz García
Supervisor: Juan Ignacio Iturrioz Sánchez

Linked Data Wrapper Curation: A Platform Perspective

Dissertation
presented to
the Department of Computer Languages and Systems of
the University of the Basque Country
in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

Iker Aitor Azpeitia Lakuntza

Supervisors: *Prof. Dr. Oscar Díaz García* and
Prof. Dr. Juan Ignacio Iturrioz Sánchez
San Sebastián, Spain, 2017

This work was hosted by the *University of the Basque Country* (Faculty of Computer Science).

Every living creature is happy when he fulfills his destiny, that is, when he realizes himself, when he is being that which in truth he is. For this reason, Schlegel, inverting the relationship between pleasure and destiny, said, “We have a genius for what we like.” Genius, man’s superlative gift for doing something, always carries a look of supreme pleasure.

– José Ortega y Gasset

Summary

Linked Data Wrappers (LDWs) turn Web APIs into RDF end-points, leveraging the LOD cloud with current data. This potential is frequently undervalued, regarding LDWs as mere by-products of larger endeavors, e.g. developing mashup applications. However, LDWs are mainly data-driven, not contaminated by application semantics, hence with an important potential for reuse. If LDWs could be decoupled from their breakout projects, this would increase the chances of LDWs becoming truly RDF end-points. But this vision is still under threat by LDW fragility upon API upgrades, and the risk of unmaintained LDWs. LDW curation might help. Similar to dataset curation, LDW curation aims to clean up datasets but, in this case, the dataset is implicitly described by the LDW definition, and "stains" are not limited to those related with the dataset quality but also include those related to the underlying API. This requires the existence of LDW Platforms that leverage existing code repositories with additional functionalities that cater for LDW definition, deployment and curation. This dissertation contributes to this vision through: (1) identifying a set of requirements for LDW Platforms; (2) instantiating these requirements in SYQL, a platform built upon Yahoo's YQL; (3) evaluating SYQL through a fully-developed proof of concept; and (4), validating the extent to which this approach facilitates LDW curation.

Resumen

El término “software wrapper” lo podríamos traducir como encapsulador, y se define como el software que encapsula otro software (o datos) para que pueda ser utilizado en un nuevo sistema. Los Linked Data Wrappers (LDW) convierten en datos semánticos RDF (Resource Description Framework) los datos que los sitios web exponen a través de sus APIs (Application Programming Interface). De este modo, los LDWs ofrecen en la nube de datos enlazados (Linked Open Data -LOD- cloud) información generada por los usuarios en los sitios web. El ejemplo más conocido es DBpedia, representación semántica en RDF de los datos de la Wikipedia.

Los LDW deben cumplir los cuatro principios del Linked Data (LD):

- nombrar los recursos en la web mediante URIs (Uniform Resource Identificator);
- utilizar URIs HTTP (HyperText Transfer Protocol) para consultar dichos identificadores;
- cuando alguien consulte una URI, ofrecer información útil mediante el uso de estándares; e
- incluir enlaces a otras URIs para descubrir nuevos recursos.

Para lograrlo, los LDWs deben convertir las consultas a URIs en llamadas a APIs web (lo que se conoce como lowering) para obtener datos RDF a partir de los datos devueltos por la API (conocido como lifting).

Por lo tanto, los LDWs son encapsuladores de datos y, por ello, no están contaminados con la lógica de la aplicación que los utiliza. Sin embargo,

la mayoría de LDWs están embebidos en las aplicaciones (por ejemplo, en mashups) limitando su potencial de reutilización. Si se segregaran de la aplicación aumentarían las posibilidades de convertirse en verdaderas fuentes de datos RDF. No obstante, la estrecha vinculación de los LDWs con la API web encapsulada lo dificulta. La continua evolución de las APIs y la falta de adaptación de los LDWs a estos cambios son una amenaza para su funcionamiento a largo plazo.

El mantenimiento de los LDWs ofrece una alternativa para su reutilización. Del mismo modo que se realiza mantenimiento de las Bases de Datos RDF, se puede optar por realizar mantenimiento de los LDWs. En el caso de los LDWs, la Base de Datos RDF está implícitamente definida en el propio LDW en tanto en cuanto encapsula una API web. A ello hay que añadir que los "problemas" de mantenimiento no se limitan únicamente a la calidad de los datos generados, también se extienden a aquellos derivados de la API. Es decir, las APIs evolucionan obligando a las aplicaciones que las utilizan a modificar su código. Estas dos vertientes del mantenimiento, a saber, la calidad de los datos generados y la evolución de las APIs, dificultan el mantenimiento y hacen los LDWs aún más frágiles. Todo ello obligaría a una mayor atención de los programadores encargados del mantenimiento. Sin embargo, los recursos destinados al mantenimiento suelen ser insuficientes, en especial, en los proyectos de investigación.

Teniendo en cuenta todo lo anterior, el mantenimiento de LDWs puede beneficiarse de plataformas similares a los repositorios de código fuente, eso sí, con nuevas funcionalidades que se ajusten a las necesidades de definición, despliegue y mantenimiento de LDWs. Esta tesis contribuye a esta visión: (1) identificando un conjunto de requisitos para las plataformas de LDWs; (2) plasmando estos requisitos en SYQL, una plataforma construida sobre Yahoo YQL; (3) evaluando SYQL en un caso de estudio; y (4), validando hasta qué punto SYQL facilita el mantenimiento de LDWs.

En torno a los LDWs se identifican tres actores: los creadores, los consumidores y los conservadores (los encargados del mantenimiento).

Los requisitos de las plataformas de LDWs se definen en base a las necesidades de los actores.

- Los creadores necesitan definir los LDWs y desplegarlos. Para definir LDWs los creadores deben lidiar con la heterogeneidad de las APIs en cuanto a protocolos de comunicación y formato de datos. Un lenguaje declarativo de definición de LDWs ofrece un buen equilibrio entre sencillez de definición y expresividad. Una vez definido, el LDW debe desplegarse, es decir, registrarse y ponerse en ejecución. Las plataformas comprueban la corrección sintáctica de los LDW y la posibilidad de ejecutarlos (por ejemplo, si tienen credenciales o no).

- Los consumidores deben encontrar (descubrir) los LDWs adecuados para sus proyectos, consultarlos para verificar su idoneidad y que estos cumplan con su función de consulta de recursos (URIs). La descripción semántica de LDWs mediante ontologías facilita el descubrimiento de los servicios ligados al LDW y permite valorar su idoneidad para cubrir las necesidades del consumidor. La descripción semántica de los LDWs suele hacerse habitualmente en términos de datos de entrada y datos de salida. Por lo tanto, los LDWs son también recursos Linked Data que pueden ser consultados. En última instancia, la función de los LDWs es la de ofrecer una vía de acceso semántica a los recursos ofrecidos por la API embebida.

- Los conservadores precisan ser conscientes de los problemas de mantenimiento de cada LDW y tener los medios para resolverlos. Para ello, deben ser alertados de errores o deficiencias que requieran una actualización del LDW. Una vez alertados, la plataforma debe ofrecer herramientas para resolver los problemas detectados.

La plataforma Semantic YQL (SYQL) implementa estos requisitos sobre Yahoo YQL. La elección de Yahoo YQL sobre la que construir una plataforma de LDWs no es baladí. Téngase en cuenta que Yahoo YQL ofrece un lenguaje susceptible de ser utilizado para definir LDWs, un motor de ejecución eficaz y una comunidad de programadores. Para atraer a los programadores de Yahoo YQL, se ha mantenido en lo posible su flujo de trabajo habitual. Además, se han realizado añadidos sobre Yahoo YQL que faciliten el mantenimiento sin necesidad de conocer el código interno de los LDWs. Para ello, se ha optado por la ingeniería inversa que presenta los LDWs como anotaciones semánticas de los datos devueltos por la API encapsulada. Por lo tanto, SYQL permite la definición, despliegue y mantenimiento de los LDWs. Como elemento distintivo se puede mencionar el testador de calidad o *Health Checker*. Un módulo que verifica la calidad de los LDWs siguiendo criterios de calidad establecidos en la literatura científica del área. La valoración de calidad se muestra en una página web para conocimiento de los conservadores.

Esta tesis doctoral también evalúa SYQL desde la perspectiva de los tres actores implicados. Un grupo de estudiantes ha actuado como productor de LDWs en dos escenarios. En el primer escenario, los estudiantes han producido un LDW desde cero, en el segundo, lo han creado a partir de un artefacto YQL previamente existente. Estos artefactos, llamados Open Data Table (ODT), permiten acceder a las APIs mediante el lenguaje de consultas Yahoo Query Language (YQL). En resumidas cuentas, un LDW no es más que un ODT extendido para describir el proceso de lowering y el de lifting. Los resultados de la evaluación indican que la mayor dificultad estriba en saber emparejar atributos devueltos por la API y su correspondiente propiedad semántica. Lógicamente, a mayor complejidad estructural de los datos de la API (por ejemplo, valores multivaluados) mayor dificultad para emparejarlos.

Para los consumidores, el criterio más relevante a la hora de hacer uso de los LDWs es el tiempo de respuesta (latencia) y cómo se degrada en escenarios de múltiples accessos concurrentes. Para cuantificarlo, se han simulado peticiones de acceso a la API en tres escenarios: (1) accediendo directamente a la API web, (2) accediendo a través de un wrapper externo, y (3) utilizando un LDW de SYQL. Las mediciones indican que, debido a las indirecciones, SYQL introduce mayor retraso en la latencia media que las otras dos alternativas. Pero sin embargo, hay que tener en cuenta que la latencia media es menor que un segundo, lo cual es asumible por muchas aplicaciones. Por otro lado, el motor de ejecución de Yahoo YQL balancea los múltiples accesos paralelos entre diferentes servidores. De este modo, la degradación en SYQL es imperceptible, mientras que el wrapper externo se degrada en exceso.

Por su parte, otro conjunto de alumnos actuando de conservadores ha evaluado las herramientas que SYQL les ofrece para mantener los LDWs. Concretamente, han actualizado LDWs mediante una herramienta de anotación que aplica la ingeniería inversa. Para ello han abordado cinco tareas de mantenimiento en diferentes escenarios: evolución de la API, actualización de la ontología, y cambios en el LOD cloud. Se han medido los tiempos en completar las tareas y se ha recopilado su satisfacción. La valoración ha sido positiva en general y los tiempos de realización de las tareas han sido relativamente bajos, teniendo en cuenta que los conservadores modifican LDWs que no han desarrollado ellos mismos.

Por lo tanto, los resultados de la evaluación son prometedores. Pero la idoneidad del enfoque planteado en esta tesis, es decir, la externalización de los LDWs a plataformas de LDWs para su mantenimiento por la comunidad, se verá confirmada cuando usuarios reales creen, consuman y mantengan este tipo de LDWs. Como primer paso en esa dirección ofrecemos los LDWs creados en esta tesis y el código fuente de SYQL para que puedan ser descargados, adaptados e instalados. Otras líneas de trabajo son la generalización de los LDWs a otros lenguajes y motores de ejecución (quizás PHP), la aplicación del Health Checker a otras fuentes de datos en el LOD cloud (por ejemplo, DBpedia) y el estudio de nuevos criterios de calidad (en concreto, la fiabilidad de los enlaces entre fuentes de datos).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Tell me to what you pay attention and I will tell you who you are."
– José Ortega y Gasset

1.1 Overview

This chapter introduces Linked Data (LD) wrapping related concepts, ideas and problems. Concretely, design science research is applied to analyze problems in web API wrapping for the Linked Data cloud. So, this chapter frames the Thesis and establish the context to understand the rest of the dissertation.

1.2 The Problem

Web APIs are an important source of current data. The importance of APIs for external data consumption should not be underestimated. According to a report by the Harvard Business Revue, *Salesforce* generates 50% of its revenue through APIs, *Expedia* generates 90%, and *eBay*, 60%, to name a few [IS15]. This explains the exponential growth in API figures [Wen17].

Unfortunately, less than 0.5% of APIs export their data using an RDF data format [DV17], being JSON-LD the preferable RDF format [SLK$^+$17]. This might be due to several circumstances: technical (i.e. mapping the underlying data representation to Linked Data formats might not be trivial), social (i.e. no demand on Linked Data representation by the service community) or financial (i.e. no clear business model). Fortunately, in case the data is available under a liberal license, producers can wrap these services to expose Linked Data. Indeed, in the 2017's Linked Open Data (LOD) cloud diagram [AMB$^+$] 36 datasets qualified as wrappers[1]. We focus on this kind of wrappers [BB, BCG07].

Commonly, LDWs are regarded as by-products of larger endeavors, e.g. developing a mashup application. Each application develops its own LDW, and its usage tends to be limited to this application. Hence, the LDW lifecycle is that of the containing application. The idiosyncratic and short-lived nature of some semantic applications might lead to abandon the application, and thus, leaving the LDW unmaintained. Indeed, it is not rare the case of LDWs that properly worked at the time they were launched, but they were no longer up at the time of this writing*: Flickr wrappr* [BB]*, GoogleArt project to RDF* [Gué11], *OAPI2LOD IATI parser* [Gdb12], *GeoNames* wrapper [SH10] or *Twitter wrapper* [Twi].

The problem is then not so much about LDW development but about unmaintained LDWs. This is unfortunate since it undermines the role of LDWs as a sustainable foundation for both the Web of Data and Semantic Applications. Causes may be many-fold: lack of interest, lack of recognition, lack of usage, lack of resources, etc. This dissertation addresses three main causes:

1. LDWs' lifecycles are coupled to those of the breakout projects. Once projects are over, so is the maintenance of the attached LDWs,

2. LDW maintenance penalty is high. This is mainly due to LDW

[1]Search conducted in `https://datahub.io` for the keyword "wrapper" in March, 2017.

fragility upon API upgrades,

3. the shortage of people involved. Traditionally, this is the case of research groups which might lack the resources for keeping LDWs up and running.

To lessen these causes, this dissertation resorts to *LDW curation*. Curation is not new to the LOD world. Evidences on LOD's mistakes and incompatibilities [HUH$^+$12] gave rise to the interest in data curation. More to the point, the fact that a dataset's own quality might impact the quality of other datasets that link to it, is being argued as "an incentive to clean stains in LOD that goes beyond that of the original dataset creators" [BRB$^+$14]. If this is so for explicit datasets, similar concerns can be risen from implicit datasets, i.e. LDWs. Different projects (e.g. *Virtuoso Sponger* [EM10] and *Bio2RDF* [CCTAD13]) resort to *GitHub* repositories for developers to clone LDWs; next, curate them in a different *GitHub* branch, and finally, send a pull request to modify the master distribution.

LDWs are code and hence, they can resort to general facilities for code artifacts, e.g. code repositories like *GitHub*. But, can we do better? After all, LDWs realize the definition of implicit datasets whose "stains" are not limited to those preventing the code from functioning but also those related with the quality of the dataset being obtained. Beyond general-purpose code repositories like *GitHub*, LDW-specific platforms could well cater for the specifics of LDWs. This includes LDW deployment but also supporting specific functionalities for LDW curation. Such platforms can act as repositories where LDWs can outlive their original applications, and most importantly, where third parties can tap into. Re-use increases the number of actors interested in keeping LDWs in shape, inspiring others to share LDW maintenance burden (i.e. the curators).

This dissertation contributes to this vision through: (1) identifying a set of requirements for LDW Platforms; (2) instantiating these requirements in SYQL, a platform built upon Yahoo's YQL; (3) evaluating SYQL through a fully-developed proof of concept; and (4), validating the extent to which this approach facilitates LDW curation.

1.3 Contributions

This dissertation addresses the unmaintained Linked Data Wrappers problem through an LDW Platform. The main contributions are:

- Set the LDW curation as a main task in the LDW lifecycle. This dissertation advocates for externalizing LDWs to increase their lifespan which contrast with ad-hoc and application built-in wrappers. This introduces new challenges, how to detach LDWs from applications and how to boost LDWs adaptation to the changing context in which they run.

- Characterize LDW Platforms. Three interdependent stakeholders involved in the LDW lifecycle are described: Producers, Consumers and Curators. Requirements for LDW Platforms from the stakeholders perspective are established: LDW definition and deployment; LDW discovery and lookup; Resource lookup; and, quality issues detection and solution.

- Instantiate an LDW Platform. A platform fulfilling requirements has been instantiated for Yahoo's YQL resulting on the Semantic YQL (hereafter SYQL /sɪlk/). LDWs for SYQL are based on the YQL language and engine in order to take advantage of the YQL programmers community.

- Evaluate the SYQL platform from the stakeholders perspective. In addition, SYQL suitability as data layer for a data-intensive Linked Data Application is checked. Both resource lookup and resource insertion has been checked to validate the LD read-write feasibility.

The wrappers and the SYQL source code developed for this dissertation are available on the Onekin Research Group *GitHub*. Concretely, the

LDW repository is at `https://github.com/onekin/ldw` and the SYQL platform is at `https://github.com/onekin/ldwServer`. A SYQL server is running at `http://rdf.onekin.org`. Here a demo video is available showing briefly how to create, deploy and curate LDWs.

1.4 Design Science Research Approach

This dissertation involves the development of different artifacts. However, this is a research project. Its purpose does not end with the development of the artifact. Rather the artifact serves to sustain (or rebate) a hypothesis through the evaluation of the artifact by the target audience. That is, besides the development of the artifact, other main research activities are involved. This requires the use of a "research methodology". This work will be handled using the “Design Science” methodology (see Figure 1.1). Below we will outline each of the Design Science task:

1. The Explicate Problem activity is about investigating and analyzing a practical problem.

2. The Define Requirements activity outlines a solution to the explicated problem in the form of an artifact and it elicits requirements, which can be seen as a transformation of the problem into demands on the proposed artifact.

3. The Design and Develop Artifact activity creates an artifact that addresses the explicated problem and fulfills the defined requirements. Designing an artifact includes determining its functionality as well as its structure.

4. The Demonstrate Artifact activity uses the developed artifact in an illustrative or real-life case, sometimes called a “proof of concept”, thereby proving the feasibility of the artifact. The demonstration will show that the artifact actually can solve an instance of the problem.

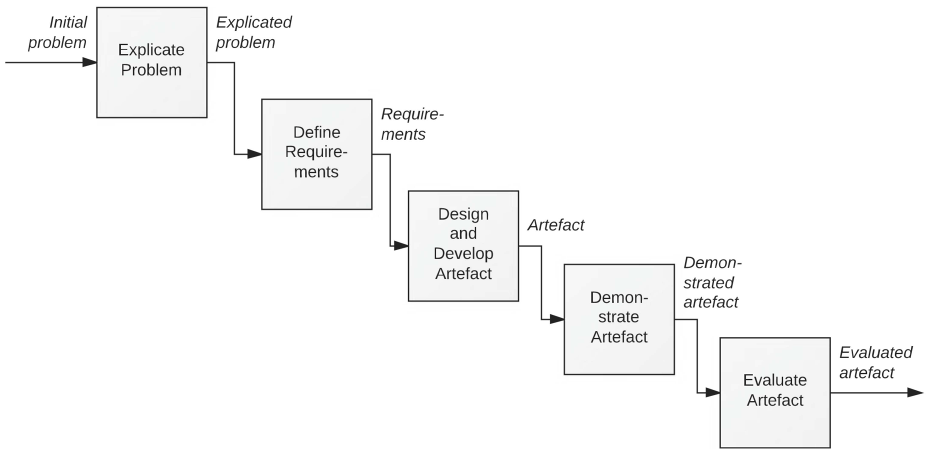


Figure 1.1: Overview of the method framework for design science [JP14]

5. Evaluate Artifact. The Evaluate Artifact activity determines how well the artifact fulfills the requirements and to what extent it can solve, or alleviate, the practical problem that motivated the research.

As indicated by P. Johannesson and E. Perjons [JP14], these tasks do not follow strictly in sequence. Rather, research is commonly iterative, moving back and forth between all the activities of problem explication, requirements definition, development, and evaluation. The arrows in Figure 1.1 should not be interpreted as temporal orderings but as input–output relationships. In other words, the activities should not be seen as temporally ordered but instead as logically related through input–output relationships.

1.5 Outline

This section summarizes the content of each chapter in this dissertation and aligns it with the Design Science steps (see Figure 1.2).

Chapter 1

This chapter introduces the main concepts on top of which this dissertation is built. LDWs are defined and the maintenance problem is identified following the DSR methodology.

Chapter 2

This chapter highlights the importance of using LDWs in Linked Data Applications along some use cases. In addition, a detailed analysis of LDW Platforms is performed.

Chapter 3

This chapter lists the objectives of the solution to address the LDW unmaintenance problem. That is, requirements for LDW Platforms in order to boost LDW curation. The involved stakeholders are recognized: Providers, Consumers and Curators.

Chapter 4

In this chapter an artifact that fulfills the requirements is shown. The SYQL platform supporting LDWs curation is described in detail.

Chapter 5

This chapter evaluates SYQL from Providers, Consumers and Curators point of view. In addition, it discusses differences and remarks of SYQL with respect to other LDW Platforms.

Chapter 6

In this chapter the writing operation in the LD cloud by LDWs is described. Besides reading resources, some LD Applications require to write resources in web sites. This is the case of our proof of concept.

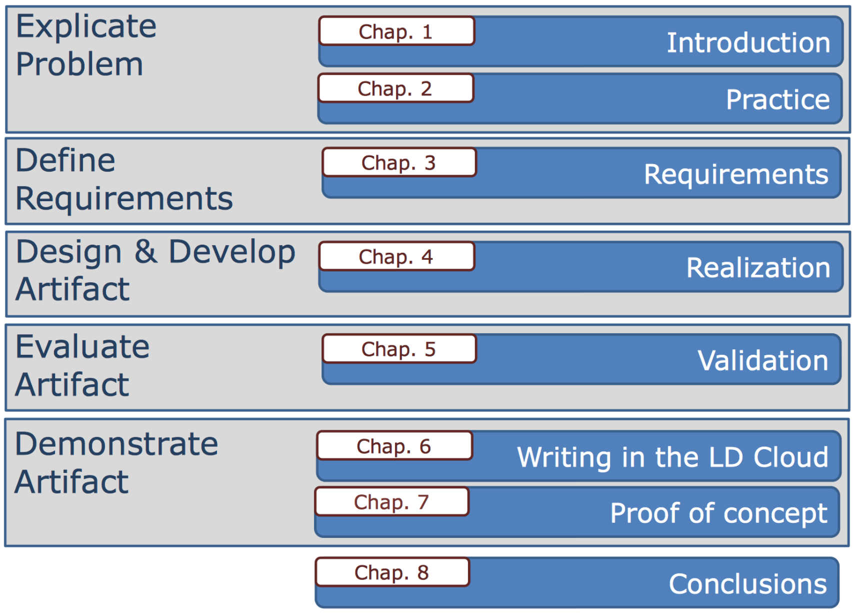


Figure 1.2: Chapter map

Chapter 7

A proof of concept demonstrates the suitability of SYQL as a data layer providing semantic resources through read-write LDWs. SYQL has to fulfill specifications of a Linked Data Application.

Chapter 8

This chapter concludes the dissertation. It summarizes the obtained results, makes an assessment and also identifies future research topics that this work raised.

1.6 Conclusion

The intention of this chapter was to give an overview of the contents of this dissertation. The topic was introduced and what, in our opinion, are its contributions were listed. The next chapter shows LDWs practice.

Chapter 2

Practice

"Scientific truth is characterized by its exactness and the certainty of its predictions. But these admirable qualities are contrived by science at the cost of remaining on a plane of secondary problems, leaving intact the ultimate and decisive questions. ... Yet science is but a small part of the human mind and organism. Where it stops, man does not stop."

– José Ortega y Gasset

2.1 Overview

This chapter delves into Linked Data wrapping practice and introduces some use cases. The aim is to highlight challenges and potential benefits of Linked Data Wrappers. Broadly, LDWs are mainly used in two scenarios: Web of Data and Semantic Applications. This section outlines the importance of LDWs in these two scenarios. Next, LDWs are analyzed as either coupled artifacts or separated artifacts.

2.2 Linked Data Wrappers

Software wrappers have been defined as "software that contains ('wraps around') other data or software, so that the contained elements can exist in the newer system" [Mag]. For our purposes, Linked Data Wrappers specializes previous definition where the wrapped content is a web API and the "newer system" is the Linked Data cloud. This could be achieved by supporting Linked Data Wrappers as REST-full services.

REST web services follow four basic design principles [Rod08]:

- use HTTP methods explicitly;
- be stateless;
- expose directory structure-like URIs; and
- transfer XML, JavaScript Object Notation (JSON), or both.

This design principles should be aligned with the four Linked Data 'rules' [BL06]:

- use URIs as names for things;
- use HTTP URIs so that people can look up those names;
- when someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL); and
- include links to other URIs, so that they can discover more things.

Although there is an obvious alignment and overlap between the approaches prescribed by REST and Linked Data, there are divergences in scope and applicability too [PDRM11]. A divergence refers to the supported HTTP methods. REST services offer the four methods: GET to retrieve a resource, POST to create a resource on the server, PUT to update a resource, and DELETE to remove a resource. In contrast, the majority of Linked Data sites are read-only. The Linked Data rules (only mentioning the

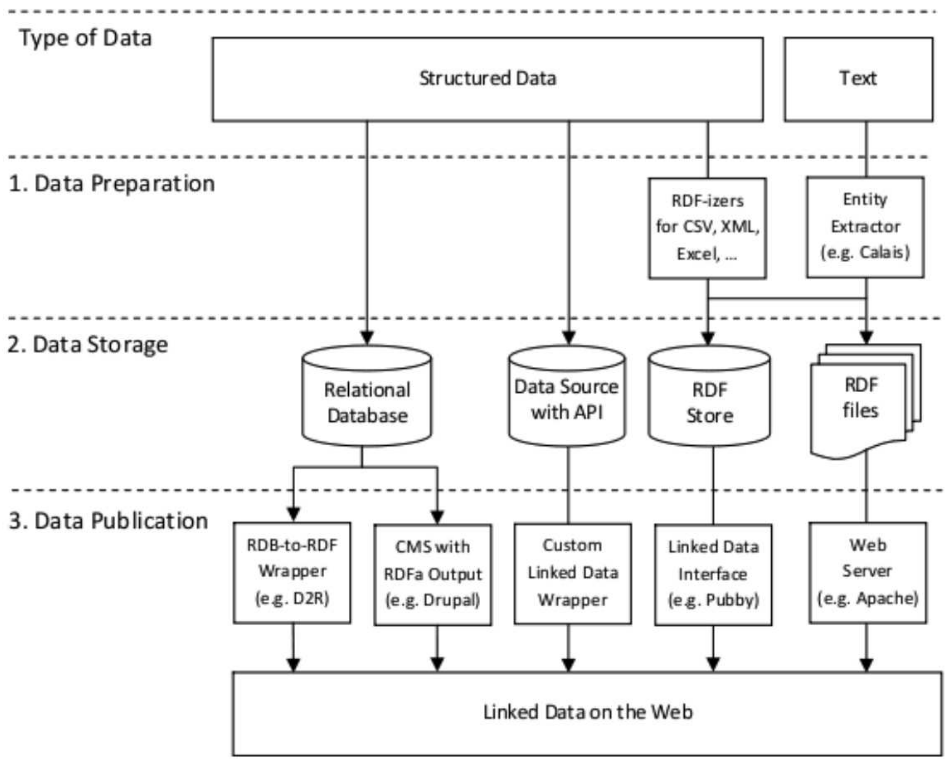

Type of Data
Structured Data
Text
1. Data Preparation
RDF-izers for CSV, XML, Excel, ...
Entity Extractor (e.g. Calais)
2. Data Storage
Relational Database
Data Source with API
RDF Store
RDF files
3. Data Publication
RDB-to-RDF Wrapper (e.g. D2R)
CMS with RDFa Output (e.g. Drupal)
Custom Linked Data Wrapper
Linked Data Interface (e.g. Pubby)
Web Server (e.g. Apache)
Linked Data on the Web

Figure 2.1: Linked Data publishing options and workflows [HB11]

lookup operation and forgetting write operations) and the Linked Open Data movement [BK11] (focusing on data publication) could create trend.

Therefore, wrappers on the web transform structured data into Linked Data (see Figure 2.1 taken from [HB11]). For example, RDB-to-RDF wrappers (e.g. D2RQ [CB]) are in charge of Relational Database tables and tuples transformation. For web APIs, custom Linked Data Wrappers are used. This dissertation focuses on LDWs wrapping web APIs.

2.3 The Practice: Linked-Data Wrapping

Web of Data. LDWs are being used to extent the LOD cloud with current data. We conducted a search upon `https://datahub.io` for the keyword “wrapper” in March, 2017: 36 datasets qualified as wrappers. But

LDW usefulness does not stop at introducing current data but also help to add “interlinkage layers” on top of existing LOD nodes. The need for interlinkage layers is evidenced by a 2014 study that concludes that only 56% of the 1014 LOD datasets studied have external links [SBP14]. In the same vein, 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 [KAU$^+$13]. The authors indicate that “this seems counter-intuitive in that LD itself is fundamentally comprised of URIs and thus, links”. More to the point, the steady introduction of new LOD nodes requires this interlinkage to be a continuous effort. Indeed, a 2016 survey about the quality of links between LD datasets, concludes that 7.9% of the links were actually dead [NKH$^+$16]. This sustains the need for continuously revising LOD interlinkage. LDWs can help by repairing/enhancing existing datasets with the broken/missing links.

Semantic Applications. They are grounded on the existence of quality datasets. Web APIs are a most important source of current data. Unfortunately, API providers (i.e. *eBay*, *Amazon*) lack a clear demand for RDF, while consumers stick to JSON/XML due to the learning curve and lifting effort to move to RDF. This chicken-and-egg “cold-start” problem could be mitigated if LDWs were in place. If API providers do not yet have a business case for leveraging their APIs to RDF, LDWs can temporarily take their place, providing the basis for semantic applications to thrive. Once semantic applications are available, this would make the case for API providers to take over, and natively provide RDF, making (some) LDWs redundant. The most recent LDW effort we are aware of is for the CrunchBase API [FMH17]. Authors acknowledged as a main “sword of Damocles” that of changes in the underlying APIs, though no solution is given except that of “monitoring the CrunchBase mailing list”.

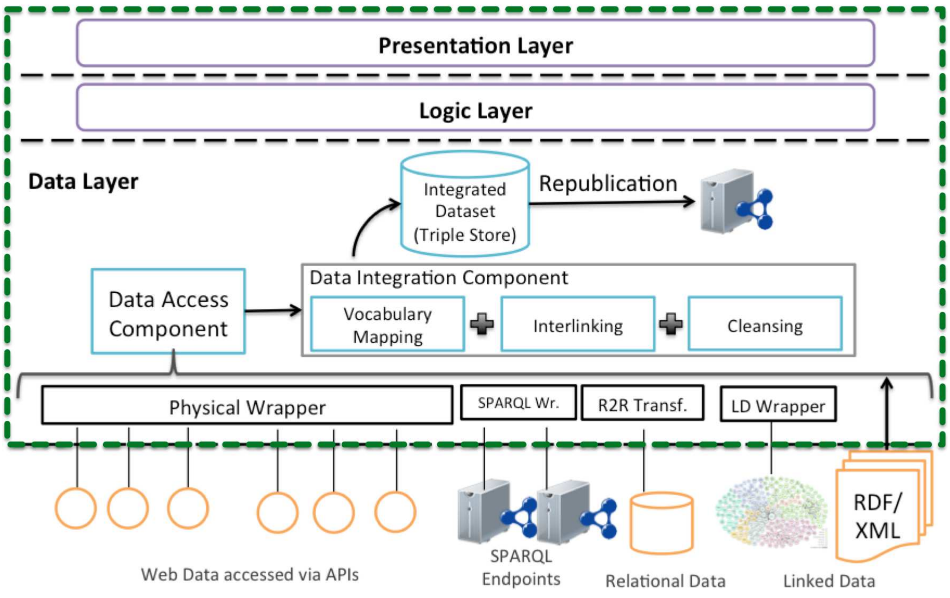
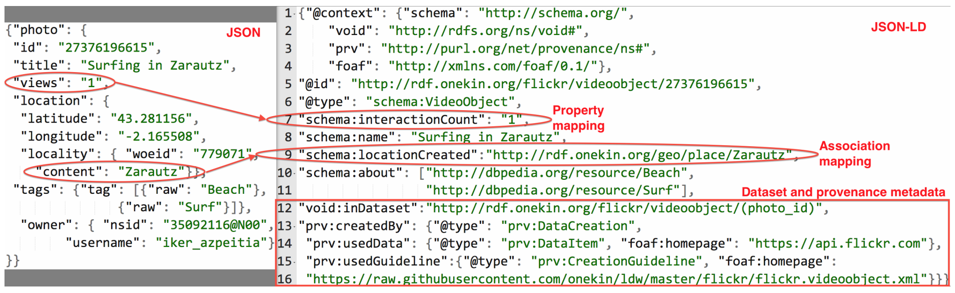


Figure 2.2: General architecture of LD Applications [SAD+14]

2.3.1 LDWs Embedded in Linked Data Applications

A general architecture of Linked Data Applications exhibits three layers: Presentation layer, Logic layer and Data layer (see Figure 2.2). The Data layer provides tools to expose traditional data sources in RDF data formats. They include wrappers for the databases and LDWs (aka RDFizers) for transforming data from other formats (e.g. XML, JSON and HTML) into RDF. Then, when all data is accessible as Linked Data, it might be stored in storages or accessed via web APIs such as SPARQL endpoints. These data might be manipulated and integrated to access in a refined form via a SPARQL query interface by application code in the Logic layer.

From a data consumption perspective, three main architectural patterns have been identified [SAD+14]. First, the *Crawling Pattern* where data is loaded in advance [GGL+14]. Second, the *Federated Query Pattern* in which complex queries are submitted to a fixed set of data sources [SHH+11]. And finally, the *On-The-Fly Dereferencing Pattern* where URIs are dereferenced at the moment that the application requires the data. This



```
{"photo": {
 "id": "27376196615",
 "title": "Surfing in Zarautz",
 "views": "1",
 "location": {
  "latitude": "43.281156",
  "longitude": "-2.165508",
  "locality": { "woeid": "779071",
    "content": "Zarautz"}},
 "tags": {"tag": [{"raw": "Beach"},
                 {"raw": "Surf"}]},
  "owner": { "nsid": "35092116@N00",
        "username": "iker_azpeitia"}
}}
```

```
{"@context": {"schema": "http://schema.org/",
    "void": "http://rdfs.org/ns/void#",
    "prv": "http://purl.org/net/provenance/ns#",
    "foaf": "http://xmlns.com/foaf/0.1/"},
"@id": "http://rdf.onekin.org/flickr/videoobject/27376196615",
"@type": "schema:VideoObject",
"schema:interactionCount": "1",
"schema:name": "Surfing in Zarautz",
"schema:locationCreated":"http://rdf.onekin.org/geo/place/Zarautz",
"schema:about": ["http://dbpedia.org/resource/Beach",
                 "http://dbpedia.org/resource/Surf"],
"void:inDataset":"http://rdf.onekin.org/flickr/videoobject/(photo_id)",
"prv:createdBy": {"@type": "prv:DataCreation",
 "prv:usedData": {"@type": "prv:DataItem", "foaf:homepage": "https://api.flickr.com"},
 "prv:usedGuideline":{"@type": "prv:CreationGuideline", "foaf:homepage":
 "https://raw.githubusercontent.com/onekin/ldw/master/flickr/flickr.videoobject.xml"}}}
```

Figure 2.3: From *Flickr* API output (left) to URI-addressable resource (right)

dissertation focuses on the last one. This pattern retrieves up to date data but performance is affected when the application must dereference many URIs. Therefore, this approach might not scale up when bulky data sets need to be retrieved[1] but it might fit scenarios where medium number of RDF resources need to be returned, frequently on demand. This is a common scenario when tapping into web APIs.

As an example, consider the *Flickr* API. This API facilitates programmatic access to pictures and videos [Fli]. Output formats include XML and JSON but not Linked Data. Figure 2.3 broadly describes the wrapping endeavor, i.e. moving from a JSON *document* in the left to a JSON-LD *resource* in the right. Notice that while a document is *retrieved*, a resource is *de-referenced*, i.e. resource content is obtained by dereferencing its URI. Hence, wrapping main tasks include [NKMF10, SH11, TKSA12]:

- lowering: i.e. mapping the URI (e.g. http://rdf.onekin.com/flickr/videoobject/{itemNumber}) to the corresponding API call (e.g. https://api.flickr.com/services/rest/?method=flickr.photos.getInfo &photo_id={itemNumber}).

[1]API producers enforce a request rate limit to prevent abuse of the service. If you exceed these thresholds, the API may stop working for you temporarily. Rates might be set on different basis: consumer-based (e.g. *Twitter* sets a maximum of 450 calls per 15'), resource-based (e.g. *Facebook* sets a maximum of 4800 calls per page and day for each active user) or operation-based (*Youtube* sets 1M calls per day for reads and 2000 for uploads).

- credentials handling: an API key is a code passed in by programs calling an API to identify the calling program, its producer, or its user to the website (e.g. *Flickr*). Normally, API keys serve to limit the number of times the API can be invoked in a certain period of time. For *Flickr,* this accounts for 3600 calls an hour. LDWs call APIs. Hence, they might require to first get an API key.

- lifting: creation of the Linked Data Resource from the API result (see the “property-mapping” arrow in Figure 2.3). Not all API data need to be exposed as Linked Data through a property mapping, and some semantic properties might be obtained from different API data as a calculation.

- interlinkage: most current APIs behave as data silos with no interlinkage with other resources. Hence, moving to the Linked Data cloud might require not only a change in the output format but also setting links with other URI-addressable related sources. Figure 2.3 illustrates this scenario (see the “association-mapping” arrow). The resource holds references to the video’s location (through the *schema:locationCreated* property) and the video’s topics on *DBpedia* (through the property *schema:about*).

- metadata (see the metadata box in Figure 2.3): this includes a link to the wrapper description (line 12), and provenance data (line 13-16).

The resulting code ends up being embedded in the Data layer of applications. Once applications are over, the interest in keeping up LDWs frequently vanishes.

2.3.2 LDWs as Separated Artifacts

LDWs have a value on their own right. The fact that they are mainly data-driven, increases their potential of reuse in different scenarios. Therefore, a broad literature exists on defining LDWs on their own: *LOD Laundromat*

Table 2.1: LDW approaches

	Data source	**Wrapper language**	**Creation time**	**Tooling**
LOD Laund.	RDF datasets	Hidden n/a	Load time	n/a
TWC LOGD	CSV	RDF	Load time	n/a
xCurator	Semistructured	Hidden n/a	Load time	Mapping tool
D2RQ	RDB	R2RML ontology	On the fly	Code generator
Virtuoso Sp.	RDB and REST	Procedural (C++, Java, ...)	Periodically	Clone
Bio2RDF	Semistructured, RDB and REST	Procedural (PHP, Java or Ruby)	Periodically	Clone
DBpedia	Wikipedia articles	WikiText template	Periodically	Debug, clone
SA-REST	Web services	RDFa upon SAWSDL ontology	On the fly	n/a
Karma	REST	*Karma* ontology	On the fly	PbE
SWEET	REST	hREST upon MicroWSMO ontology	On the fly	Recommender
LIDS/LOS	REST	Ontology and procedural	On the fly	n/a

[BRB+14], *TWC LOGD* [DLE+11], *xCurator* [YHM11], *D2RQ, Virtuoso Sponger, Bio2RDF, DBpedia* [LIJ+15], *SA-REST* [SGL07], *Karma* [TKSA12], *SWEET* [MPD10] and *LIDS/LOS* services [NKMF10, SH11]. This subsection compares these platforms along five dimensions: the data source being wrapped, the wrapper language, the creation time, tool availability, and finally, data curation support. First four dimensions are collected in Table 2.1 while curation support is displayed in Table 2.2.

Data Sources. There are several initiatives to wrap heterogeneous data sources to Linked Data. *D2RQ* wraps relational databases (RDB), *DBpedia* converts Wikipedia HTML pages, and *SA-REST* focuses on web services.

Table 2.2: Spotting and Cleaning activities during data curation

	Artifact	**Spotting**	**Cleaning**
LOD Laundromat	Datasets	Automatic	Automatic
TWC LOGD	Datasets	Consumers	Consumers
xCurator	LDW	Consumers	Administrators
Virtuoso Sponger	LDW	Developers	Developers
Bio2RDF	LDW	Developers	Developers

But it is the wrapping of REST API's where more initiatives showed up. More encompassing approaches such as *Virtuoso Sponger* or *Bio2RDF* offer a middleware for a variety of data sources (relational database, web service or REST).

Wrapper Language. *DBpedia* resorts to wiki templates, akin to the wiki origins of this initiative. In *D2RQ,* wrapping is specified through the R2RML ontology, where "TripleMaps" objects map relational databases' tables and columns into RDF classes and properties, respectively. Departing from declarative specifications, other authors resort to general-purpose procedural languages (e.g. *Bio2RDF* and *Virtuoso Sponger*), wrapper ontologies (e.g. *Karma*, *TWC LOGD*), or a mixture (e.g. *SA-REST*, *SWEET* and *LIDS/LOS*), depending on the target audience (i.e. the Semantic Web community for *Karma*).

Creation time. This dimension refers to the time the target RDF data is created from the data source. Broadly, this dimension is related with the obsolescence of the data source. For volatile data (e.g. REST data sources), RDF resources are created when they are requested on the fly. For more stable data (i.e. CSV, semistructured or RDF Dataset files), wrapping might happen at loading time. Finally, some platforms such as *Bio2RDF*, *Virtuoso Sponger* or *DBpedia*, allow for RDF data to be loaded periodically in search of a higher throughput.

Tooling. Promoting collaborative LDW development involves dedicated 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*

[NVCM13] is a tool for semi-automatically generating customized *R2RML* mappings from databases. *Virtuoso Sponger* and *Bio2RDF* resort to *GitHub* as the LDW repository. In contrast, *DBpedia* shares wrappers as wiki pages. *SWEET* offers an ontology-assisted annotation recommender based on *Watson* [dMS$^+$08]. *Karma* resorts to “programming-by-example” (PbE) where users generate LDWs out of a set of examples of API calls.

Curation. Table 2.2 outlines main projects in the LOD area where curation is being addressed. For the purposes of this dissertation, the main insights come from who conducts the curation, specifically, who conducts two of its main tasks: detection (i.e. spotting *the stain*) and intervention (i.e. cleaning *the stain*). Ideally, both tasks should be automated. Unfortunately, curation is not fully automated for most data types, requiring user intervention. Here, the user can be limited to the platform administrator or extended to any consumer. The amplitude of the curator spectrum very much depends on the complexity of the dataset or the LDW at hand, but also on the richness of *the stains* to be spotted. For instance, *LOD Laundromat* is a curation service for RDF datasets. Being an automatic curation service, it detects and repairs only a fixed number of issues. Alternatively, *xCurator* allows for consumers to spot *stain*s that are next handled by system administrators. In the same vein, *TWC LOGD* allows for consumers to generate personal versions of datasets as needed. If we move to the LDW realm, both *Virtuoso Sponger* and *Bio2RDF* resort to *GitHub* repositories for developers to clone LDWs; next, curate them in a different *GitHub* branch, and finally, send a pull request to modify the master distribution.

LDWs are code and hence, they can resort to general code repositories like *GitHub*. This work advocates for dedicated platforms that leverage existing repositories to account for LDW definition, deployment and curation.

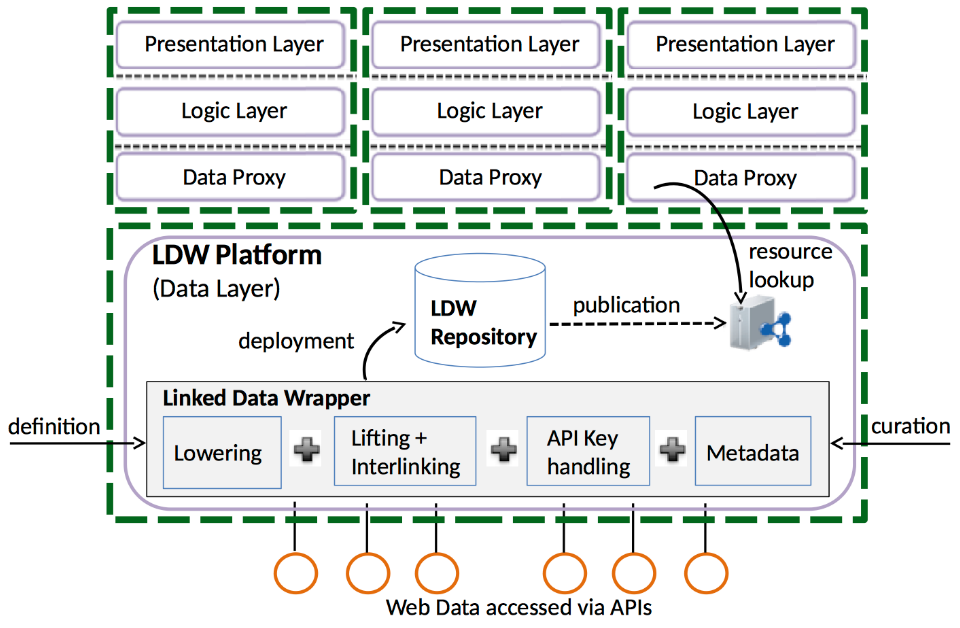


Figure 2.4: Externalizing LDW concerns into a dedicated platform

2.4 The Case for LD Applications

The vision is for LDW concerns to be externalized into a separated platform (see Figure 2.4). Implications are many-fold:

- at development time, LDWs are *specified* at the LDW Platform. Being a specialized platform, utilities can be offered to speed-up both specification and deployment. From the application's perspective, API resources are now accessed as native RDF resources. From the developers' perspective, LDWs are specified outside the application boundaries but in the LDW Platform.

- at maintenance time, LDWs are *curated* at the LDW Platform. Facilities should be provided for visualizing the current functioning status of LDWs, and to spot (and amend) eventual malfunctions. Curators might or might not coincide with the original LDW developers, so code understandability becomes a critical feature.

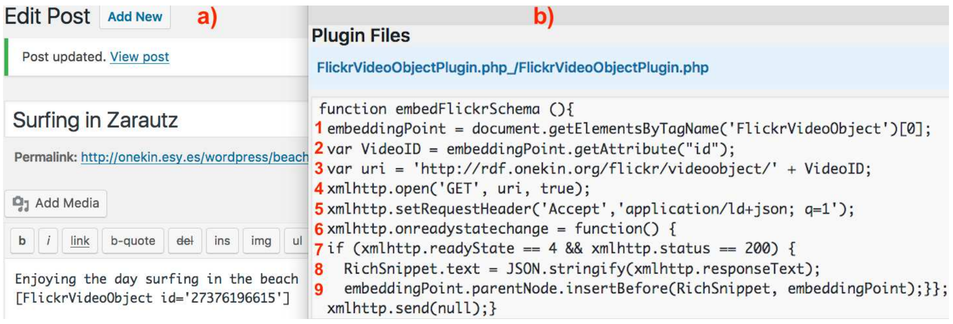
- at runtime, LDWs are *enacted* at the time resources are looked up or created. From the application's perspective, no difference should exists between static RDF resources, and RDF resources dynamically assembled.

This section introduces different usage scenarios, namely, annotation plug-ins, cross publishing tools, RDF visualizers and semantic mashups. Specifically, this dissertation uses ***WordPress*** [Wor], ***TABASCO*** [IDA11b], ***LOD-milla*** [lod] and ***LinkedWidgets*** [lin] as representatives of Content Management Systems (CMSs), Task Automation Services (TAS), RDF graph visualizers and mashup platforms, respectively. For each platform, an application is developed where LDW needs are externalized. In this way, other developers can tap into existing LDWs.

2.4.1 CMS Platforms: WordPress

A CMS is a computer application that supports the creation and modification of digital content using a common user interface. *WordPress* is a popular open-source CMS. Here, content owners care about their pages ranking high in search engines. Recently, Google, Yahoo and Bing join forces to provide the *schema.org* ontology in order to mark up structured data in the Web [GBM16]. Search Engines consume so annotated content and show it in a flashy way. The term "Rich Snippet" was coined by Google to refer to those *schema.org* formatted samples of a site's content. Once Web content is marked up along the snippet directives, search engines can offer a more detailed account of web sites, making it more enticing for users to click on, and easier for Search Engines to extract information [Sim11]. The importance of Rich Snippets is highlighted by the fact that WordPress offers over two hundred plug-ins for WordPress sites to be annotated this way. Bloggers are provided with a snippet editor that inlays the corresponding Rich Snippet in the blog page when referring to let's say, people or organizations.

So far, bloggers are prompted to introduce this metadata manually.



Surfing in Zarautz

Permalink: http://onekin.esy.es/wordpress/beach

Enjoying the day surfing in the beach [FlickrVideoObject id='27376196615']

FlickrVideoObjectPlugin.php_/FlickrVideoObjectPlugin.php

```
function embedFlickrSchema (){
embeddingPoint = document.getElementsByTagName('FlickrVideoObject')[0];
var VideoID = embeddingPoint.getAttribute("id");
var uri = 'http://rdf.onekin.org/flickr/videoobject/' + VideoID;
xmlhttp.open('GET', uri, true);
xmlhttp.setRequestHeader('Accept','application/ld+json; q=1');
xmlhttp.onreadystatechange = function() {
if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
  RichSnippet.text = JSON.stringify(xmlhttp.responseText);
  embeddingPoint.parentNode.insertBefore(RichSnippet, embeddingPoint);}};
xmlhttp.send(null);}
```

Figure 2.5: *WordPress* editors: (a) Post editor and (b) plug-in editor

```
{"@context": {"schema": "http://schema.org/"},
  "@id": "http://rdf.onekin.org/flickr/videoobject/27376196615",
  "@type": "schema:VideoObject",
  "schema:name": "Surfing in Zarautz",
  "schema:description": "Surfing in Zarautz is like a golden day of sun on the beach.",
  "schema:thumbnailUrl": "https://c1.staticflickr.com/8/7411/27376196615_d83fddf183",
  "schema:uploadDate": "2016-05-31 08:14:23",
  "schema:duration": "16",
  "schema:contentUrl": "https://www.flickr.com/photos/35092116@N00/27376196615/",
  "schema:interactionCount": "1"
}
```

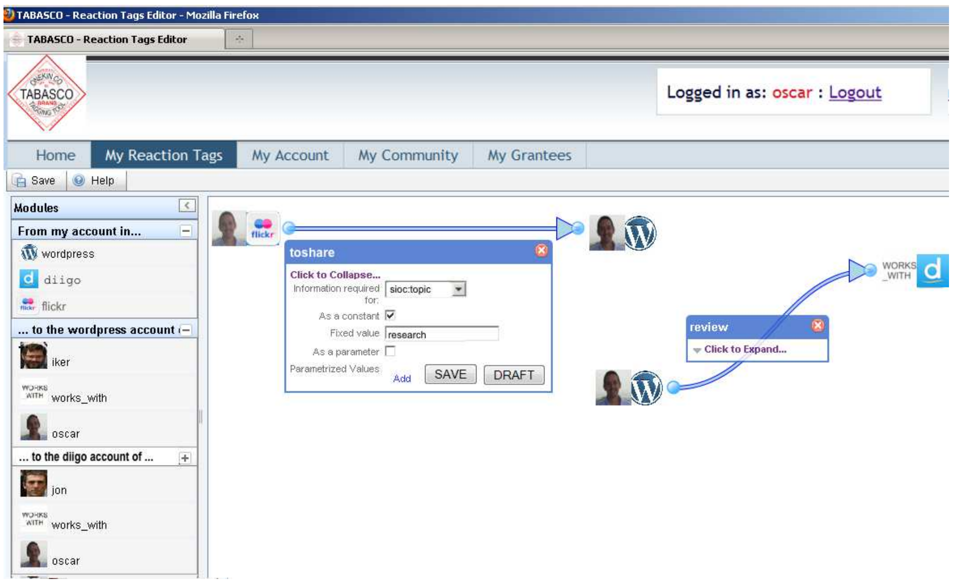
Figure 2.6: A schema:VideoObject Rich Snippet

However, it is not rare for this metadata already be available via an API. In this case, it is possible to develop a plug-in that obtains this information automatically from the website API rather than prompting the user. Figure 2.5 (a) provides an example. A new post is being edited that inlays a video taken from *Flickr* (e.g. the video with ID ’27376196615’). The plug-in provides annotation mark-up (e.g. *[FlickrVideoObject id=videoID])* for obtaining the Rich Snippet out of an API call to *Flickr*. Not only does this alleviate bloggers from introducing the metadata manually, but also avoids mismatches between the metadata provided by bloggers and the metadata already available through APIs. Unfortunately, these APIs rarely provide their output in JSON-LD. Therefore, the WordPress plug-in needs to handle the API call as well as the mapping from the API format to Rich Snippet JSON-LD. Rather than embedding this wrapping functionality as part of the plug-in, this dissertation advocates for this functionality to be detached into an LDW Platform from where it can be re-used. Provided this is the case, our sample plug-in is reduced to the snippet in Figure 2.5 (b): the code requests the *flickr.videoobject* LDW service available at `https://github.com/onekin/ldw/blob/master/flickr/flickr.videoobject.xml`. The video ID is scrapped from the post rendering (line 2); the LDW URI is constructed (e.g. *'http://rdf.onekin.org/flickr/videoobject/' + video ID)* (line 3)*;* finally, the URI is dereferenced (lines 4-6), and the Rich Snippet is obtained (line 8). Figure 2.6 depicts the embedded Rich Snippet. The WordPress plugin is available at `https://github.com/onekin/ldw/blob/master/Flickr.WPplugin.php`.

2.4.2 Task Automation Services: TABASCO

Cross publishing is described as "information posted on one site is published automatically to another" [LGdSN10]. Such information is materialized as resources whose type is dependent of the container site: blog posts in weblogs, bookmarks in bookmarking sites, wiki-articles in wikis, and so forth [BBFD08]. Users perform resource republishing while they are browsing the web or, alternatively, they define cross publishing rules in advance. As an example of the former, the well known *Facebook Like* button [Fac] is viewed across at least 2 million websites daily [Dat]. For the latter, Task Automation Services (TAS) (e.g. *IFTTT* [IFT] and *Zapier* [Zap]) support users without any formal programming skills on defining cross publishing trigger-action rules. In doing so, the platform monitors the source site to detect the triggering condition and, if detected, to execute the action. The action creates a resource in the target site from the resource in the source site [Ova14].

Semantic resources simplify rules definition and resource derivation due to understanding properties' meaning is easier. TABASCO (TAg-BASed inter-site COmmunication) is a case in point. It is a TAS performing user tasks coded in tags. This is accomplished by Event-Condition-Action (ECA) rules where the event rises if a resource contains a "re-



Information required for: sioc:topic
As a constant ☑
Fixed value research
As a parameter ☐

Figure 2.7: Reactive Tag running example:*“toshare”*

active tag”. For example, Oscar labels research videos in Flickr with the “toshare” tag to remember he wants to republish these videos in his Wordpress blog. In order to automatize video publication, he defines this rule: *“on tagging toshare at Oscar’s Flickr, do create a post on Oscar’s Wordpress into the research category”* (see Figure 2.7). To achieve this, TABASCO monitors Oscar’s Flickr account to detect new resources labeled with the *toshare* keyword. If a new video is detected then TABASCO derives a new semantic resource from the source resource and publishes it in the Oscar’s blog.

TABASCO can reuse the *flickr.videoobject* LDW to retrieve videos and check tags out. However, *schema:VideoObject* resources do not hold tags yet Flickr API provides tags, hence, TABASCO administrators upgrade the *flickr.videoobject* LDW adding the *sioc:topic* property. Besides upgrading and reusing the *flickr.videoobject* LDW, two new LDWs are created. The first one is the *flickr.person* LDW to retrieve users’ video lists to be monitored (available at `https://raw.githubusercontent.com/onekin/ldw/master/flickr/flickr.person.xml`). The second one is an LDW for the sake of post creation in WordPress (*wordpress.weblog* available at `https://raw.githubusercontent.com/onekin/ldw/master/wordpress/wordpress.weblog.xml`).

It is worth to note differences between this scenario and the previous one. In the CMS Platform scenario, a VideoObject Rich Snippet is embedded into an arbitrary post whilst in the TAS scenario a blog post about the video is created.

2.4.3 RDF Graph Visualizers: LODmilla

Linked Data exploration is being supported through different visualization tools [DP17, DR11]. An example is *LODmilla*. It permits navigate and explore multiple LOD datasets, save LOD views and share them with other users. For the purposes of this dissertation, the point to note is that *LODmilla* allows for links to be navigated dynamically, exploring the LOD in a personal way.

The question is to extend the exploration out of the existing LOD. There exists plenty of APIs out there to tap into. The current LOD can be idiosyncratically extended with dynamic resources obtained through API calls. All it is needed is the existence of LDWs that permit to close the chasm between API native format and JSON-LD, including interlinkage with existing LOD sources. Let's take an example. We can initiate the exploration at a given *Flickr* user, and thereupon retrieve his videos. This is straightforward with the previously developed LDWs. In addition, we can interlinkage *schema:VideoObject* resources to other resources either LOD-based (e.g. *DBpedia* resources) or API-obtained (e.g. *GeoPlanet* resources). Figure 2.8 depicts how this exploration looks like at *LODmilla* merging LOD-sourced data (e.g. *DBpedia*) and API-sourced data (e.g. *Flickr, GeoPlanet* [Neta] *&* *Wunderground* [Wea]) in the same graph through LDWs.

The exploration starts at a given *Flickr* person (35092116@N00 node in Figure 2.8) through the *flickr.person* LDW created in the TAS scenario.

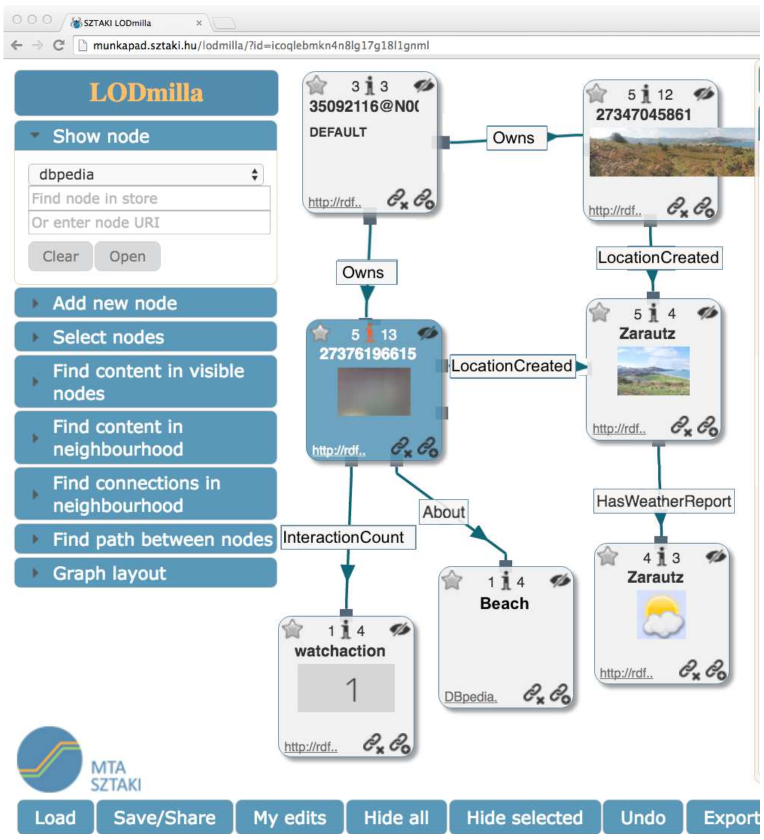
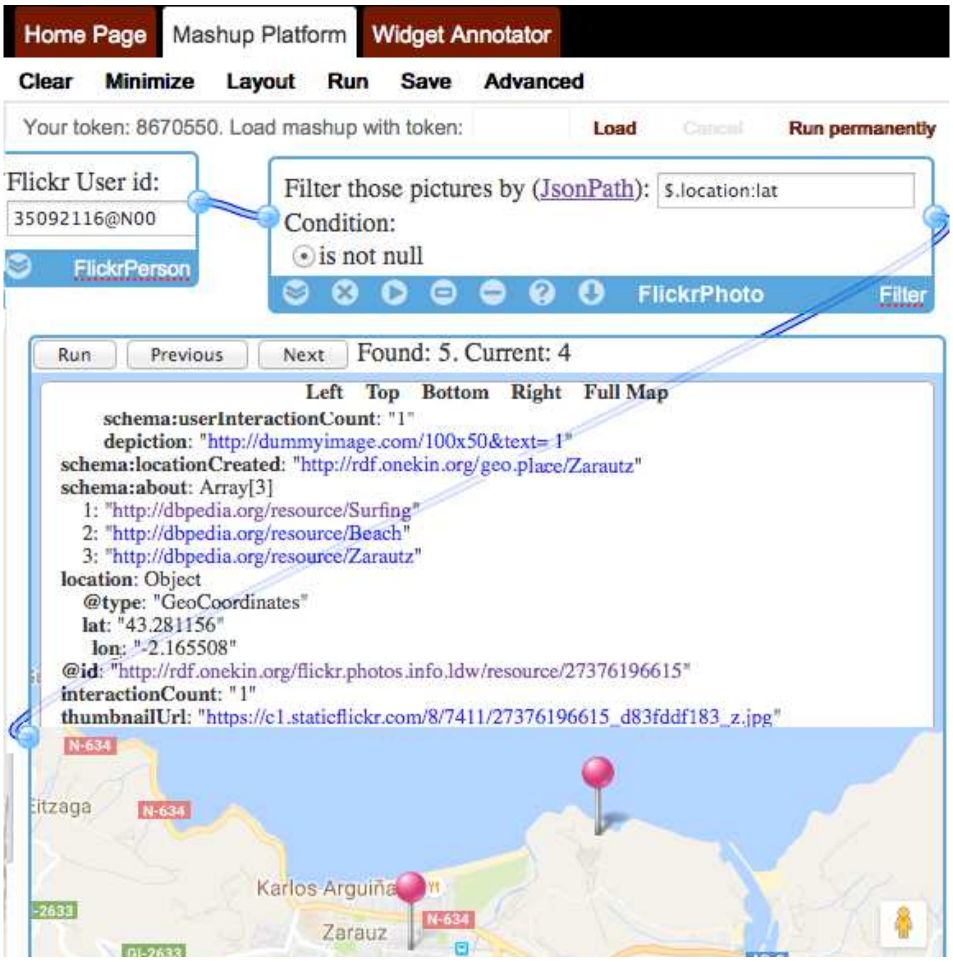


Figure 2.8: *LODmilla* exploration graph

Videos are next explored (through the *flickr.videoobject* LDW). Using *LODmilla* facilities, users decide which video properties to show up: *schema:interactionCount* (i.e the number of interactions for the video), *schema:about* (i.e. the subject matter of the video), etc. This permits to keep exploring on the basis of these resources. Specifically, *schema: locationCreated* resources hold places information and from places the weather reports are retrieved. This example requires an LDW for turning *GeoPlanet* API into a data set. Notice that the *flickr.videoobject* LDW needs to be upgraded adding two links *schema:about* and *schema: locationCreated* to point to *DBpedia* and *GeoPlanet* resources, respectively.

The bottom line is that LDWs permit to combine in the very same graph



Your token: 8670550. Load mashup with token:

Flickr User id: 35092116@N00

Filter those pictures by (JsonPath): $.location:lat
Condition:
(•) is not null

Found: 5. Current: 4

```
    schema:userInteractionCount: "1"
    depiction: "http://dummyimage.com/100x50&text= 1"
schema:locationCreated: "http://rdf.onekin.org/geo.place/Zarautz"
schema:about: Array[3]
  1: "http://dbpedia.org/resource/Surfing"
  2: "http://dbpedia.org/resource/Beach"
  3: "http://dbpedia.org/resource/Zarautz"
location: Object
  @type: "GeoCoordinates"
  lat: "43.281156"
   lon: "-2.165508"
@id: "http://rdf.onekin.org/flickr.photos.info.ldw/resource/27376196615"
interactionCount: "1"
thumbnailUrl: "https://c1.staticflickr.com/8/7411/27376196615_d83fddf183_z.jpg"
```

Figure 2.9: *LinkedWidgets* mashup

LOD-sourced resources and API-sourced resources, hence introducing a dynamicity that it is seldom obtained using LOD alone. In this way, *LOD-milla* users can save the exploration to be next run periodically, and observe how dynamic data (e.g. interaction counters, weather forecast) changes.

2.4.4 Semantic Mashups: LinkedWidgets

Semantic mashups are mashup applications using RDF as its background data model, and SPARQL for tasks execution [KK15]. These applications offer new functionality by combining, aggregating, and transforming data

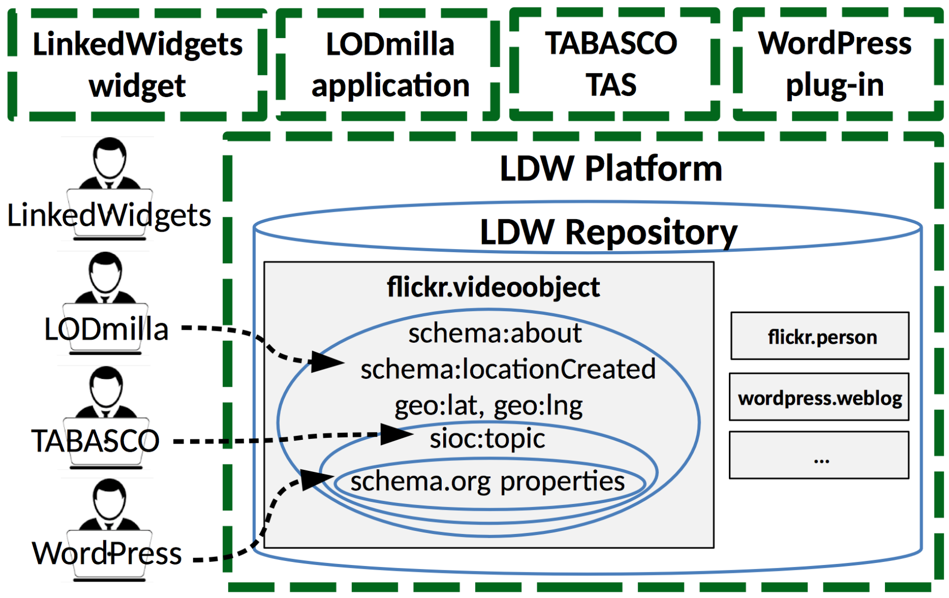


Figure 2.10: The vision: LDWs are created, upgraded and used by the community

available on the Web of Data. The benefits brought by semantic technologies w.r.t traditional Web mashups, is the use of the RDF data model as the unified data model for combining data from heterogeneous data resources. Different tools have been proposed to empower end-users to create mashups [GS10, LLSL16]. *LinkedWidgets* is a case in point.

In *LinkedWidgets*, mashups are modeled as widgets that are orchestrated using a pipe-like approach. Figure 2.9 depicts a *LinkedWidget* mashup that involves arranging somebody's *Flickr* videos into Google Maps. It looks like Yahoo Pipes but the novelty comes for the underlying data exchange technology: JSON-LD. This adds a semantic layer to the data and makes it machine readable.

Though JSON-LD certainly improves interoperability, the low number of APIs offering this format requires a wrapping effort. This is the case for our sample problem. *Flickr* APIs need to be consulted to obtain the person's videos (*flickr.person* LDW*)* and the video metadata (*flickr.videoobject* LDW*)* to be later displayed in the Google map. This wrapping effort might put some users off. Here, the notion of LDW-as-a-service can help. Specifically, the *flickr.videoobject* and the *flickr.person* LDWs could have well be made available as a result of the previous use cases. If so, *LinkedWidgets* developers can tap into these LDWs when creating their widgets.

Figure 2.10 summarizes this dissertation's vision: LDWs are created (*WordPress* scenario), upgraded (*LODmilla* scenario) and used (*Linked-Widgets* scenario) by the community as developers confront wrapping needs in distinct scenarios. Developer needs could provoke to create, upgrade and use LDWs in the same scenario (e.g. TABASCO scenario).

2.5 Conclusion

The issue: LDWs' lifecycles are coupled to those of the breakout applications. To lessen this problem, this dissertation introduces a new artifact: the LDW Platform. Unlike project embedded wrappers, separated wrappers promote reusability. LDWs are created, upgraded and used by the community as developers confront wrapping needs in distinct scenarios.

Chapter 3

Requirements

"Every intellectual effort sets us apart from the commonplace, and leads us by hidden and difficult paths to secluded spots where we find ourselves amid unaccustomed thoughts."

– *José Ortega y Gasset*

3.1 Overview

LDW Platforms aim at becoming single-stop solutions for LDW management. Specifically, LDW Platforms should account for three main stakeholders: producers (i.e. those who develop LDWs from scratch), consumers (i.e. those who re-use someone else's LDWs) and curators (i.e. those who perform some kind of LDW upgrading). The rest of this chapter identifies requirements for each stakeholder.

3.2 Requirements for LDW Platforms

3.2.1 Producer Requirements

Allow for LDW Definition

Mechanisms should be provided to address the specifics of LDW development such as lowering, lifting, or interlinkage (see Section 4.3.1). Platforms offer a possibility of abstracting developers from the heterogeneity of API requests and its optimization, making LDW definition more declarative, and hence, more effective.

LDW definition admits different compromises between expressiveness and learnability. 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 LDW specifications that ease user involvement. In the case of *DBpedia*, this is realized in terms of wiki templates, akin to the wiki origins of this initiative. In relational databases, wrapping is specified through the R2RML ontology [DSC12], 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. *Bio2RDF*), wrapper ontologies (e.g. *TWC LOGD*), or a mixture (e.g. *SWEET*), depending on the target audience (i.e. programmers for *Bio2RDF* vs. the Semantic Web community for *Karma*).

Allow for LDW Deployment

Deployment starts by registering the LDW into the platform. At this point, some checks are made about LDW syntactic correctness [LIJ+15] and credential availability. Credentials are codes requested by the API servers to verify the calls are being made through a valid account. API keys are the most common mechanism. An API key is a code passed in by programs calling an API to identify the calling program, its producer, or its user.

API key provision admits two alternatives. The API key can be provided by *the LDW producer* at LDW specification time. Alternatively, the API key can be obtained from *the LDW consumer* at dereferencing time. This mimics the handling of credentials in stored procedures in Data Base Management Systems [Kyt05].

3.2.2 Consumer Requirements

Allow for LDW Discovery

LDW discovery helps consumers to identify potential LDW services. The use of ontologies become paramount in so far as providing an homogeneous semantic description (most important in a sharing setting) [DV17]. For LDWs, LIDS and LOS are two approaches to document LDW inputs and outputs using Query Graph Patterns [NKMF10, SH11]. Next, SPARQL can be used to query these patterns, though the complexity of this notation makes it not the most accessible option. In a similar vein but with a more affordable notation, *Karma* resorts to an RDF language to describe inputs, outputs and their relationships where models can be queried using SPARQL [TKSA12].

Allow for LDW Lookup

Once LDWs of interest are located, consumers need to go down to the nitty-gritty. Here, LDWs can be documented along their dual nature: implicit dataset definition *vs.* services. As for the former, LDWs can be characterized by their dataset content and dataset quality. Here, LDW producers can tap into the Vocabulary of Interlinked Datasets (VoID) [ACHZ11], an RDF Schema vocabulary for expressing metadata about RDF datasets. VoID increases discoverability and facilitates metadata consumption from multiple LDWs [HB11, SBP14]. In addition, Debattista et al. evidence the importance of the quality of Linked Data if an LD Application ecosystem wants to be developed [DLA16]. Reusable resources should provide information about their quality not only to ease the process of selection but also to increase the chances of reuse. Therefore, if LDWs are going to become reusable, quality information should be provided. Accordingly, W3C's Data Quality Vocabulary (DQV) [AI16] is being proposed to assess the dataset quality via a number of observed properties [LBC17]. As implicit definition of datasets, LDWs can be qualified along DQV.

On the other hand, as services, LDWs need to be invoked and its service quality characteristics reported. Invocation wise, producers can resort to W3C's Hydra Core Vocabulary. This lightweight vocabulary permits to create hypermedia-driven web APIs [LG13]. By specifying a number of concepts commonly used in web APIs, it enables a server to advertise valid state transitions following REST best practices. This approach can be extended to LDWs.

Allow for Resource Lookup

An LDW Platform is a Linked Data Platform [SAM15]. As such, it should comply with the W3C standard for resource management [MGCEG13]. Specifically, LDW Platforms should support resource lookup. Compared with explicit dataset, the difference stems from resources being dynamically obtained from API data at the time they are dereferenced.

3.2.3 Curator Requirements

Allow for Spotting Stains

Means are needed to make the community aware of stains in LDWs. Stains are not limited to those related with the quality of the dataset being obtained but also include those preventing the code from functioning (e.g. API upgrades). As for the former, data curation is tackled in *LOD Laundromat*, *Bio2RDF* or *TWC LOGD* (see Section 2.3.2). As for spotting code faults, inspiration can be drawn from incident management systems (e.g. JIRA [ODKM15]) and on-line Linked Data validators. For example, W3C's RDF Validation service [Pru] and *Vafu* validation service [Red] check whether Semantic Web data is correctly published according to best practices[1]. Besides automatic issue detection, users can detect and notify issues to be curated [AZS$^+$16, KHS12].

Allow for Cleaning up Stains

Once stains are spotted, mechanisms should be in place to easy a prompt repair. Different attempts have been conducted to facilitate LDW maintenance to developers other than the authors. Declarativeness is one way forward. *DBpedia* introduces *wikitext* templates, i.e. *DBpedia*-specific wrappers along the lines of Wikipedia templates. *Bio2RDF* supports open source PHP scripts, Java programs and Ruby gems into a single *GitHub* repository, facilitating scripts modification by anyone wishing to improve the quality of RDF conversions. *D2RQ* platform provides a proprietary server where LDW authors can customize automatically-generated LDWs[2]. Our scenario departs from the previous ones in the data source being wrapped, i.e., APIs rather than databases or Wikipedia.

3.3 Conclusion

This chapter gathers requirements for different stakeholders involved in LDW creation, consumption and curation. Requirements are motivated by the existing literature, outlining different ways in which the requirement is being addressed so far. The intention is not to provide an exhaustive account but just to motivate the need.

[1]Best practices are those defined by the Linked Data principles [BL06], the Best Practice Recipes [BPM$^+$08] and the Cool URIs [SCAV08].

[2]The *generate-mapping* tool creates a D2RQ mapping file by analyzing the schema of an existing database where table names and column names are used as default values. Next, administrators can customize these default mappings to curate the generated code.

Chapter 4

Realization

"For, in fact, the common man, finding himself in a world so excellent, technically and socially, believes that it has been produced by nature, and never thinks of the personal efforts of highly-endowed individuals which the creation of this new world presupposed."
– José Ortega y Gasset

4.1 Overview

This chapter describes the SYQL (Semantic YQL) platform, an artifact which fulfills requirements listed in Chapter 3. SYQL is heavily based on YQL. Besides the technical facilities, YQL allows us to tap into an existing community. At the time of this writing (July 2017), the YQL community exhibits the following figures [ODT]: 151 contributors, 3291 commits, 37 open and 17 closed issues, 25 open and 403 closed pull requests, 732 stars, and 464 forks. We believe LDW concerns are not so alien to API programmers. By moving to YQL, our hope is to tap into this sibling community.

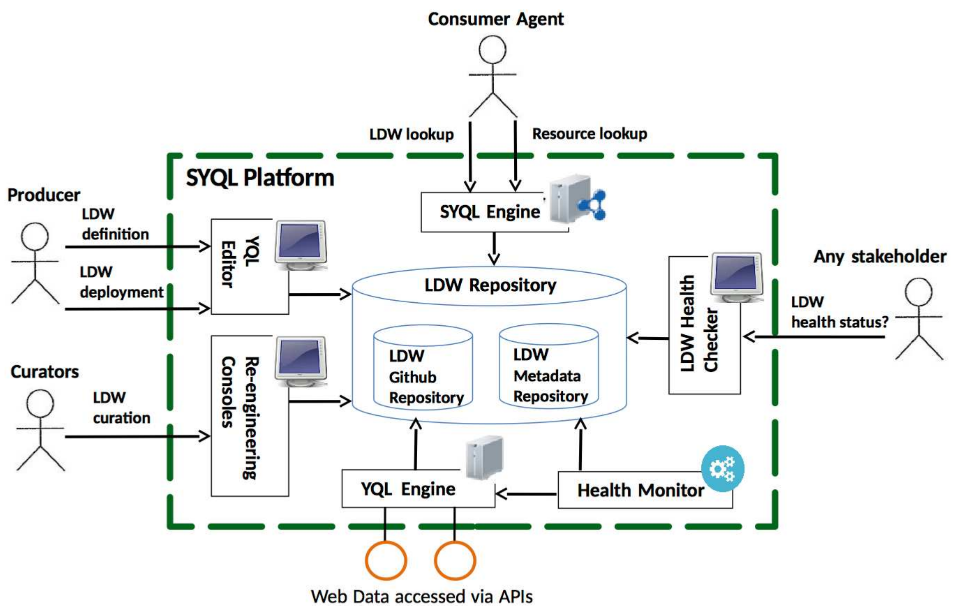


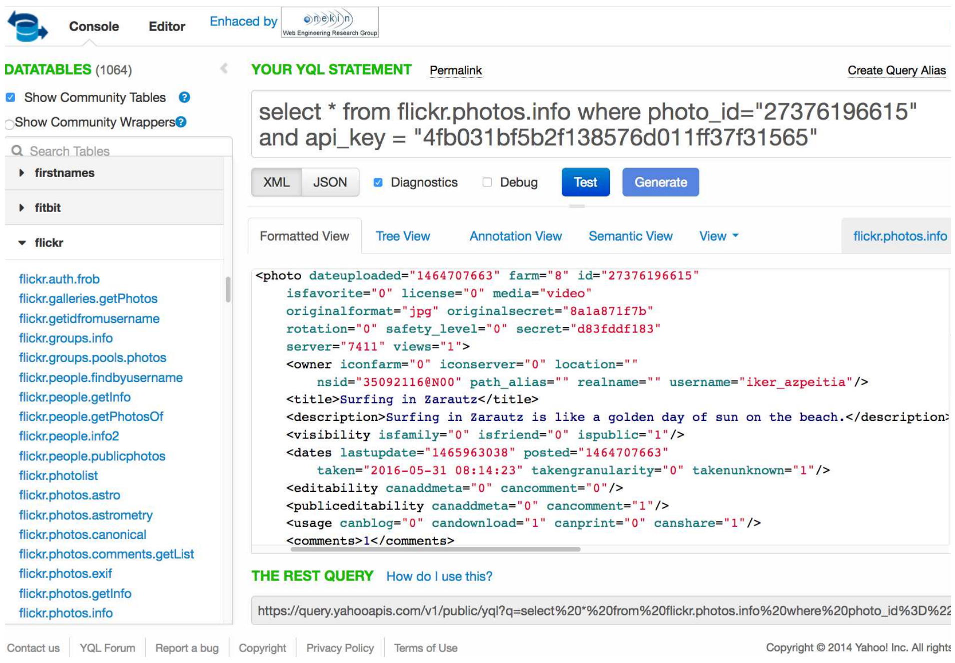
Figure 4.1: SYQL main components

4.2 A Platform for LDW Management: Architecture

Figure 4.1 outlines SYQL's architecture. Care is taken for the three stakeholders: consumers (developing applications where resources are dereferenced); producers (defining and deploying LDWs from scratch) and curators (curating and tracking LDW functioning status). Next sections delve into how these stakeholders' needs are considered in SYQL. A video about the different services is available at `http://rdf.onekin.org/`. First, a brief about YQL is provided.

4.2.1 YQL Basics

YQL is a query engine, which is hosted by Yahoo, and exposed as a REST endpoint. Requests are specified in terms of a SQL-like language: the *Yahoo Query Language*. Here, this dissertation will use "YQL" to denote both, i.e. the platform and the language, unless the context does not make



```
select * from flickr.photos.info where photo_id="27376196615"
and api_key = "4fb031bf5b2f138576d011ff37f31565"
```

```
<photo dateuploaded="1464707663" farm="8" id="27376196615"
    isfavorite="0" license="0" media="video"
    originalformat="jpg" originalsecret="8a1a871f7b"
    rotation="0" safety_level="0" secret="d83fddf183"
    server="7411" views="1">
    <owner iconfarm="0" iconserver="0" location=""
        nsid="35092116@N00" path_alias="" realname="" username="iker_azpeitia"/>
    <title>Surfing in Zarautz</title>
    <description>Surfing in Zarautz is like a golden day of sun on the beach.</description>
    <visibility isfamily="0" isfriend="0" ispublic="1"/>
    <dates lastupdate="1465963038" posted="1464707663"
        taken="2016-05-31 08:14:23" takengranularity="0" takenunknown="1"/>
    <editability canaddmeta="0" cancomment="0"/>
    <publiceditability canaddmeta="0" cancomment="1"/>
    <usage canblog="0" candownload="1" canprint="0" canshare="1"/>
    <comments>1</comments>
```

```
https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20flickr.photos.info%20where%20photo_id%3D%22
```

Figure 4.2: YQL Console. The "Annotation View" tab is added for LDW curation

clear which one it refers to.

YQL aims at hiding APIs' specifics into a uniform table-like metaphor. To this end, it resorts to a SQL-like syntax. As an example, the following YQL statement retrieves *Flickr* data about the photo (or video) whose ID is *27376196615* (see Figure 4.2):

> *select * from flickr.photos.info where photo_id="27376196615"*
> *and api_key = "4fb031bf5b2f138576d011ff37f31565"*

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

The *YQL* Language. YQL includes *SELECT, INSERT, UPDATE* and *DELETE* statements that permit to handle API requests *à la SQL*. Behind the curtains, YQL maps these statements into the corresponding API methods. To this end, producers should provide Open Data Tables.

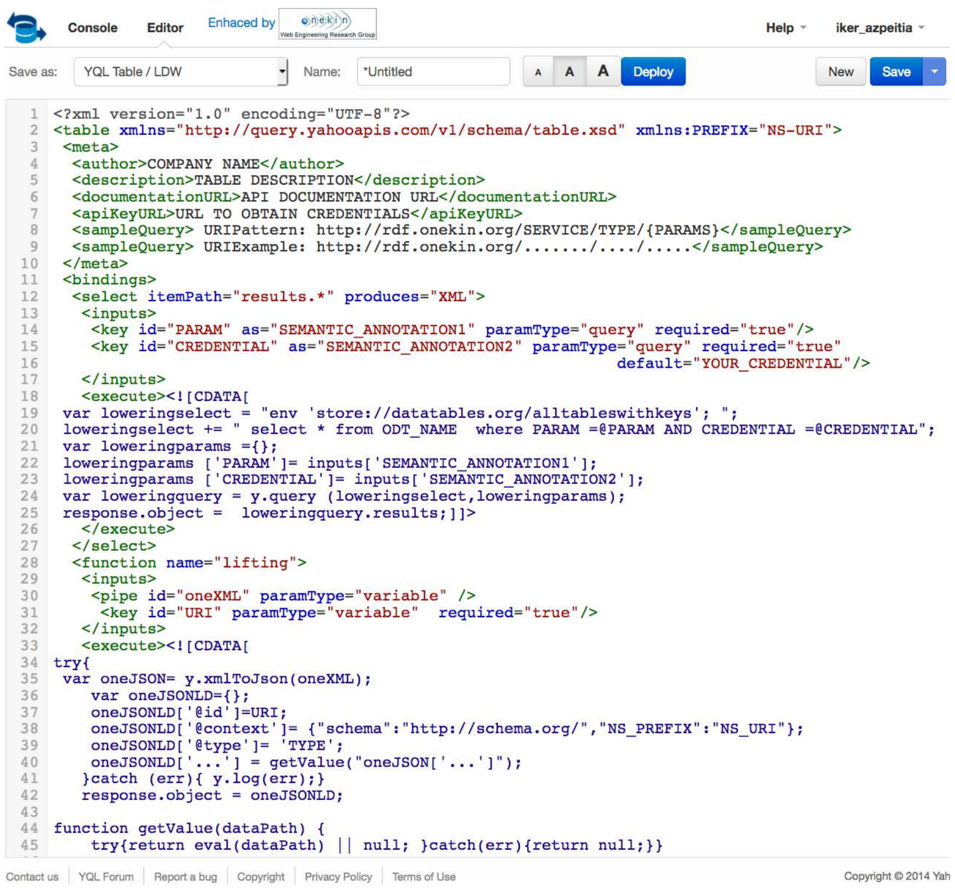
Open Data Tables (ODTs). Broadly, ODTs are syntactic sugar for

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <table xmlns="http://query.yahooapis.com/v1/schema/table.xsd">
3  <meta>
4   <author>Yahoo! Inc.</author>
5   <documentationURL>http://www.flickr.com/services/api/flickr.photos.getInfo.html</documentationURL>
6   <sampleQuery>select * from flickr.photos.info where photo_id='2186714153'</sampleQuery>
7   <apiKeyURL>http://www.flickr.com/services/apps/create/apply/</apiKeyURL>
8  </meta>
9   <bindings>
10  <select itemPath="rsp.photo" produces="XML">
11   <urls><url env="all">https://api.flickr.com/services/rest/?method=flickr.photos.getInfo</url></urls>
12   <inputs>
13         <key id="photo_id" type="xs:string" paramType="query" />
14         <key id="api_key" type="xs:string" paramType="query" private="true"/>
15   </inputs>
16  </select>
17  </bindings>
18 </table>
```

Figure 4.3: ODT *flickr.photos.info*

API parameters. Figure 4.3 shows the *flickr.photos.info* ODT. Main tags include <*meta*> and <*bindings*>. The former contains descriptive information about the ODT such as author, description or documentation link (lines 3-8). Bindings (lines 9-17) indicate how SQL operations are mapped into API calls. An entry exists for each operation (i.e. <*select*>, <*insert*>, <update> and <delete>). The snippet illustrates the SELECT case (lines 10-16): <*url*> accounts for the URL pattern to invoke (line 11) whereas <*inputs*> denotes the possible YQL statement input fields (lines 12-15). Each field (e.g. *photo_id*) accounts for variables to be instantiated when SELECT is enacted. 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 API access control mechanisms. 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"), ODTs can be provided by producers (known as "community tables"). A full list of community tables can be found at `http://www.datatables.org/`.

The YQL Console. The YQL Console [Netb] enables to run YQL statements interactively from a browser (see Figure 4.2). 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 REST counterpart of the YQL statement, ready to be embedded in the application.



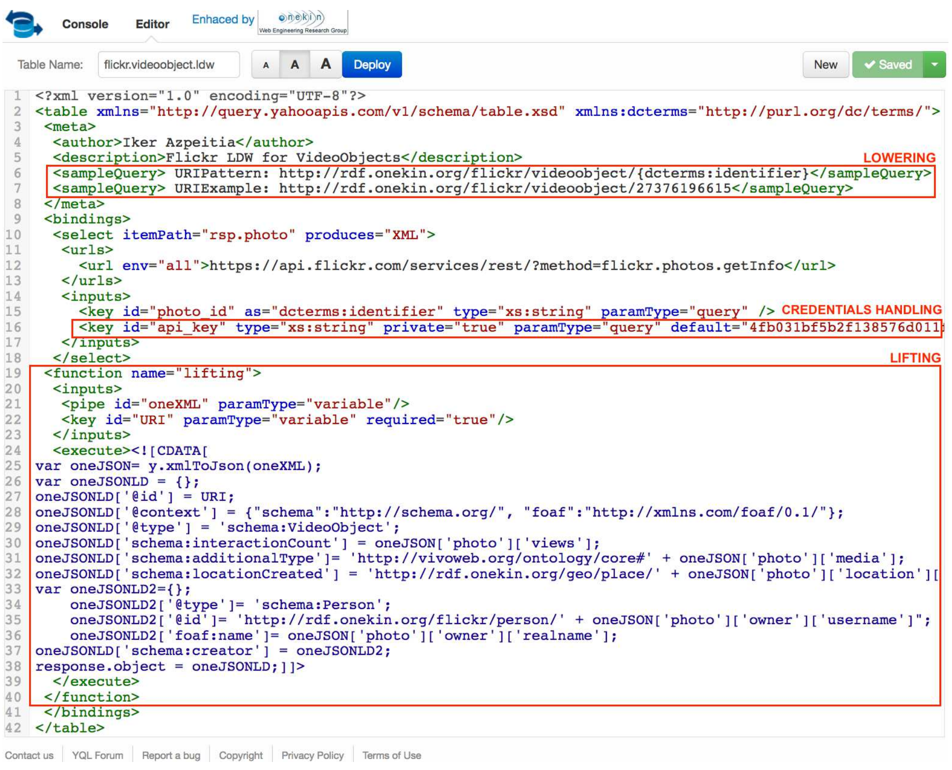
```
<?xml version="1.0" encoding="UTF-8"?>
<table xmlns="http://query.yahooapis.com/v1/schema/table.xsd" xmlns:PREFIX="NS-URI">
 <meta>
  <author>COMPANY NAME</author>
  <description>TABLE DESCRIPTION</description>
  <documentationURL>API DOCUMENTATION URL</documentationURL>
  <apiKeyURL>URL TO OBTAIN CREDENTIALS</apiKeyURL>
  <sampleQuery> URIPattern: http://rdf.onekin.org/SERVICE/TYPE/{PARAMS}</sampleQuery>
  <sampleQuery> URIExample: http://rdf.onekin.org/......./..../.....</sampleQuery>
 </meta>
 <bindings>
  <select itemPath="results.*" produces="XML">
   <inputs>
    <key id="PARAM" as="SEMANTIC_ANNOTATION1" paramType="query" required="true"/>
    <key id="CREDENTIAL" as="SEMANTIC_ANNOTATION2" paramType="query" required="true"
                                                   default="YOUR_CREDENTIAL"/>
   </inputs>
   <execute><![CDATA[
 var loweringselect = "env 'store://datatables.org/alltableswithkeys'; ";
 loweringselect += " select * from ODT_NAME  where PARAM =@PARAM AND CREDENTIAL =@CREDENTIAL";
 var loweringparams ={};
 loweringparams ['PARAM']= inputs['SEMANTIC_ANNOTATION1'];
 loweringparams ['CREDENTIAL']= inputs['SEMANTIC_ANNOTATION2'];
 var loweringquery = y.query (loweringselect,loweringparams);
 response.object =  loweringquery.results;]]>
   </execute>
  </select>
  <function name="lifting">
   <inputs>
    <pipe id="oneXML" paramType="variable" />
     <key id="URI" paramType="variable"  required="true"/>
   </inputs>
   <execute><![CDATA[
try{
 var oneJSON= y.xmlToJson(oneXML);
    var oneJSONLD={};
    oneJSONLD['@id']=URI;
    oneJSONLD['@context']= {"schema":"http://schema.org/","NS_PREFIX":"NS_URI"};
    oneJSONLD['@type']= 'TYPE';
    oneJSONLD['...'] = getValue("oneJSON['...']");
   }catch (err){ y.log(err);}
   response.object = oneJSONLD;

function getValue(dataPath) {
    try{return eval(dataPath) || null; }catch(err){return null;}}
```

Figure 4.4: An LDW template to be completed

Community tables are listed on the left. Once an ODT table is selected, YQL statements (e.g. SELECT) can be enacted, and the results show up at the "Formatted View" tab. In addition, the REST-call counterpart of the query is also provided, ready to be embedded in the application. Next, this dissertation looks at how this approach can be extended for LDWs.

4.3 Addressing Producer Requirements

Producers develop LDWs from scratch. At design time, they look for APIs (or even better, YQL ODTs) that meet their data needs. At implementation time, they resort to a wrapping template for addressing lifting and lowering



```
<?xml version="1.0" encoding="UTF-8"?>
<table xmlns="http://query.yahooapis.com/v1/schema/table.xsd" xmlns:dcterms="http://purl.org/dc/terms/">
 <meta>
  <author>Iker Azpeitia</author>
  <description>Flickr LDW for VideoObjects</description>
  <sampleQuery> URIPattern: http://rdf.onekin.org/flickr/videoobject/{dcterms:identifier}</sampleQuery>
  <sampleQuery> URIExample: http://rdf.onekin.org/flickr/videoobject/27376196615</sampleQuery>
 </meta>
 <bindings>
  <select itemPath="rsp.photo" produces="XML">
   <urls>
     <url env="all">https://api.flickr.com/services/rest/?method=flickr.photos.getInfo</url>
   </urls>
   <inputs>
     <key id="photo_id" as="dcterms:identifier" type="xs:string" paramType="query" />
     <key id="api_key" type="xs:string" private="true" paramType="query" default="4fb031bf5b2f138576d011
   </inputs>
  </select>
 <function name="lifting">
  <inputs>
   <pipe id="oneXML" paramType="variable"/>
   <key id="URI" paramType="variable" required="true"/>
  </inputs>
  <execute><![CDATA[
var oneJSON= y.xmlToJson(oneXML);
var oneJSONLD = {};
oneJSONLD['@id'] = URI;
oneJSONLD['@context'] = {"schema":"http://schema.org/", "foaf":"http://xmlns.com/foaf/0.1/"};
oneJSONLD['@type'] = 'schema:VideoObject';
oneJSONLD['schema:interactionCount'] = oneJSON['photo']['views'];
oneJSONLD['schema:additionalType']= 'http://vivoweb.org/ontology/core#' + oneJSON['photo']['media'];
oneJSONLD['schema:locationCreated'] = 'http://rdf.onekin.org/geo/place/' + oneJSON['photo']['location'][
var oneJSONLD2={};
    oneJSONLD2['@type']= 'schema:Person';
    oneJSONLD2['@id']= 'http://rdf.onekin.org/flickr/person/' + oneJSON['photo']['owner']['username']";
    oneJSONLD2['foaf:name']= oneJSON['photo']['owner']['realname'];
oneJSONLD['schema:creator'] = oneJSONLD2;
response.object = oneJSONLD;]]>
  </execute>
 </function>
 </bindings>
</table>
```

Figure 4.5: YQL Editor Console. Turning the ODT in Figure 4.3 into an LDW

(i.e. LDW definition). At deployment time, they register LDWs with the platform (i.e. LDW deployment).

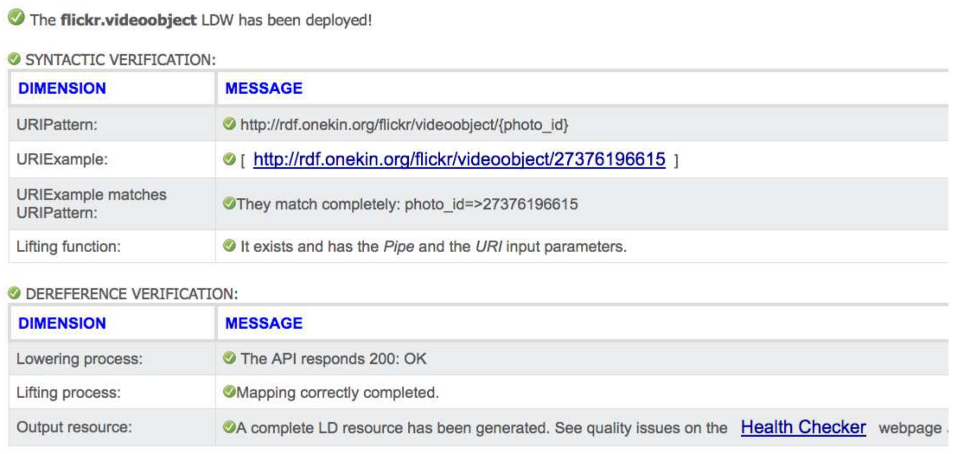
4.3.1 Allow for LDW Definition

SYQL resorts to YQL expressiveness to define LDWs and provides a **"SELECT wrapping template"** to guide developers (see Figure 4.4). The template accounts for the three main steps: lowering, lifting and credentials handling. As an example, let's tap into the ODT in Figure 4.3, and turn it into an LDW (see Figure 4.5).

Lowering (i.e. mapping the URI's (e.g. http://rdf.onekin.com/flickr/videoobject/{itemNumber}) to the corresponding API call (e.g. https://api.flickr.com/services/rest/?method=flickr.photos.getInfo&

photo_id={itemNumber})). YQL's *sampleQuery* tag is used to describe the lowering through the URI pattern (line 6) and some URI examples (line 7). When the SYQL platform receives a URI (e.g. `http://rdf.onekin.org/flickr/videoobject/27376196615`), it dynamically identifies the ODT at hand through pattern matching against the registered *URIPatterns*. The lowering mapping from the *URIPattern* to the ODT input parameters is realized through pattern matching (i.e. line 6 to line 15 *{dcterms:identifier}* binding). Worth noticing, the URI parameter is annotated along the *dcterms* ontology (line 6). This will turn useful during LDW discovery. Note too that the select part in this example lacks the execute part shown in the LDW template. It is due to the API call is as simple as a REST call to *Flickr*. In this way an indirection (i.e. calling to other ODT through a SELECT statement) is avoided. In cases where API calls are complex and they are programed into an ODT this indirection may be advisable.

Lifting (i.e. creation of the Linked Data Resource from the API result). YQL's *function* tag is recast for lifting. Specifically, the *wrapping template* advices each XML tuple to be turned into an RDF resource which is serialized as JSON-LD (i.e. *oneJSONLD*, line 26). The *lifting* function holds <*inputs*> and <*execute*> tags. The former indicates the function's parameters which are set to <*pipe*> (i.e. holds a result tuple of the ODT table described *à la* XML) (line 21) and <*key*> (i.e. to cast the URI for the returned RDF resource) (line 22). As for <*execute*> (lines 24-39), it holds the JavaScript code that obtains JSON-LD from the XML tuple (i.e. from *oneXML* pipe input to *oneJSONLD*). The line 25 parses the *oneXML* input to a JSON object. The lines 28 and 29 create the namespace and the type of the resource, respectively. Line 30 creates an RDF property from an *oneJSON* parameter. Interlinkage is also described here by constructing URIs out of existing parameters. Specifically, line 31 links the video to a *vivoweb* ontology class type and line 32 links to a *GeoPlanet* resource about the locality. Lines 33-36 create an embedded *schema:Person* resource that is linked to the video through the *schema:creator* association


✓ The **flickr.videoobject** LDW has been deployed!

✓ SYNTACTIC VERIFICATION:

DIMENSION	MESSAGE
URIPattern:	✓ http://rdf.onekin.org/flickr/videoobject/{photo_id}
URIExample:	✓ [http://rdf.onekin.org/flickr/videoobject/27376196615]
URIExample matches URIPattern:	✓They match completely: photo_id=>27376196615
Lifting function:	✓ It exists and has the *Pipe* and the *URI* input parameters.

✓ DEREFERENCE VERIFICATION:

DIMENSION	MESSAGE
Lowering process:	✓ The API responds 200: OK
Lifting process:	✓Mapping correctly completed.
Output resource:	✓A complete LD resource has been generated. See quality issues on the Health Checker webpage

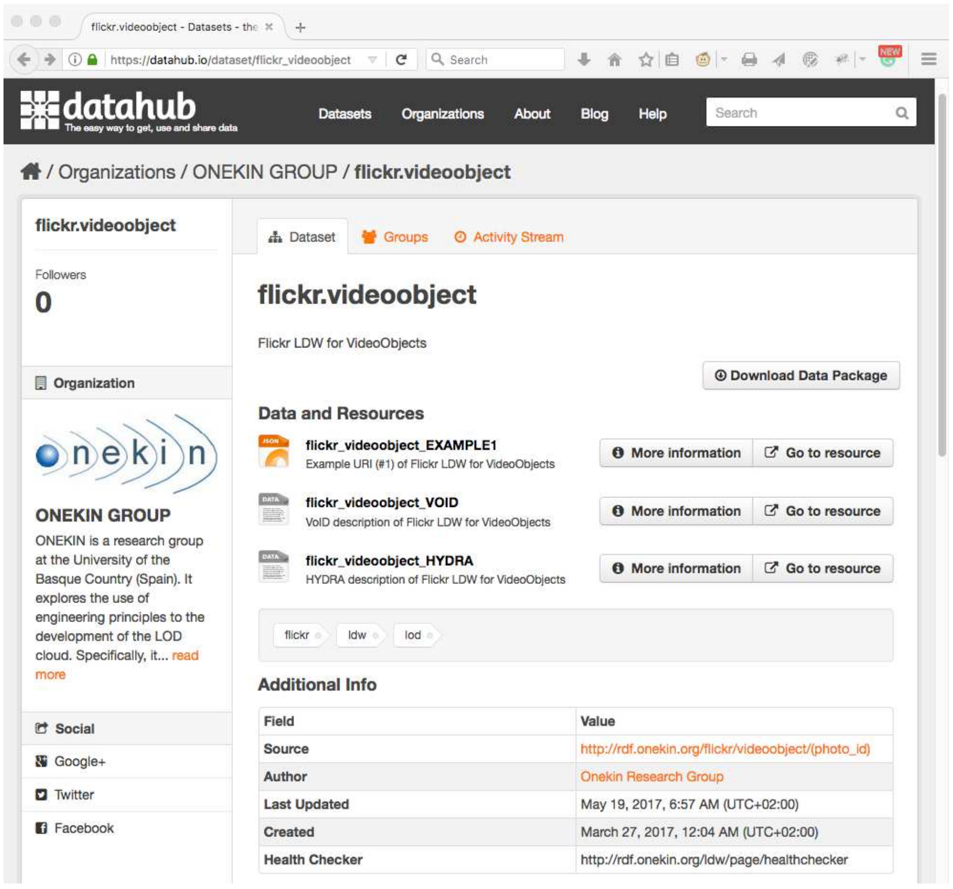
Figure 4.6: SYQL Verification window. Acknowledgement messages at deployment time

(line 37).

Credentials handling. API keys are codes requested by the servers to verify the calls are being made through a valid account. The question arise about whether these keys should be provided by either the *LDW producer* or the *LDW consumer*. When performance is not an issue (the number of invocations per API key is limited), *LDW consumers* can stick to the producer's API key. In this case, the API key is embedded in the LDW itself (see default value in line 16). In this way, all lookups will reuse the same API key. In a more demanding setting, the extensive use of the same API key could cause a capacity bottleneck. Here, *LDW producers* might resort to API keys which are provided by consumers at lookup time.

4.3.2 Allow for LDW Deployment

Once defined, LDWs need to be deployed before being stored at the *GitHub* repository. Deployment also takes place through the YQL Editor. Besides setting the different registries, LDW deployment also includes quality verifications. After all, this is a reuse architecture where eventual errors expand beyond the original authors to potential LDW consumers.



flickr.videoobject - Datasets - the
https://datahub.io/dataset/flickr_videoobject
Search
datahub
The easy way to get, use and share data
Datasets
Organizations
About
Blog
Help
Search

/ Organizations / ONEKIN GROUP / **flickr.videoobject**

flickr.videoobject

Followers

0

Organization

ONEKIN GROUP

ONEKIN is a research group at the University of the Basque Country (Spain). It explores the use of engineering principles to the development of the LOD cloud. Specifically, it... read more

Social

Google+

Twitter

Facebook

Dataset | Groups | Activity Stream

flickr.videoobject

Flickr LDW for VideoObjects

Download Data Package

Data and Resources

flickr_videoobject_EXAMPLE1
Example URI (#1) of Flickr LDW for VideoObjects — More information | Go to resource

flickr_videoobject_VOID
VoID description of Flickr LDW for VideoObjects — More information | Go to resource

flickr_videoobject_HYDRA
HYDRA description of Flickr LDW for VideoObjects — More information | Go to resource

flickr ldw lod

Additional Info

Field	Value
Source	http://rdf.onekin.org/flickr/videoobject/(photo_id)
Author	Onekin Research Group
Last Updated	May 19, 2017, 6:57 AM (UTC+02:00)
Created	March 27, 2017, 12:04 AM (UTC+02:00)
Health Checker	http://rdf.onekin.org/ldw/page/healthchecker

Figure 4.7: Metadata for the *flickr.videoobject* LDW in *DataHub*

So far, two types of verifications are conducted, namely, syntactic and dereferenced-based (see Figure 4.6). Failure to meet any of them prevents the LDW from being registered.

Syntactic verification. It checks whether LDWs are schema compliant. Through an XML Schema parser, distinct syntactic errors are pointed out: no *URIPattern*, no *URIExample*, lack of lifting <*function*>; LDW badly parameterized.

Dereference verification. LDW definitions include *URIExample*s. At registration time, LDWs are put to the test using these *URIExample*s. Possible errors include: not enough credentials, no resource returned, or JavaScript errors.

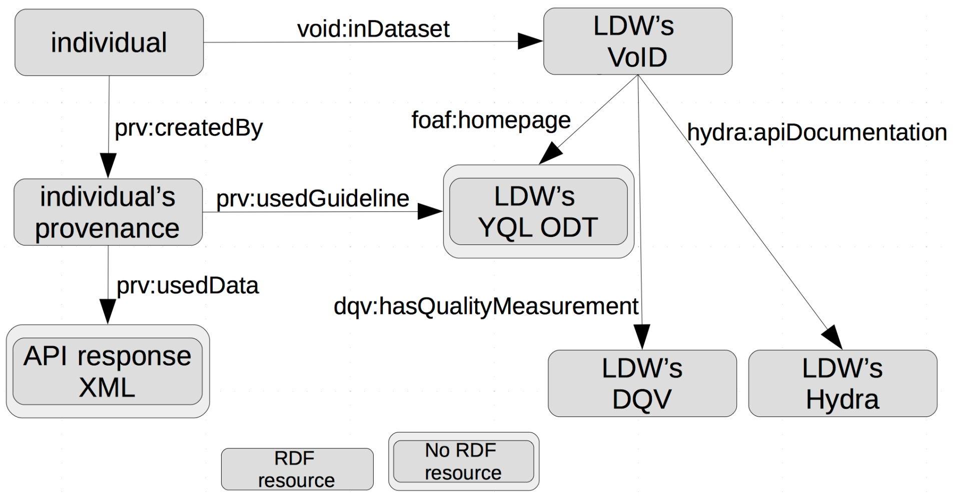


Figure 4.8: Individuals and LDW representations

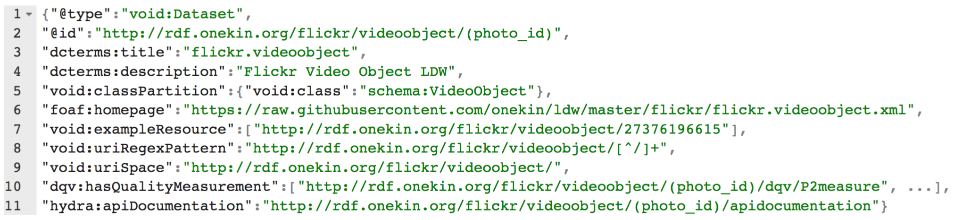
Quality issues are detected as well but they do not prevent registration. Instead, they are shown in the Health Checker to warn about quality issues (see Section 4.5).

4.4 Addressing Consumer Requirements

Consumers build applications out of LDWs. At design time, consumers look for LDWs that meet their data and quality service needs (i.e. LDW discovery). At implementation time, consumers need help to create the environment for calling LDWs. Finally, at runtime, consumers' applications dereference individuals obtained through LDWs (i.e. resource lookup).

4.4.1 Allow for LDW Discovery

SYQL does not support LDW discovery. Rather, it relies on the *DataHub* portal for LDW discovery. Refer to https://datahub.io/organization/linked-data-wrappers for details. The aim: increasing the visibility of SYQL LDWs. *DataHub* records and lists datasets metadata. After deploying an LDW, SYQL automatically registers it in *DataHub* to make easier its



```
1  {"@type":"void:Dataset",
2  "@id":"http://rdf.onekin.org/flickr/videoobject/(photo_id)",
3  "dcterms:title":"flickr.videoobject",
4  "dcterms:description":"Flickr Video Object LDW",
5  "void:classPartition":{"void:class":"schema:VideoObject"},
6  "foaf:homepage":"https://raw.githubusercontent.com/onekin/ldw/master/flickr/flickr.videoobject.xml",
7  "void:exampleResource":["http://rdf.onekin.org/flickr/videoobject/27376196615"],
8  "void:uriRegexPattern":"http://rdf.onekin.org/flickr/videoobject/[^/]+",
9  "void:uriSpace":"http://rdf.onekin.org/flickr/videoobject/",
10 "dqv:hasQualityMeasurement":["http://rdf.onekin.org/flickr/videoobject/(photo_id)/dqv/P2measure", ...],
11 "hydra:apiDocumentation":"http://rdf.onekin.org/flickr/videoobject/(photo_id)/apidocumentation"}
```

Figure 4.9: *Flickr.videoobject* LDW VoID description

discovery. In that way, access points to the LDW's VoID and Hydra descriptions are provided (see Figure 4.7).

Visibility, and eventually the recognition that goes by using LDWs, might turn rather important. Recognition is being reported as one of the main spurs for sharing [PS09]. The Semantic Web community has so understood when the Semantic Web Journal announced in 2012 the first "Special Call" for Linked Dataset descriptions as a way not only to disseminate but also to acknowledge the effort and importance of these resources [HHJ16]. In the same way that explicit datasets, LDWs might avail of these initiatives.

4.4.2 Allow for LDW Lookup

For LDW description, SYQL resorts to the combined use of VoID, Hydra and DQV. Figure 4.8 sets the two main resource types: individuals and LDWs. LDWs exhibit a two-fold nature. As implicit definition of datasets, they can be characterized through VoID. As REST services, LDWs might be documented through Hydra. Figure 4.9 shows the VoID description for an LDW dataset along the structure depicted in Figure 4.8. Since resources are generated on the fly, it is not possible to work out statistical information (e.g. the number of entities stored in the API's service). However, other structural metadata is provided in the VoID description: the class of the individuals (line 5), *GitHub* repository for the LDW code (line 6), the example URI (line 7), the pattern of supported URIs (line 8), the base URI (line 9). In addition, *dqv:hasQualityMeasurement* links to a set of DQV resources (line 10) while *hydra:apiDocumentation* points to the LDW's Hydra documents (line 11).

LDW lookup might be conducted by both humans and agents. The former, to be informed about LDW characteristics. For easy access, SYQL turns (part of) this information into an HTML page: the Health Checker (see Section 4.5). In addition, and similar to the role of WSDL for web services, interpreting and invoking LDWs might be facilitated by the use of standards for LDW description. Hydra allows data to be enriched with machine-readable affordances which enable interaction. By specifying a number of concepts commonly used in web APIs, it enables a server to advertise valid state transitions following REST best practices.

Specifically, SYQL resorts to Hydra for a main purpose: credential provision. Credentials can be provided by either producers (at deployment time) and consumers (at resource lookup). The former scenario might lead to a capacity bottleneck if a large number of resource lookups are based on the same API key. Alternatively, SYQL might also avail of API keys provided by *consumers* at lookup time. This is when Hydra comes into play. The LDW's Hydra document holds an RDF credential description to be used at the time resources are looked up. Consumer-provided keys take precedence over producer-provided keys.

4.4.3 Allow for Resource Lookup

Once deployed an LDW, the LDW Platform starts dereferencing URIs that conform to the LDW's *URIPattern*. URI dereferencing involves five main tasks (see Figure 4.10):

1. LDW retrieval, where the wrapper is downloaded from the LDW repository;

2. lowering, where the YQL select statement is prepared, and the credentials provided;

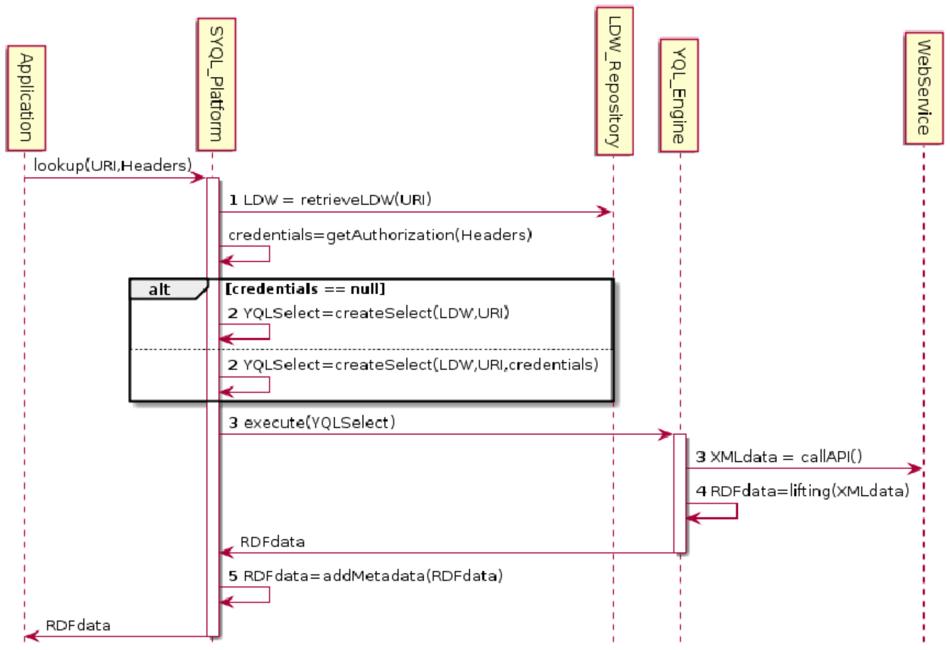


Figure 4.10: URI lookup sequence diagram. The "alt" deviation tackles the consumer-provided API-key scenario

3. API calling, where the select statement is enacted, and the XML document obtained[1];

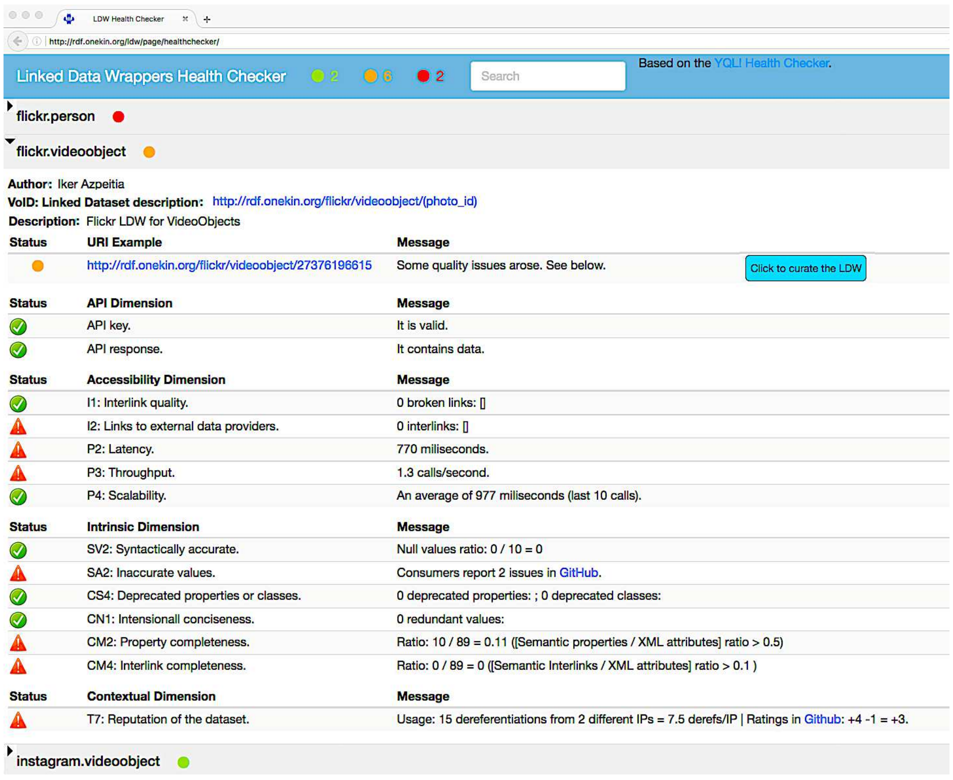
4. lifting, where the XML document is turned into an RDF resource; and finally

5. metadata enrichment, where dataset and provenance metadata are added[2].

As for the latter, Figure 2.3 shows an example along the structure depicted in Figure 4.8: *void:inDataset* holds a link to VoID dataset metadata (line

[1]SYQL focuses on wrapping the REST APIs whose inputs are given as part of the invocation URIs. Alternatively, APIs might also take XML or JSON as input (using HTTP POST). An example is the use of OAuth as an authentication mechanism. Here, API call is not as easy as constructing an URL string but parameters need first to be encrypted and next, passed as a POST parameter. This is commonly taken care of within the YQL's ODT.

[2]SYQL resorts to the Provenance Ontology `http://purl.org/net/provenance/ns#`.


LDW Health Checker
http://rdf.onekin.org/ldw/page/healthchecker/
Linked Data Wrappers Health Checker
● 2 ● 6 ● 2
Search
Click to curate the LDW

Based on the YQL! Health Checker.

flickr.person ●

flickr.videoobject ●

Author: Iker Azpeitia
VoID: Linked Dataset description: http://rdf.onekin.org/flickr/videoobject/(photo_id)
Description: Flickr LDW for VideoObjects

Status	URI Example	Message
●	http://rdf.onekin.org/flickr/videoobject/27376196615	Some quality issues arose. See below.

Status	API Dimension	Message
✓	API key.	It is valid.
✓	API response.	It contains data.

Status	Accessibility Dimension	Message
✓	I1: Interlink quality.	0 broken links: []
⚠	I2: Links to external data providers.	0 interlinks: []
⚠	P2: Latency.	770 miliseconds.
⚠	P3: Throughput.	1.3 calls/second.
✓	P4: Scalability.	An average of 977 miliseconds (last 10 calls).

Status	Intrinsic Dimension	Message
✓	SV2: Syntactically accurate.	Null values ratio: 0 / 10 = 0
⚠	SA2: Inaccurate values.	Consumers report 2 issues in GitHub.
✓	CS4: Deprecated properties or classes.	0 deprecated properties: ; 0 deprecated classes:
✓	CN1: Intensionall conciseness.	0 redundant values:
⚠	CM2: Property completeness.	Ratio: 10 / 89 = 0.11 ([Semantic properties / XML attributes] ratio > 0.5)
⚠	CM4: Interlink completeness.	Ratio: 0 / 89 = 0 ([Semantic Interlinks / XML attributes] ratio > 0.1)

Status	Contextual Dimension	Message
⚠	T7: Reputation of the dataset.	Usage: 15 dereferentiations from 2 different IPs = 7.5 derefs/IP | Ratings in Github: +4 -1 = +3.

instagram.videoobject ●

Figure 4.11: Health Checker Console

12); *prv:usedData* describes the data source (line 14); *prv:usedGuideline* indicates how the data has been created (e.g. pointing to the LDW URL) (lines 15 and 16).

4.5 Addressing Curator Requirements

Curators keep LDWs in shape. At design time, curators become aware of LDW stains. At implementation time, curators clean stains by upgrading the LDW at hand. Finally, at deployment time, LDWs are checked to be fully functional.

4.5.1 Allow for Spotting Stains

SYQL introduces the Health Checker, a daemon that periodically checks LDWs for stains, and renders the output as a Web page. Figure 4.11 illustrates the case for the 10 LDWs developed so far: *green* denotes that the LDW works and passes all the quality filters (2 LDWs); *yellow* indicates that the LDW works but still holds some stains (6 LDWs)*;* finally, *red* indicates that the LDW does not work, i.e. returns an error status code (2 LDW).

Stains can refer to either the functioning status or data quality issues. Hereafter, Zaveri et al.´s quality framework is used [ZRM$^+$16].

Functioning-status Stains This mainly corresponds to the **Accessibility** dimension in the Zaveri et al.´s quality framework. It involves aspects related to "the access, authenticity and retrieval of data to obtain either the entire or some portion of the data (or from another linked dataset) for a particular use case". Table 4.1 indicates how quality aspects find their way in SYQL. For instance, availability is checked out by dereferencing the LDW's sample URI. So, sample URIs act as a sort of regression testing bucket. In addition, SYQL keeps an aggregate of how LDWs behave in the last 10 calls w.r.t. latency (Zaveri et al.´s P2 subdimension), throughput (P3) and scalability (P4). Back to Figure 4.11, click on the *flickr.videoobject* LDW for its quality measures to show up: contains no interlinks (the I2 subdimension), 770 millisecond latency, 1.3 calls/second throughput, and 977 millisecond average elapsed time for the last ten calls ("scalability").

In addition to Zaveri et al's characteristics, this dissertation includes two issues of concern for APIs: the expiration of the API key, and the return of no value by the API. Both scenarios are also noted in the Health Checker window (see Figure 4.11 under the heading *"API Dimension"*).

Table 4.1: Quality dimensions. *"Abr"* stands for the abbreviation used in [ZRM+16]

Dimension	**Subdimension**	**Abr**	**Metric**	**SYQL realization**
Accessibility	Availability	A3	Dereferenceability of the URI	Sample URIs work
	Interlinking	I1	Detection of good quality interlinks	Number of broken links
		I2	Existence of links to external data producers	Number of external links
	Performance	P2	Low latency	Minimum request to response delay
		P3	High throughput	Number of requests per second
		P4	Scalability of a data source	Average throughput of the last ten calls
Intrinsic	Syntactic validity	SV2	Syntactically accurate values	Detection of null values
	Semantic accuracy	SA2	No inaccurate values	Notifications via *GitHub* comments
	Consistency	CS4	owl:DeprecatedProperty not used	Number of deprecated properties
	Conciseness	CN1	High intensional conciseness	Number of redundant attributes
	Completeness	CM2	Property completeness	Rate of XML elements lifted
		CM4	Interlinking completeness	Rate of XML elements in interlinks
Contextual	Trustworthiness	T7	Reputation of the dataset	Number of derefs and ratings in *GitHub*

Data-quality Stains This mainly corresponds to the Intrinsic and Contextual dimensions in the Zaveri et al.´s quality framework.

Intrinsic. It refers to whether information correctly (syntactically and semantically), compactly and completely represents the real world, and whether information is logically consistent in itself, independently of the user's context. Back to the example, Figure 4.11 reports three warnings. First, the SA2 subdimension: consumers report 2 issues through *GitHub*. Second, the CM2 subdimension: the ratio of properties per XML attributes is low. Finally, the CM4 subdimension: the ratio of interlinks per XML attributes is low. The values are computed along the formulae proposed by Zaveri et al.

Contextual. This dimension tackles aspects that highly depend on the context of the task at hand. For trustworthiness, SYQL supports the reputation of the dataset (i.e. "assignment of explicit trust ratings to the dataset by humans or analyzing external links or page ranks"). This can be worked out based on human rating and rate of LDW reuse. SYQL works out these measures from LDWs' *GitHub* repositories, specifically from how users rate the LDW's code. As for LDW reuse, SYQL keeps a counter of the number of times the LDW is being used from different IPs. Back to the example, Figure 4.11 indicates that the sample LDW has been subject to 15 dereferenciations from 2 different IPs where the LDW has received 4 thumbs up and 1 thumbs down.

Worth mentioning, some of Zaveri et al.´s features are met "by construction". That is, the fact that datasets are obtained out of API calls ensures the fulfillment of the following features:

- human-readable properties and metadata (U1): Figure 2.3 shows the automatically generated metadata. In addition, the description, author name, etc. are extracted from the LDW definition.
- exemplary URIs (U2): the *URIExample* is compulsory for the lowering process (see line 7 in Figure 4.5).
- regular expression that matches the URI of the dataset (U3): the

URIPattern also is compulsory and allows to derive a regular expression (see line 6 in Figure 4.5).

- indication of the vocabularies used (U5): the vocabularies are taken from the *@context* property in the lifting function (see line 28 in Figure 4.5).
- provision of the data in different serialization formats (V1): the SYQL server performs content negotiation and dispatches the requested serialization format: JSON-LD, RDF/XML, Notation3, N-Quads, N-Triples or Turtle.

4.5.2 Allow for Cleaning up Stains

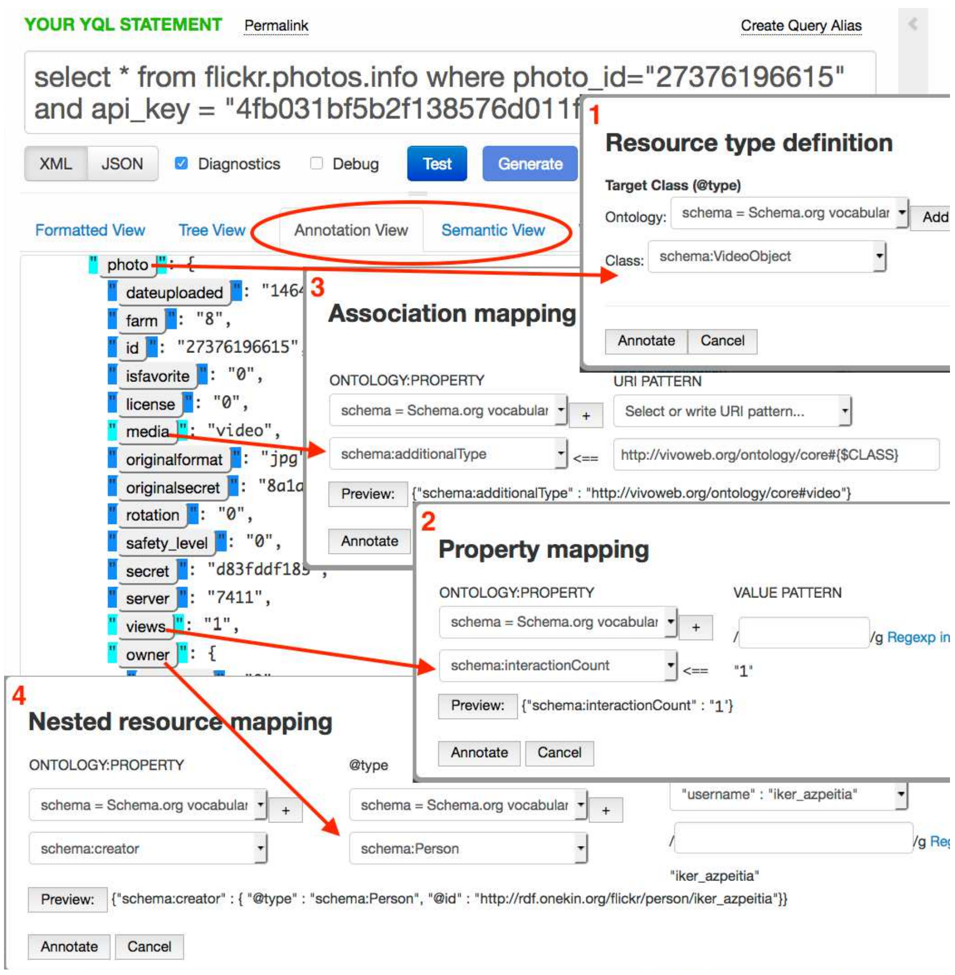
Producers start from scratch. By contrast, curators do not start afresh but depart from someone else's code. This moves to the forefront understandability.

SYQL resorts to JavaScript for LDW implementation. This might put some curators off. To fight this back, SYQL performs reverse engineering[3] (hereafter re-engineering), i.e. LDW code is processed for extracting knowledge about how the lifting has been conducted. This knowledge is described in terms of annotation overlays on top of the sample API's output document provided by YQL (see Figure 4.12). The rationale is that LDW semantics is frequently limited to a mapping from XML tags (from the API output) to the ontology concepts. If this is the case, SYQL can re-engineer LDW code as a collection of annotations.

As an example, consider the LDW in Figure 4.5. Figure 4.12 shows the output:

- Class-type mapping annotation (window 1): the tag element accounts for a resource (e.g. the *schema:VideoObject* class).

[3]Reverse engineering is "the processes of extracting knowledge or design information from anything man-made and re-producing it or re-producing anything based on the extracted information" [Eil11].



select * from flickr.photos.info where photo_id="27376196615"
and api_key = "4fb031bf5b2f138576d011f

```
"photo": {
  "dateuploaded": "1464
  "farm": "8",
  "id": "27376196615"
  "isfavorite": "0",
  "license": "0",
  "media": "video",
  "originalformat": "jpg"
  "originalsecret": "8a1d
  "rotation": "0",
  "safety_level": "0",
  "secret": "d83fddf185",
  "server": "7411",
  "views": "1",
  "owner": {
```

Resource type definition

Target Class (@type)

Ontology: schema = Schema.org vocabular

Class: schema:VideoObject

Association mapping

ONTOLOGY:PROPERTY: schema = Schema.org vocabular; schema:additionalType

URI PATTERN: Select or write URI pattern...; http://vivoweb.org/ontology/core#{$CLASS}

Preview: {"schema:additionalType" : "http://vivoweb.org/ontology/core#video"}

Property mapping

ONTOLOGY:PROPERTY: schema = Schema.org vocabular; schema:interactionCount

VALUE PATTERN: / /g Regexp in; "1"

Preview: {"schema:interactionCount" : "1"}

Nested resource mapping

ONTOLOGY:PROPERTY: schema = Schema.org vocabular; schema:creator

@type: schema = Schema.org vocabular; schema:Person

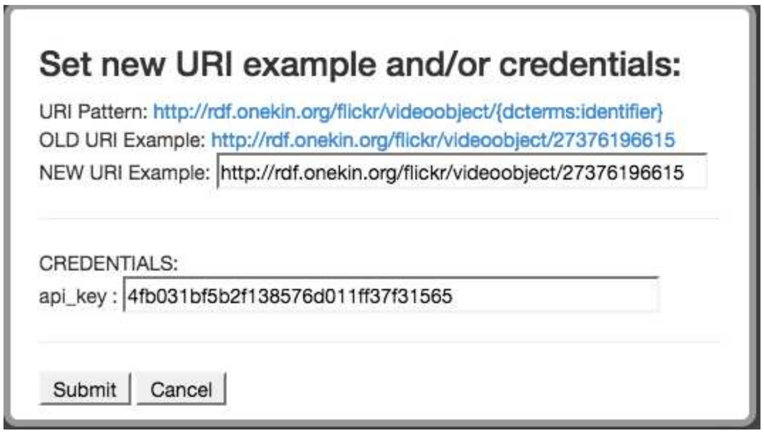
"username" : "iker_azpeitia"; / /g Reg; "iker_azpeitia"

Preview: {"schema:creator" : { "@type" : "schema:Person", "@id" : "http://rdf.onekin.org/flickr/person/iker_azpeitia"}}

Figure 4.12: YQL Console augmented with the *Annotation View* tab

- Property mapping annotation (window 2): the tag element accounts for an RDF property (e.g. the *views* tag is mapped to the *schema :interactionCount* property).

- Association mapping annotation (window 3): the tag element accounts for an RDF association. In this scenario, the XML element supports an interlink to another LD resource. In the example, the photo's <*media*> element is lifted to *VivoWeb* URI: the `http://vivoweb.org/ontology/core#video` is created from the *video* value.



Set new URI example and/or credentials:

URI Pattern: http://rdf.onekin.org/flickr/videoobject/{dcterms:identifier}
OLD URI Example: http://rdf.onekin.org/flickr/videoobject/27376196615
NEW URI Example: http://rdf.onekin.org/flickr/videoobject/27376196615

CREDENTIALS:
api_key : 4fb031bf5b2f138576d011ff37f31565

Figure 4.13: Credentials and *URI Example* curation window

- Nested resource mapping annotation (window 4): the tag element accounts for a resource. As an example, the <*owner*> tag in a <*photo*> stands for a *Person* resource hold in *schema:creator*. This resource holds the name, username and location properties.

The advantage is clear: Figure 4.12 is easier to understand than Figure 4.5. Re-engineering improves the chances of users to understand someone else's code, curate it, and move back to code. So far, re-engineering is limited to LDWs that follow the wrapping template (see Section 4.3.1).

Besides the lifting annotation tool, a form is provided to change the URI Example and credentials. If the API do not supply data it could be because the expiration of the API key or because the example resource (i.e. URI example) is not anymore available (*"API Dimension"* in the Health Checker). Curators can change credential values (e.g. api_key) or example URIs (see Figure 4.13) through a curation window. The LDW is remotely edited with these new values and redeployed in SYQL. The Verification window (see Figure 4.6) will indicate if the wrapper is running again or problems persist.

4.6 Conclusion

SYQL supports LDW creation, deployment, inspection and curation. As for curation, the Health Checker detects data quality as well as API issues in order to highlight LDWs weaknesses. LDW re-engineering for lifting annotations simplify LDW maintenance by reducing programming and annotation barriers. Additionally, a remote credentials and example URI editor ease API related issues curation.

SYQL is a public server on the web (`http://rdf.onekin.org`). In addition, LDWs (`https://github.com/onekin/ldw`) and the SYQL's source code (`https://github.com/onekin/ldwServer`) are freely available in GitHub. Third parties could download and improve them in order to run new instances.

Chapter 5

Validation

"That science is incapable of solving in its own way those fundamental questions is no sufficient reason for slighting them."
– José Ortega y Gasset

5.1 Overview

This chapter evaluates the extent to which SYQL fulfills the requirements for LDW Platforms. Table 5.1 outlines SYQL realization of the requirements for LDW Platforms. All in all, this work's main issue is not so much about LDW definition or quicker resource lookup, but the one of extending LDW lifecycle through curation. The challenge is not about accomplishing the change (after all, the LDW is already there) but the mechanisms available for detecting the change (i.e. the Health Checker) and conducting the change over someone else's code (i.e. the code-to-annotation reverse engineering approach). We are not aware of other approaches that tackle similar issues. Hence, evaluation-by-comparison is not possible. Thus, this dissertation evaluates Quality-in-Use as for the curation perspective. Nevertheless, and for completeness sake, this dissertation also evaluates SYQL from the perspective of producers and consumers.

Table 5.1: SYQL realization of the requirements for LDW Platforms

Stakeholder	**Requirement**	**SYQL Realization**
Producer	Allow for LDW definition	Wrapping template on top of ODT tables
	Allow for LDW deployment	LDWs deployed as YQL services
Consumer	Allow for LDW discovery	LDWs are publicized as "intensional datasets" at the *DataHub* portal
	Allow for LDW lookup	Dereferenceable VoID & Hydra documentation
	Allow for resource lookup	URI dereferencing
Curator	Allow for spotting stains	Health Checker
	Allow for cleaning up stains	Code-to-annotation re-engineering

5.2 Producer Perspective

This evaluation from the producers perspective aims to measure the "quality of solution" when producers have to develop programmatically LDWs, with no programming supporting tools more than the naked YQL console and editor. Two possible scenarios: (1) create an LDW from scratch and (2) take advantage from an existing ODT table.

5.2.1 Measuring Effectiveness

ISO/IEC 25010:2011 [fS11] 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 efficiency (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). 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 efficiency, indicators include task completion time and learning time. In this evaluation, *"task completion time"* is used 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.

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

- Scenario a: From ODT to LDW. Here, subjects were given an existing ODT (i.e. *lastfm.events.getinfo*). The aim was to leverage this ODT to become an 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.

- Scenario b: From API to LDW. Here, students started from scratch,

Table 5.2: 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

i.e. the API (in this case, the *authenticjobs.search* API[1]). 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.

Effectiveness Results. Table 5.2(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,

[1]`http://www.authenticjobs.com/api/documentation/.`

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 5.2(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 *lastfm.events.getinfo* ODT, and next, adapt it to the *authenticjobs'* specifics. This accounts for a collaborative LDW development. 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. A considerable dispersion on the time involved in LDW development is appreciated. The first LDW involved 20' on average while the second took 50' on average. Spend time was proportional to the student's JavaScript experience.

5.3 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 an 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 [Eri13]. 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[2]. The implementation accounts for 250 lines of PHP code.
- as an LDW on top of a YQL's ODT (i. e. ***FlickrODT***). Here, the wrapper was developed and deployed using SYQL infrastructure.

5.3.1 Measuring URI-Dereferencing Latency

This experiment wants to measure the latency introduced by wrapping w.r.t. directly invoking the *Flickr*'s API. To this end, dereferencing was

[2]`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 upgraded the code and installed it in our server.

Table 5.3: Latency average values (ms)

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

Table 5.4: Median latency values based on a number of threads (ms)

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

conducted 1000 times with one call per second. The experiment was repeated three times at different hours of the day. Table 5.3 shows the results. Outcomes indicate that *Flickrwrappr* 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. This dissertation can tentatively conclude that for sparsely used wrappers, ODT indirection might improve maintenance but introduces a time penalty.

5.3.2 Measuring URI-Dereferencing Scalability

This second experiment looks at wrapper behavior with different loads. Here, the experiment subjects the wrappers to different dereferencing petition loads: 10, 50, 250 and 2000 threads. The process was repeated 10 times every 5 seconds. Table 5.4 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.

5.4 Curator Perspective

This section evaluates how successful SYQL is in facilitating users the curation of third-party LDWs. From this perspective, Quality-in-Use becomes paramount. Specifically, the evaluation entails facing subjects with the curation of someone else's LDWs: *flickr.videoobject* (see Figure 4.5). Curation scenarios include [3]:

- API evolution. This might impact both the lowering and lifting of LDWs. Two scenarios are considered:
  - Task 1.1: API key expiration. No data is retrieved from the API. Subjects should update LDW's API key.
  - Task 1.2: API resulting document structure changes. This causes property lifting to stop working. Challenges include recreating the mapping between the attribute and the property.
- Ontology upgrade. This might impact class mappings and property mappings.
  - Task 2.1: Switching *dc:subject* for *dcterms:subject*[4]. Difficul-

[3]Besides API, the LD cloud and data ontologies might also suffer changes. Linked Open Vocabularies (LOV) database [VV] is a case in point. This database stores every different version of a vocabulary over time. For instance, LOV reports 26 different versions of *schema.org*, 10 versions of *FOAF*, 3 *DBpedia ontology* versions or 13 *Dublin Core Metadata* versions. Each version might entail an upgrade on the LDW using the ontology.

[4]This is a real case: Dublin Core refined the *dc* namespace by *dcterms* [Ini13]. The *dcterms:subject* range suggests to use a non-literal value (e.g. `http://dbpedia.org/resource/Spain`) instead of a literal value (e.g. Spain).

ties include transforming an *Property mapping* to an *Association mapping*.

  - Task 2.2: Class definition. Retype RDF resources as *schema: VideoObject*. To increase discoverability and reusability, the use of general purpose ontologies as *de-facto* standards is recommended. In addition, more specific classes into the class hierarchy is recommended too. For instance, *schema:VideoObject* is a subclass of *schema:Media-Object* which is a subclass of *schema:CreativeWork,* and so on. In the sample wrapper, resources are typed as *schema:MediaObject*. Subjects had to type resources as pertaining to *schema:VideoObject* subclass.

- Linked Data cloud evolution. New nodes might enrich existing LDWs with additional interlinkage.

  - Task 3.1: A new interlinkage to a Linked Data cloud node. Let's suppose a consumer is interested in knowing where the videos were taken. The subjects must create a new property (e.g. *schema:locationCreated*) which points to the place where the video has been taken (e.g. `http://linkedgeodata.org/api/3/intersects/Zarautz`). The *LinkedGeoData* service provides information about places [SLHA12].

Next subsection describes the experiment.

5.4.1 Measuring Efficiency and Satisfaction

Measures. The evaluation focuses on two of the Quality-in-Use model characteristics proposed by the ISO/IEC 25010:2011 standard [5]:

[5]The effectiveness, freedom from risk and context coverage has not been evaluated in this experiment.

- Efficiency, which relates to the resources spent in relation to the accuracy and completeness with which users achieve goals. A main indicator of *efficiency* is task completion time.

- Satisfaction, which relates to the degree to which a user is satisfied with their perceived achievement of pragmatic goals, including the results of use and the consequences of use. It was assessed through specifically designed questionnaires measured by attitude rating scales such as SUMI [KC93].

Setting. In order to eliminate differences in the perception of the sample 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 52.0.

Subjects. The experiment was conducted among 12 graduate students applying in a Master in Web Engineering. The majority of participants were male (75%). Regarding age, all participants were in the 22-26 age range. This experiment was realized at the end of 10 hours course in Web Programming issues, where students were familiarized 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 XML and JSON, but not with JSON-LD. Seven students were expert JavaScript programmers and five had basic skills.

Instrument. A questionnaire served to gather users' experience. It consisted of two parts, one to gather the participants' background and one to evaluate *efficiency* and *satisfaction*. In order to measure *efficiency*, participants had to annotate the start time and the finishing time of each task. *Satisfaction* was measured using 7 questions with a 5-point Likert scale (1=completely disagree, 5=completely agree).

Procedure. Before starting, a 45 minute talk was given, introducing

Table 5.5: Time spent on each task (in minutes)

	avg.
Task 1.1 API evolution. Expired credential	0.8
Task 1.2 API evolution. Changed path	6
Task 2.1 Ontology upgrade. Property evolution	2.9
Task 2.2 Ontology upgrade. Class redefinition	1.8
Task 3.1 Cloud evolution. Increase interlinkage	3.2

the purpose. A user-guide sheet was distributed among participants with all this information. Next, subjects were faced with the aforementioned tasks.

Efficiency results. Table 5.5 shows the average time performing each task. The experiment was arranged along the three sources of LDW fragility, namely:

- API evolution. Task 1.1 requires less than one minute on average. It implies changing the API key. Next, Task 1.2. It took 6 minutes on average. Main challenge was to explore the API response on the search for the missing property (as a result of API evolution) within the XML structure.

- Ontology upgrade. Tasks 2.1 and 2.2 involve interacting with the annotation tool to swap properties (i.e. from *dc:subject* to *dcterms: subject*) and class membership (i.e. from *schema:MediaObject* to *schema:VideoObject*), respectively. Subjects spent 2.9' for Task 2.1. and 1.8' for Task 2.2. The reduction in time w.r.t. Task 1.2 (which conceptually is not so different) can be presumably due to now the mapping operates upon already annotated XML elements, fewer in quantity and hence, easier to spot.

- Linked Data cloud expansion. Task 3.1 was twofold: composing a URI out of the object name, and selecting the association that links the annotated resource with a composed URI. This required moving to the lifting annotator, create a new resource from a string (e.g.

Table 5.6: Satisfaction assessment: from 1 (*“total disagreement”*) to 5 (*“total agreement”*)

	avg.
I easily pinpoint to the property I want to annotate	3.5
I easily realize whether properties were annotated or not	3.1
Defining instances types was easy	3.9
Defining property mapping was easy	4.1
Defining association mapping was easy	3.9
Pre-views help fixing mapping errors	3.7
The Semantic View tab is useful	4.2

Zarautz) and select the association (i.e. *schema:locationCreated*).

Satisfaction results. An evaluation questionnaire was prepared to ascertain the satisfaction of subjects in using the annotation facility. This facility is realized through the “Annotation View” and the “Semantic View” tabs in the YQL Console (see Figure 4.12). Table 5.6 displays the results using a Likert scale from 1 (“total disagreement”) to 5 (“total agreement”) for the 12 subjects (S1, S2, etc). The weakest results are obtained for property searching (3.5 avg. points) or the awareness of what is being annotated (3.1 avg. points). This may be due to scalability matters when scrolling large XML documents in search for a given element. Color conventions (i.e. dark blue for unannotated, light blue for annotated) might also be too faint to easily spot what properties have not yet being annotated. By contrast, pop-up windows for setting either resources’ type, property mapping and association mapping are found intuitive enough with 3.9, 4.1 and 3.9 points, respectively. Showing the semantic counterpart for the annotation at hand (i.e. pre-views) was also of interest (3.7 avg. points). In general, the *Semantic View* tab was highly regarded (4.2 avg. points).

5.5 Comparing SYQL with other Platforms

Platforms can serve different aims, and hence, being driven by different requirements. Platform comparison can then be unfair if the requirements

Table 5.7: LDW Platform's requirement compliance

	LDW Def./ Dep.	**LDW Discovery**	**LDW lookup**	**Resource lookup**	**Quality check-ing**	**LDW curation**
LOD Laund.	No (1)	Yes (1)	Yes (4)	Yes (2)	Yes (1)	No
TWC LOGD	Yes (2)	No	Yes (1)	Yes (2)	No	No
xCurator	Yes (3)	No	No	Yes (1, 2)	Yes (1)	No
D2RQ	Yes (3)	No	Yes (1)	Yes (2)	No	No
Virtuoso Sponger	Yes	No	No	Yes (1, 2)	No	Yes
Bio2RDF	Yes	No	Yes (1)	Yes (2)	No	Yes
DBpedia	Yes	No	Yes (1)	Yes (1, 2)	Yes (1)	Yes
SA-REST	Yes	No	No	No	No	No
Karma	Yes (3)	Yes	No	Yes (3)	No	No
SWEET	Yes	Yes	Yes (4)	Yes (3)	No	No
LIDS/LOS	Yes	No	No	Yes (1)	No	No
SYQL	Yes	No (in *DataHub*)	Yes (1,2,3)	Yes (1)	Yes (1, 2, 3)	Yes

of the comparison are not those that drive the platform design. Nevertheless, this comparison is needed to show out the additional contributions, and what is also important, the extent to which existing platforms can embrace the new requirements. This section addresses the extent to which the aforementioned platforms fulfill these requirements.

Table 5.7 holds the output where each requirement admits two values (i.e. *"yes"* or *"no"*) according to these criteria:

- *LDW definition/deployment. Yes*: users can define their own wrappers[6]. *No*: there is no way to define wrappers;
- LDW discovery. *Yes*: facilities are provided to query LDWs[7]. *No*:

[6]Legend: 1-built-in, 2-automatic, 3-semiautomatic.
[7]Legend: 1-datasets.

no query facilities;

- *LDW lookup. Yes*: LDWs are RDF resources[8]. *No*: LDWs are not semantically described;
- *Resource lookup. Yes*: individual resources are dispatched[9]. *No*: resources are not accessible through their URI;
- *Quality checking. Yes*: some quality assessment is conducted[10]. *No*: no quality assessment is performed.
- *LDW* curation. *Yes*: users can enhance someone else's LDWs. *No*: users can only enhance their own LDWs, if any.

None of the listed systems cover all the requirements. Systems aim constraints the requirements their fulfill. *LOD Laundromat* is a fully automatized RDF to RDF datasets cleaner. Hence, users cannot define or maintain wrappers. *SA-REST* consumes wrappers and data into a proxy server so they are not publicly provided nor validated. *SWEET* and *LIDS/LOS* focus on APIs semantic description. Issues of quality or maintenance are not tackled. *Virtuoso Sponger* and *Karma* allow to create wrappers but do not focus on quality and maintenance. *D2RQ* is a server to be locally installed, therefore it ignores discoverability. *SYQL* has been mainly influenced by five developments: *Bio2RDF*, *xCurator, DBpedia*, *TWC LOGD* and *Karma*. Next, this dissertation provides a deeper comparison.

Bio2RDF shares the vision of an open community of wrapper producers. It allows producers to program in their preferred programming language which lowers technological barriers but complicates reusability. By contrast, SYQL aims to promote both LDW sharing and the engagement of the API community. Its declarativeness and popularity among API programmers make YQL's ODTs this dissertation's bet. In addition, SYQL

[8]Legend: 1-VoID (dataset description), 2-Hydra (APIs documentation), 3- DQV (data quality), 4-metadata.

[9]Legend: 1-dereferencing URIs, 2-SPARQL endpoint, 3-ad-hoc dereferenciation.

[10]Legend: 1-intrinsic, 2-accessibility, 3-contextual.

provides an re-engineering and annotation tool to engage consumers in curating wrappers.

xCurator offers a semiautomatic wrapper development while the maintenance process gathers consumers' feedback. The main difference with SYQL lies in openness. SYQL is totally open: everybody can create and curate wrappers. By contrast, in *xCurator,* data consumers can report data problems but only administrators can curate wrappers.

DBpedia wrappers are syntactically validated whereas generated data is assessed by selected consumers detecting and reporting errors [AZS$^+$16, KZAL13]. Users can ask for edition rights in order to curate wrappers [DBp16]. The main difference stems from *DBpedia* being Wikipedia-specific while SYQL is agnostic.

TWC LOGD also faces upgrading but with a different approach. Upgrades are incrementally created adding new properties. That is, if there are *n* different upgrades, there will be *n* different wrappers. In that way, each consumer can pick up his favorite version. By contrast, SYQL only keeps a single wrapper version, though producers can resort to *GitHub*'s version control to create new LDWs out of previous versions.

Karma also addresses API-based LDWs. Both *Karma* and SYQL resort to annotations. However, *Karma* illustrates a generative endeavor (from annotations to code) whereas SYQL is a re-engineering effort (from code to annotations). This difference stems from the different targeted audiences: *Karma* targets LDW *producers* whereas SYQL aims at helping *curators* in cleaning someone else's LDWs.

5.6 Conclusion

YQL ODTs (a combination of XML declarative language and Javascript program) are a powerful mechanism to access APIs and, by extension, to create LDWs. Lessons learned from the Producers perspective encouraged us to provide programming libraries and templates to easy LDWs development. Even more, the *Annotator tool* is designed not only for LDW curation but for LDWs creation too. The aim is to lower the needed programming and semantic knowledge skills in LDW creation and curation. In fact, Curators upgrade LDWs not developed by themselves. The re-engineering simplifies maintenance tasks by showing code as annotations.

The SYQL platform (through the YQL system) manages well a number of concurrent calls. However, Consumers must to take into account the latency introduced by LDWs. LDWs do not fit well for LD Applications requiring very fast responses but are suitable for other kind of LD Applications where one second latency is acceptable.

To sum it up, SYQL and LDWs promote collaborative behaviors:

- Producers are more effective cloning ODTs in order to define new LDWs,
- Consumers take advantage of online LDWs on top of the YQL system, and
- Curators are able to maintain LDWs developed by others.

Chapter 6

Writing in the LD Cloud

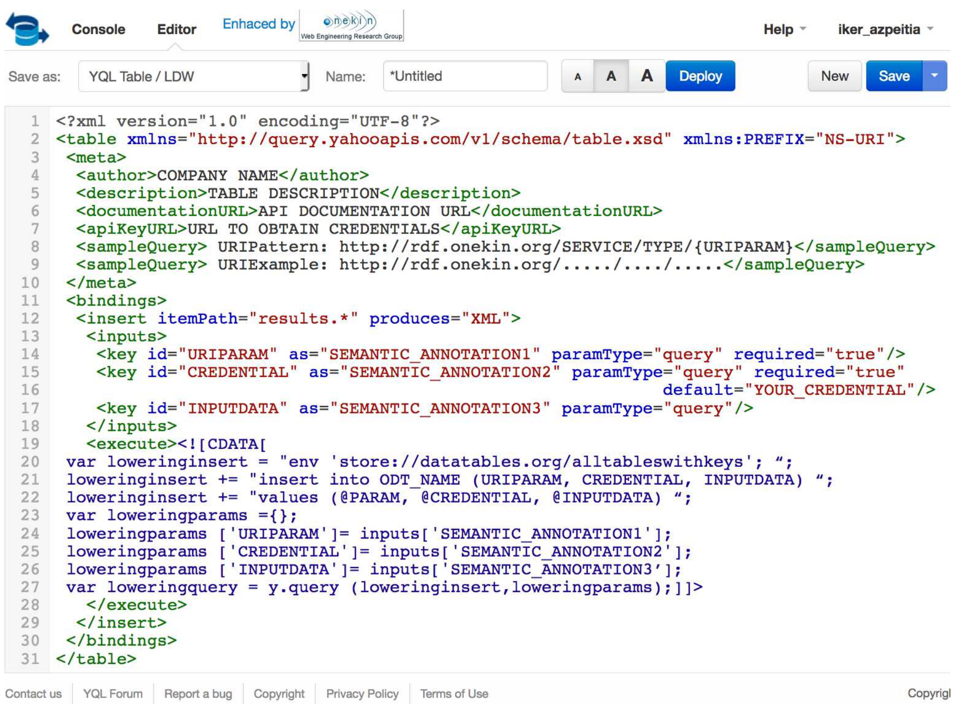
“Whether he be an original or a plagiarist, man is the novelist of himself.”
– José Ortega y Gasset

6.1 Overview

LDWs support different operations although, so far, only resource lookup (GET method) has been shown. In this chapter resource insertion operation (POST method) is discussed. This dissertation advocates LDW curation to increase LDW lifespan. Curation is not influenced by operations supported by the LDW. However, tools and procedures could differ depending on the operations.

6.2 Definition

The insertion operation is based on the YQL *INSERT* statement. For example, the following statement creates a new blog post in the `http://oscaronekin.wordpress.com` *Wordpress* blog:



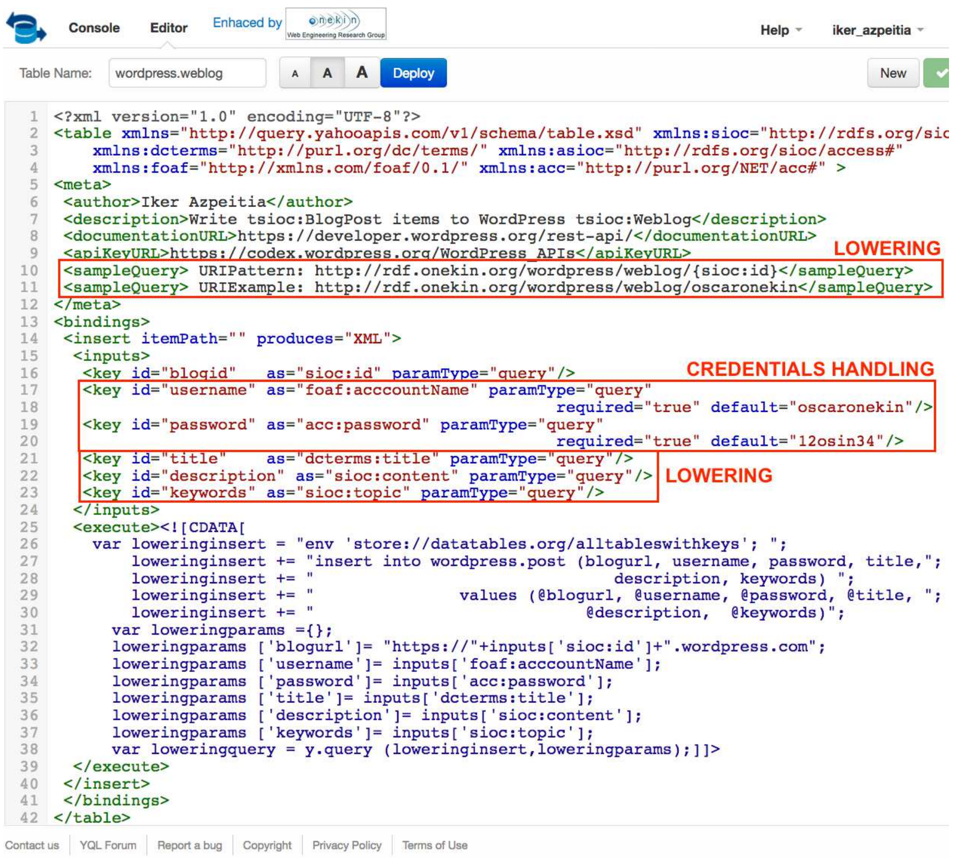
```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <table xmlns="http://query.yahooapis.com/v1/schema/table.xsd" xmlns:PREFIX="NS-URI">
 3  <meta>
 4   <author>COMPANY NAME</author>
 5   <description>TABLE DESCRIPTION</description>
 6   <documentationURL>API DOCUMENTATION URL</documentationURL>
 7   <apiKeyURL>URL TO OBTAIN CREDENTIALS</apiKeyURL>
 8   <sampleQuery> URIPattern: http://rdf.onekin.org/SERVICE/TYPE/{URIPARAM}</sampleQuery>
 9   <sampleQuery> URIExample: http://rdf.onekin.org/...../..../.....</sampleQuery>
10  </meta>
11  <bindings>
12   <insert itemPath="results.*" produces="XML">
13    <inputs>
14     <key id="URIPARAM" as="SEMANTIC_ANNOTATION1" paramType="query" required="true"/>
15     <key id="CREDENTIAL" as="SEMANTIC_ANNOTATION2" paramType="query" required="true"
16                                                          default="YOUR_CREDENTIAL"/>
17     <key id="INPUTDATA" as="SEMANTIC_ANNOTATION3" paramType="query"/>
18    </inputs>
19    <execute><![CDATA[
20  var loweringinsert = "env 'store://datatables.org/alltableswithkeys'; “;
21  loweringinsert += "insert into ODT_NAME (URIPARAM, CREDENTIAL, INPUTDATA) “;
22  loweringinsert += "values (@PARAM, @CREDENTIAL, @INPUTDATA) “;
23  var loweringparams ={};
24  loweringparams ['URIPARAM']= inputs['SEMANTIC_ANNOTATION1'];
25  loweringparams ['CREDENTIAL']= inputs['SEMANTIC_ANNOTATION2'];
26  loweringparams ['INPUTDATA']= inputs['SEMANTIC_ANNOTATION3'];
27  var loweringquery = y.query (loweringinsert,loweringparams);]]>
28    </execute>
29   </insert>
30  </bindings>
31 </table>
```

Figure 6.1: An LDW template with the insert operation to be completed

insert into wordpress.post (blogurl, username, password, title, description, tags) values ('http://oscaronekin.wordpress.com', 'oscaronekin', '12osin34', 'Demo video', 'https://www.flickr.com/...', 'research')";

SYQL also provides an **"INSERT wrapping template"** (see Figure 6.1). For resource insertion a YQL *INSERT* statement is used (lines 20-22). Here, the template accounts for two main steps: lowering and credentials handling (see Figure 6.2). It is worth to note that this LDW lacks the lifting step since the resource insertion process does not retrieve data from the API to be lifted.

Lowering. In the insertion operation, lowering involves not only to match the URI pattern (line 10 in Figure 6.2) and URI examples (line 11) but also to send RDF data to the API. Input data is annotated defining the properties of the input RDF resource (lines 21-23). These properties



```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <table xmlns="http://query.yahooapis.com/v1/schema/table.xsd" xmlns:sioc="http://rdfs.org/sic
 3     xmlns:dcterms="http://purl.org/dc/terms/" xmlns:asioc="http://rdfs.org/sioc/access#"
 4     xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:acc="http://purl.org/NET/acc#" >
 5 <meta>
 6  <author>Iker Azpeitia</author>
 7  <description>Write tsioc:BlogPost items to WordPress tsioc:Weblog</description>
 8  <documentationURL>https://developer.wordpress.org/rest-api/</documentationURL>
 9  <apiKeyURL>https://codex.wordpress.org/WordPress_APIs</apiKeyURL>
10  <sampleQuery> URIPattern: http://rdf.onekin.org/wordpress/weblog/{sioc:id}</sampleQuery>
11  <sampleQuery> URIExample: http://rdf.onekin.org/wordpress/weblog/oscaronekin</sampleQuery>
12 </meta>
13 <bindings>
14  <insert itemPath="" produces="XML">
15   <inputs>
16    <key id="blogid"   as="sioc:id" paramType="query"/>
17    <key id="username" as="foaf:acccountName" paramType="query"
18                                               required="true" default="oscaronekin"/>
19    <key id="password" as="acc:password" paramType="query"
20                                               required="true" default="12osin34"/>
21    <key id="title"    as="dcterms:title" paramType="query"/>
22    <key id="description" as="sioc:content" paramType="query"/>
23    <key id="keywords" as="sioc:topic" paramType="query"/>
24   </inputs>
25   <execute><![CDATA[
26     var loweringinsert = "env 'store://datatables.org/alltableswithkeys'; ";
27         loweringinsert += "insert into wordpress.post (blogurl, username, password, title,";
28         loweringinsert += "                                    description, keywords) ";
29         loweringinsert += "                        values (@blogurl, @username, @password, @title, ";
30         loweringinsert += "                                 @description,  @keywords)";
31       var loweringparams ={};
32       loweringparams ['blogurl']= "https://"+inputs['sioc:id']+".wordpress.com";
33       loweringparams ['username']= inputs['foaf:acccountName'];
34       loweringparams ['password']= inputs['acc:password'];
35       loweringparams ['title']= inputs['dcterms:title'];
36       loweringparams ['description']= inputs['sioc:content'];
37       loweringparams ['keywords']= inputs['sioc:topic'];
38       var loweringquery = y.query (loweringinsert,loweringparams);]]>
39   </execute>
40  </insert>
41  </bindings>
42 </table>
```

Figure 6.2: YQL Editor Console. The *wordpress.weblog* LDW

are sent to the API through pattern matching (e.g. line 21 to line 35 *{dcterms:title}* binding). The execution part (lines 25-39) allows to compose a YQL INSERT statement in order to create the resource. Note that this statement is an indirection to the *wordpress.post* ODT (lines 26-30).

Credentials handling. In the current example, the username (lines 17-18) and the password (lines 19-20) are credentials. Credentials are managed in the same way in all the operations. However, looked up resources usually are publicly accessible and hence credentials (e.g. API key) allow to manage requests on the server (e.g. preventing abuse of the API). In contrast, inserted resources are injected into users' accounts, therefore websites require personal credentials (e.g. password) in order to control

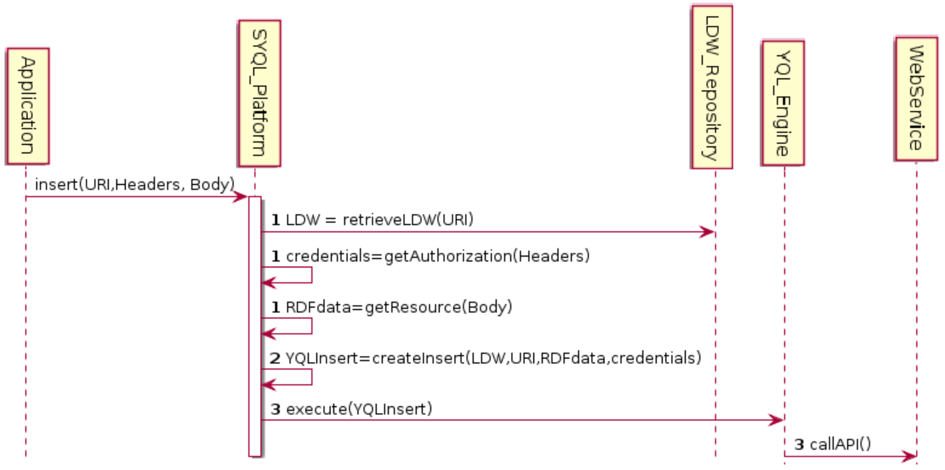

Application
SYQL_Platform
LDW_Repository
YQL_Engine
WebService
insert(URI,Headers, Body)
1 LDW = retrieveLDW(URI)
1 credentials=getAuthorization(Headers)
1 RDFdata=getResource(Body)
2 YQLInsert=createInsert(LDW,URI,RDFdata,credentials)
3 execute(YQLInsert)
3 callAPI()

Figure 6.3: Resource creation sequence diagram

access and identify the account. LDW creators, for sure, will not provide valid data for these credentials, otherwise their own private data would be not secure.

6.3 Deployment

When an LDW with an insert operation is deployed, only syntactic errors are detected (except the lack of lifting *function*). Execution is not tested because an example input RDF is not provided and the aforementioned cautions with the personal credentials.

6.4 Resource Insertion

SYQL processes POST methods for URIs that conform to the LDW's *URI-Pattern*. Resource insertion follows three main tasks (see Figure 6.3):

1. data retrieval, where the LDW wrapper is downloaded from the LDW repository. In addition, credentials and the RDF data are extracted from the request body;

2. lowering, where the YQL insert statement is prepared with the RDF data and the credentials; and

3. API calling, where the insert statement is enacted resulting in a new resource in the web service.

As said before, the insertion operations access personal accounts, hence, personal credentials are provided in the request body as well as the RDF data. For example, the POST method in order to create a blog post in *Wordpress* may provide this data:

> *POST* `http://rdf.onekin.org/wordpress/weblog/oscaronekin`
>
> *Authorization = {"hydra:supportedProperty": [{"hydra:title": "foaf:accountName", "schema:value": "oscaronekin"}, {"hydra:title": "acc:password", "schema:value": "12osin34"}]}*
>
> *{"@context":{"sioc": "http://rdfs.org/sioc/ns#", "tsioc": "http://rdfs.org/sioc/types#", "dcterms":"http://purl.org/dc/terms/"}, "@type":"tsioc:BlogPost", "sioc:topic":"research", "dcterms:title": "Demo video", "sioc:content": "https://www.flickr.com/photos/..."}*

6.5 Spotting Stains

Stains in LDW

Quality of insertion operations is checked based on real executions. That is, each time an application inserts a resource, data is gathered to asses the operation's quality. So, SYQL assesses quality based on the last 10 insertions. The quality features that are valid for insert operations are listed in Table 6.1. They are a subset of those for the lookup operation.

Functioning-status Stains. Although performance quality is not as critical for insert operation as it is for lookup, the latency (P2), throughput

Table 6.1: Quality dimensions for the writing operation. *"Abr"* stands for the abbreviation used in [ZRM+16]

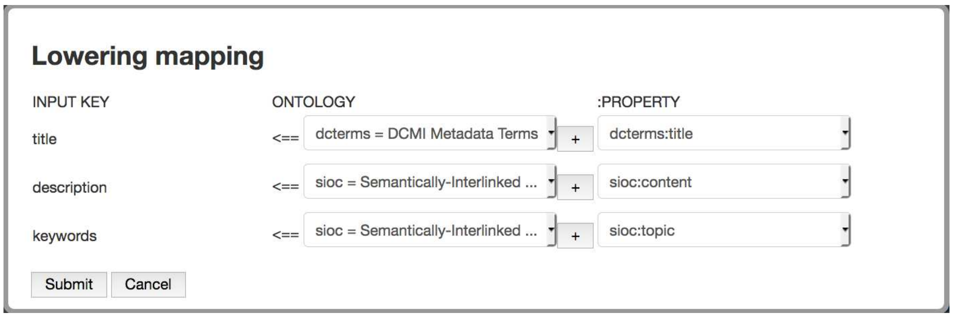
Dimension	Subdimension	Abr	Metric	SYQL realization
Accessibility	Performance	P2	Low latency	Minimum request to response delay
		P3	High throughput	Number of requests per second
		P4	Scalability of a data source	Average throughput of the last ten calls
Intrinsic	Semantic accuracy	SA2	No inaccurate values	Notifications via *GitHub* comments
Contextual	Trustworthiness	T7	Reputation of the dataset	Number of insertions and ratings in *GitHub*

(P3) and scalability (P4) quality features are registered. In addition, the expiration of credentials is detected if the LDW returns an HTTP error status.

Data-quality Stains. Resources inserted in sites are hidden for SYQL. Hence, data-quality stains are reported by stakeholders by GitHub. Concretely, consumers report comments about inaccurate data creation (SA2) (e.g. incorrectly matched *dcterms:title* to the resource description) or their subjective assessment (T7).

6.6 Cleaning up Stains

Re-engineering is applied for lowering annotations too. Mappings lower the RDF data in the request's body to the YQL insert statement. Accordingly, the Health Checker offers a lowering annotation tool to modify these mappings. For example, Figure 6.4 shows the lowering annotator for mappings in Figure 6.2 (lines 21-23).



Lowering mapping

INPUT KEY		ONTOLOGY		:PROPERTY
title	<==	dcterms = DCMI Metadata Terms	+	dcterms:title
description	<==	sioc = Semantically-Interlinked ...	+	sioc:content
keywords	<==	sioc = Semantically-Interlinked ...	+	sioc:topic

Figure 6.4: Lowering mapping editor

6.7 Conclusion

LDWs are enhanced YQL ODTs, hence, they could support the four CRUD operations since YQL offers SELECT, INSERT, UPDATE and DELETE statements. So far, SYQL supports lookup and resource insertion operations. Curation tools are adapted for each operation (e.g. lowering mapping vs. lifting mapping) but LDWs are uniformly managed no matter the operations they support.

Chapter 7

Proof of Concept

"The metaphor is perhaps one of man's most fruitful potentialities. Its efficacy verges on magic, and it seems a tool for creation which God forgot inside one of His creatures when He made him. All our other faculties keep us within the realm of the real, of what is already there. The most we can do is to combine things or to break them up. The metaphor alone furnishes an escape; between the real things, it lets emerge imaginary reefs, a crop of floating islands. A strange thing, indeed, the existence in man of this mental activity which substitutes one thing for another — from an urge not so much to get at the first as to get rid of the second."

– José Ortega y Gasset

7.1 Overview

Section 2.4 describes four scenarios (i.e. CMS, TAS, Linked Data visualizers and Semantic Mashups) where LDWs are useful. This chapter delves into one of them, TABASCO, a Linked Data Application that is going to be used to check out SYQL.

7.2 TAg-BASed inter-site COmmunication

Tagging is an important task for different systems and services such as *Diigo* [Dii], *WordPress* or *Flickr* which allow participants to annotate a particular resource (e.g. a web page, a blog post, an image) with a freely chosen set of keywords (aka tags). Tags can be a powerful tool for social navigation [MF06], helping people to share and discover new information contributed by other community members. Notice however that such collaboration is restricted to the site itself. Collaboration-wise, these websites behave as islands where collaboration is restricted to resources and users within the website walls.

However, it is very common for users to keep an account in distinct tagging sites depending on a broad range of issues: the resource type, the utilities offered by the site, the supporting community, confidentiality, etc. Therefore, taggable resources will most likely be scattered throughout the Web. The potential synergies among many sites, communities, and services are expensive to exploit, and their data are difficult and cumbersome to link and reuse. The main reason for this lack of interoperation is that for the most part in the Social Web, common standards still do not exist for knowledge and information exchange and interoperation.

This chapter introduces a framework for *TAg-BASed, inter-site COmmunication* (*TABASCO*). The system permits users to communicate seamlessly through heterogeneous websites. Users are represented through their website accounts. Tasks are those set by the websites themselves, and normally available through an API. Tags are the means to denote the message that enacts the associated task in the target account (hereafter referred to as "reactive tags"). Messages are originated in *the sender website* and impact on *the receiver website*. Finally, web resources (e.g. bookmarks, blog posts, etc) stand for message parameters.

As an example, let *U1* and *U2* be two users that hold an account in *Flickr* and *Wordpress*, respectively. *Flickr* keeps videos, and supports tagging. *Wordpress* manages blog posts, and permits to file and categorize posts. In this example, *Flickr* and *Wordpress* will play the role of the sender and receiver sites, respectively. *U1* wants to communicate to *U2* when an interesting video is worth to be shared. To this end, *U1* tags the interesting video in *Flickr* as *"toshare"*. This tag is a reactive tag, i.e. its reactive semantic has been previously defined in *TABASCO* by *U1* provided he holds *U2* authorization. This makes *TABASCO* monitor *U1* tagging behaviour in *Flickr*. When *"toshare"* is used, *TABASCO* enacts its associated semantics: creating a new post in *U2*'s *Wordpress* account.

The previous scenario illustrates the notion of *Collaboration Space* as a graph of nodes (i.e. user accounts), and labelled edges (i.e. reactive tags). Edges introduce collaboration paths whereby tagging on the source node triggers some site-dependent reaction on the target node.

TABASCO aims at binding disperse communities together. From a communication perspective, the approach accounts for uniformity and site independence. So far, communication is provided within the site's boundaries through distinct mechanisms: button, command lines or even tags (for instance, the so-called machine tags in *Delicious*, e.g. *for:Jon*). *TABASCO* uses reactive tags for messaging along no matter the website. Any website supporting tagging is liable to use reactive tags. From the website perspective, reactive tags are just standard tags. It is *TABASCO* monitoring what makes the tag be reactive. Reactive semantics is specified using Event-Condition-Action (ECA) rules. TABASCO looks up a sender site to retrieve the item list, then retrieves the new items' descriptions to check if they hold the reactive tag (the event) and, if it is the case, creates derived items in the receiver site (the action). Thus, a site (e.g. Wordpress) is a container (e.g. a weblog) of items (e.g. blog posts). Containers accommodate items that hold tags.

Social Web is fed with User Generated Data [KDN08]. SYQL provides API data as Linked Data. So, SYQL could supply User Generated Data that is otherwise not available as Linked Data. Semantic data helps TABASCO define reactive tags. In addition, SYQL supports the following key requirements identified for TABASCO:

- **interoperability**. TABASCO provides an additional layer on existing tagging systems. This brings issues on both syntactic interoperability (such as data formats and communication protocols) and semantic interoperability (e.g. existence of a shared reference model).

  - Syntactic interoperability. LDWs hide API intricacies into YQL ODTs which allows SYQL to lower URI calls to YQL statements. In this way, TABASCO interacts with APIs as if they were LD data sites on the cloud. Moreover, APIs evolution would be detected and solved in SYQL's Health Checker without consequences for TABASCO.

  - Semantic interoperability. Smooth system interoperation is achieved by abstracting site specifics through ontologies. User accounts, websites and the semantics of reactive tags are all captured by adapting existing ontologies, namely, FOAF, SIOC and ECA-ML, respectively. LDWs map API data into ontology terms.

- **integrity**. Integrity is the assurance that the information can only be accessed or modified by those authorized to do so. TABASCO extends tagging consequences outside a single user account. Tagging on one user account might impact someone else's user account. Users should keep control of who and how their accounts are accessed. Hydra documents in SYQL describe credentials required for each LDW. TABASCO interprets these descriptions and asks users for credential values. Thus, TABASCO manages user authorizations and consumer credentials which take precedence over the LDW producer credentials.

7.3 Adding a New Site

TABASCO does not require any plugin on participating sites. Sender sites need to provide tagging capabilities. Both sender and receiver sites should be LD sites.TABASCO administrators add a new site (e.g. *Wordpress*) specifying the site name, the Hydra description of the items lookup service and the Hydra description of the container service (the service that storages the items). It is worth to note that although TABASCO is intended to test SYQL it could use services in the LD cloud as long as they provide Hydra descriptions. In any case, TABASCO administrators need to find a Linked Data source and assess its adequacy. To this end, administrators would search the LD source in DataHub, the referral dataset directory. Next, they would inspect source's metadata and the supported items to decide whether it is appropriate. SYQL assist administrators by providing a rich metadata (i.e. quality measures in the Health Checker) and by allowing to modify the item to provide required semantic properties (i.e. re-engineering tools). Ultimately, if the site's LD source does not exist administrators can define and deploy a LDW for the site.

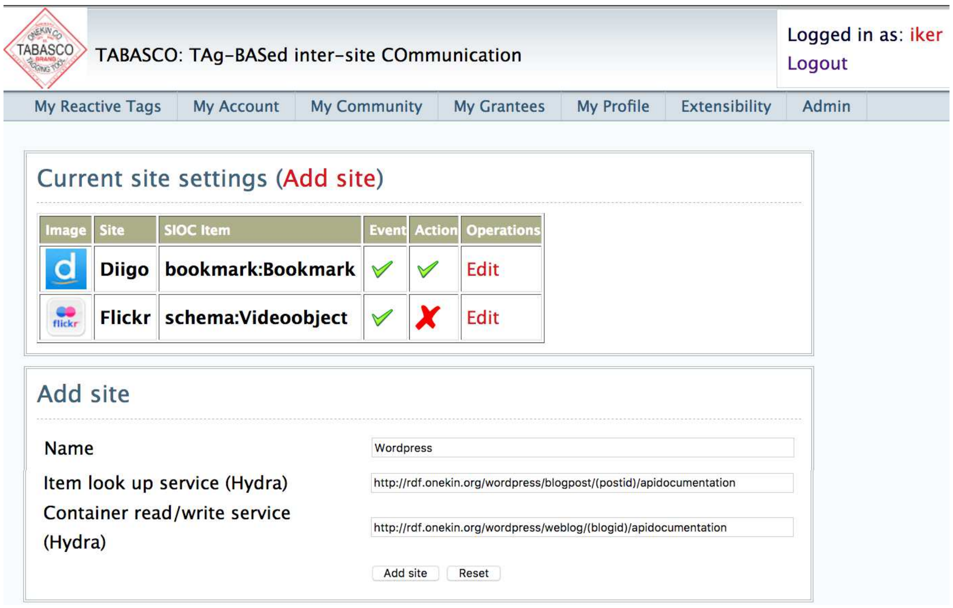
In the example form in Figure 7.1 a TABASCO administrator specifies the *Wordpress* site by means of the *wordpress.weblog* LDW's Hydra description[1] (the items container) and the *wordpress.blogpost* LDW's Hydra description[2] (the items lookup service). The form also lists the sites specified so far. For example, the *Flickr* site cannot be used in the action part of the ECA rules because it does not support the resource insertion operation (i.e. POST). In contrast, the *wordpress.weblog* LDW[3] supports resource lookup and resource insertion operations and, hence, *Wordpress* could be used in rules' event and action parts.

Hydra descriptions reveal how to interact with sites by means of (1) an

[1]Available at `http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation`.

[2]Available at `http://rdf.onekin.org/wordpress/blogpost/(postid)/apidocumentation`.

[3]Available at `https://raw.githubusercontent.com/onekin/ldw/master/wordpress/wordpress.weblog.xml`.



Current site settings (Add site)

Image	Site	SIOC Item	Event	Action	Operations
	Diigo	bookmark:Bookmark	✓	✓	Edit
	Flickr	schema:Videoobject	✓	✗	Edit

Add site

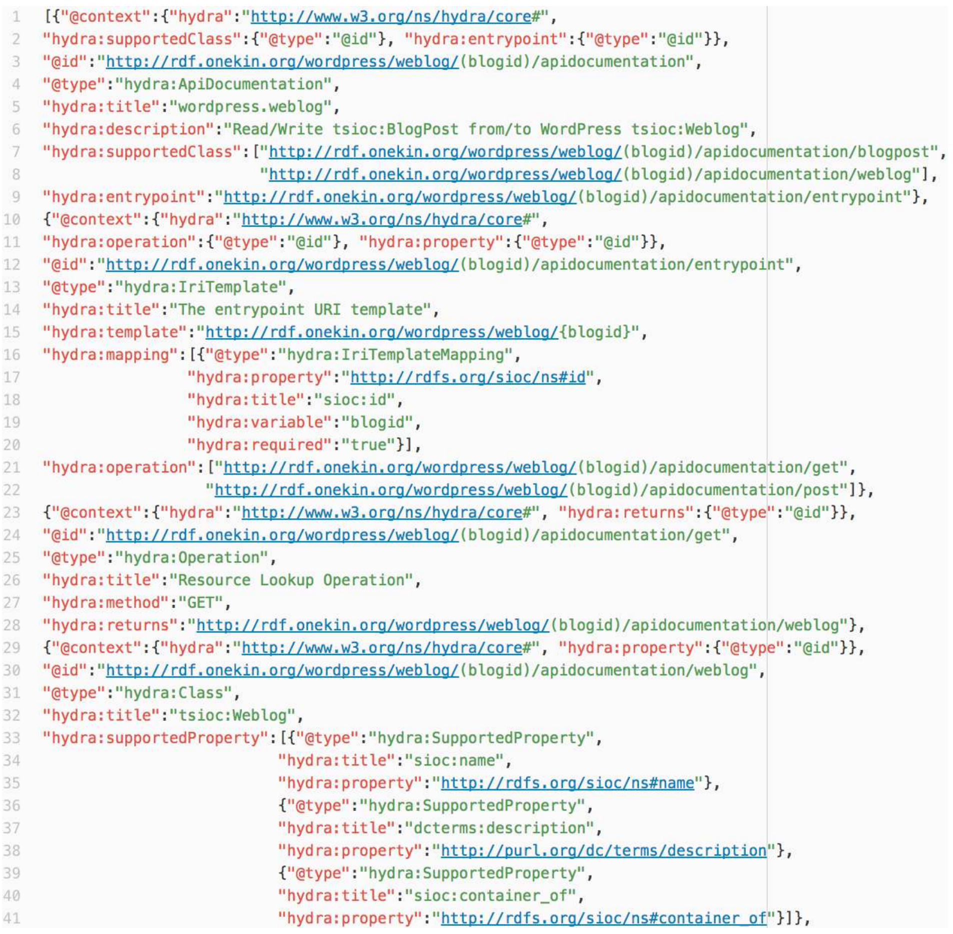
Name	Wordpress
Item look up service (Hydra)	http://rdf.onekin.org/wordpress/blogpost/{postid}/apidocumentation
Container read/write service (Hydra)	http://rdf.onekin.org/wordpress/weblog/{blogid}/apidocumentation

Figure 7.1: Adding a new site to TABASCO

entry point URI pattern, (2) the supported operations and (3) the required credentials. The entry point (lines 10-22 in Figure 7.2) lists the supported operations (lines 21-22) as well as the URI pattern (line 15) and the variable(s) to be expanded (e.g. blogid) in order to obtain a correct container's URI (lines 16-20). Current example supports both GET (resource lookup) and POST (resource insertion) operations. Other LDWs could only support one of them.

The GET operation (lines 23-28) returns *tsioc:Weblog* items (line 28) with these properties: *sioc:name* (lines 33-35), *dcterms:description* (lines 36-38) and *sioc:container_of* (lines 39-41). *sioc:container_of* holds the list of contained items (e.g. blog posts).

The POST operation (lines 42-48 in Figure 7.3) sets credentials (lines 62-75) and data required to create *tsioc:BlogPost* items (lines 49-61). The blog posts should contain the *sioc:topic* (lines 53-55), *dcterms:title* (lines 56-58) and *sioc:content* (lines 59-61) properties. As for credentials, the *acc:password* (lines 66-70) and the *foaf:accountName* (lines 71-75) are required. Note that the POST operation requires credentials and the GET



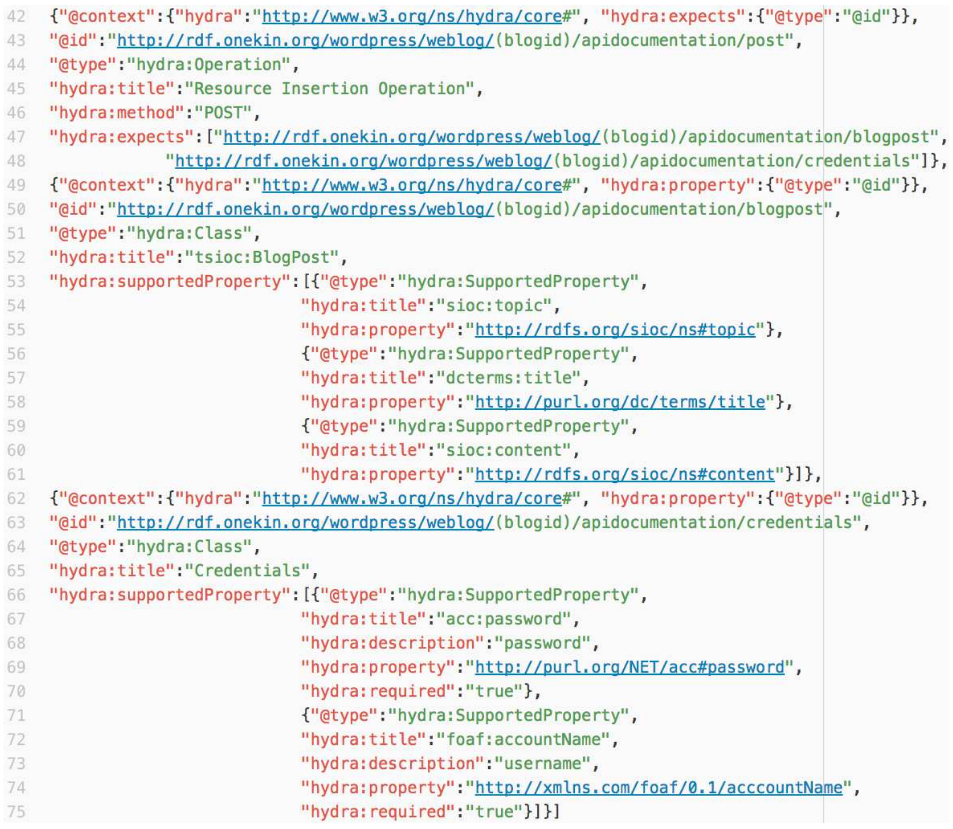
```
[{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#",
"hydra:supportedClass":{"@type":"@id"}, "hydra:entrypoint":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation",
"@type":"hydra:ApiDocumentation",
"hydra:title":"wordpress.weblog",
"hydra:description":"Read/Write tsioc:BlogPost from/to WordPress tsioc:Weblog",
"hydra:supportedClass":["http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/blogpost",
                        "http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/weblog"],
"hydra:entrypoint":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/entrypoint"},
{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#",
"hydra:operation":{"@type":"@id"}, "hydra:property":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/entrypoint",
"@type":"hydra:IriTemplate",
"hydra:title":"The entrypoint URI template",
"hydra:template":"http://rdf.onekin.org/wordpress/weblog/{blogid}",
"hydra:mapping":[{"@type":"hydra:IriTemplateMapping",
                "hydra:property":"http://rdfs.org/sioc/ns#id",
                "hydra:title":"sioc:id",
                "hydra:variable":"blogid",
                "hydra:required":"true"}],
"hydra:operation":["http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/get",
                  "http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/post"]},
{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#", "hydra:returns":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/get",
"@type":"hydra:Operation",
"hydra:title":"Resource Lookup Operation",
"hydra:method":"GET",
"hydra:returns":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/weblog"},
{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#", "hydra:property":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/weblog",
"@type":"hydra:Class",
"hydra:title":"tsioc:Weblog",
"hydra:supportedProperty":[{"@type":"hydra:SupportedProperty",
                          "hydra:title":"sioc:name",
                          "hydra:property":"http://rdfs.org/sioc/ns#name"},
                          {"@type":"hydra:SupportedProperty",
                          "hydra:title":"dcterms:description",
                          "hydra:property":"http://purl.org/dc/terms/description"},
                          {"@type":"hydra:SupportedProperty",
                          "hydra:title":"sioc:container_of",
                          "hydra:property":"http://rdfs.org/sioc/ns#container_of"}]},
```

Figure 7.2: *Wordpress* entry point and GET operation descriptions

operation does not. It is aligned with the LDW definition of the insert and the select part[4]. Creating an item in a user's blog requires authorization whilst reading public posts is always allowed.

To sum up, Hydra allows TABASCO to retrieve and manage site information. Concretely, the items' properties, the supported operations and the required credentials are described. In this way TABASCO configures itself without administrators interaction. Administrators only have to dis-

[4]The wordpress.weblog LDW is available at `https://raw.githubusercontent.com/onekin/ldw/master/wordpress/wordpress.weblog.xml`.



```
{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#", "hydra:expects":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/post",
"@type":"hydra:Operation",
"hydra:title":"Resource Insertion Operation",
"hydra:method":"POST",
"hydra:expects":["http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/blogpost",
             "http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/credentials"]},
{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#", "hydra:property":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/blogpost",
"@type":"hydra:Class",
"hydra:title":"tsioc:BlogPost",
"hydra:supportedProperty":[{"@type":"hydra:SupportedProperty",
                           "hydra:title":"sioc:topic",
                           "hydra:property":"http://rdfs.org/sioc/ns#topic"},
                           {"@type":"hydra:SupportedProperty",
                           "hydra:title":"dcterms:title",
                           "hydra:property":"http://purl.org/dc/terms/title"},
                           {"@type":"hydra:SupportedProperty",
                           "hydra:title":"sioc:content",
                           "hydra:property":"http://rdfs.org/sioc/ns#content"}]},
{"@context":{"hydra":"http://www.w3.org/ns/hydra/core#", "hydra:property":{"@type":"@id"}},
"@id":"http://rdf.onekin.org/wordpress/weblog/(blogid)/apidocumentation/credentials",
"@type":"hydra:Class",
"hydra:title":"Credentials",
"hydra:supportedProperty":[{"@type":"hydra:SupportedProperty",
                           "hydra:title":"acc:password",
                           "hydra:description":"password",
                           "hydra:property":"http://purl.org/NET/acc#password",
                           "hydra:required":"true"},
                           {"@type":"hydra:SupportedProperty",
                           "hydra:title":"foaf:accountName",
                           "hydra:description":"username",
                           "hydra:property":"http://xmlns.com/foaf/0.1/acccountName",
                           "hydra:required":"true"}]}]
```

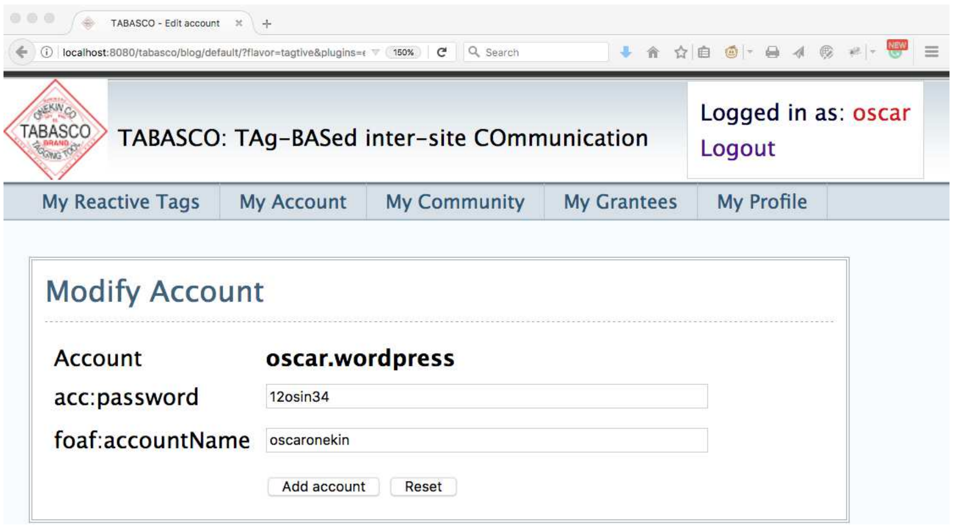
Figure 7.3: POST operation description

cover and adapt appropriate LDWs in *DataHub* or create them in order to fulfill data requirements.

7.4 Credentials Management

Hydra describes credentials for accessing user accounts. For example, the Wordpress site definition in Figure 7.3 indicates that the POST operation requires the *acc:password* and *foaf:accountName* credentials. TABASCO automatically asks users for this credentials without administrator involvement. Figure 7.4 shows the form collecting credentials in order to authorize TABASCO to operate on behalf of Oscar.

TABASCO behaves as a “guarantor” of credentials. Users *grant/request*



Modify Account

Account	oscar.wordpress
acc:password	12osin34
foaf:accountName	oscaronekin

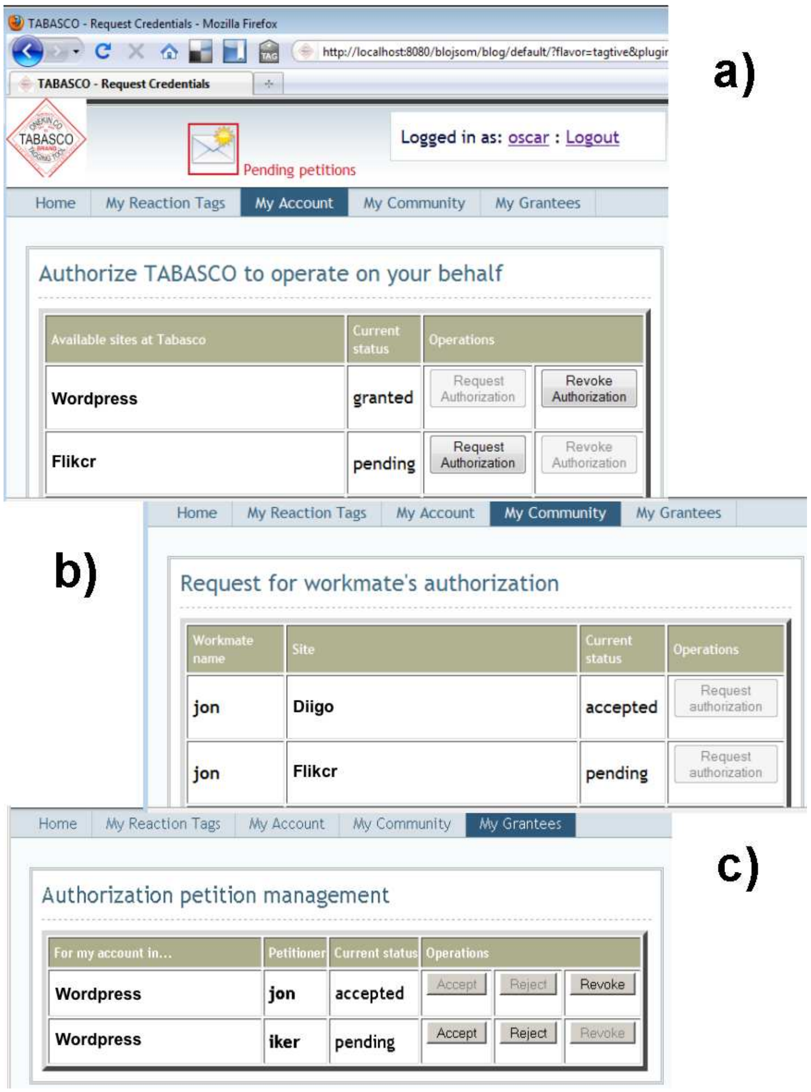
Figure 7.4: Account form requesting required data

authorization tokens to/from TABASCO. This requires the previous consent from the authorization owner. Figure 7.5 outlines the main TABASCO graphic user interfaces for this purpose.

Registration (*"My Account"* tab: Figure 7.5(a)). Users first indicate whether their accounts will become nodes of the *Collaboration Space*. The process goes as follows: (1) the user selects the website (e.g. *Wordpress*), (2) TABASCO asks user for credentials to authorize TABASCO to work on his or her account (e.g. user and password), (3) TABASCO checks whether credentials are valid, and (4) the user account is registered for TABASCO to work on this account.

Authorization request (*"My Community"* tab: Figure 7.5(b)). Even if TABASCO holds an authorization, this does not imply that any registered user can enjoy this authorization. Rather, defining reactive tags over a user account requires authorization privileges upon this account. The petition lifecycle goes along the following stages: start, pending, accepted/rejected and revoked.

Authorization grantee (*"My Grantees"* tab: Figure 7.5(c)). Authorization petitions are managed by account owners themselves. Petitions are


a)
TABASCO - Request Credentials - Mozilla Firefox
http://localhost:8080/blojsom/blog/default/?flavor=tagtive&plugir
TABASCO - Request Credentials
TABASCO
Pending petitions
Logged in as: oscar : Logout
Home
My Reaction Tags
My Account
My Community
My Grantees
b)
c)

Authorize TABASCO to operate on your behalf

Available sites at Tabasco	Current status	Operations	
Wordpress	granted	Request Authorization	Revoke Authorization
Flikcr	pending	Request Authorization	Revoke Authorization

Request for workmate's authorization

Workmate name	Site	Current status	Operations
jon	Diigo	accepted	Request authorization
jon	Flikcr	pending	Request authorization

Authorization petition management

For my account in...	Petitioner	Current status	Operations		
Wordpress	jon	accepted	Accept	Reject	Revoke
Wordpress	iker	pending	Accept	Reject	Revoke

Figure 7.5: TABASCO tabs: a) granting TABASCO access to your accounts; b) requesting authorization on someone else's account; and c) managing authorization petitions on your accounts

notified through the *"mail"* icon, and handled through the *"My Grantees"* tab. If granted, TABASCO extends the credential to the petitioner so that he can now define reactive tags on this account. Authorization can be revoked at any moment by the account owner. This process is internal to TABASCO, and it does not involve any additional interaction with the website (e.g. Wordpress). At any moment, owners can check the status of their tokens, and revoke authorizations. This disables the affected rules (i.e. affected tags are not longer reactive) but does not delete them. If the authorization is later renewed, these rules are enabled again. Rule deletion should be explicitly conducted by the creator. Rule deletion does not imply the removal of the companion reactive tag from the source site.

In an experiment conducted among 10 PhD students revealed a deviation on the opinion about transferring user credentials to TABASCO. Two of them strongly agreed, three agreed, one was neutral and four disagreed. Although they were informed that the transferred authorizations can be revoked, some of them felt very reluctant to hand out credentials. We hope this suspicion will decrease as OAuth[5] becomes mainstream. Another misunderstanding was about the nature of TABASCO. TABASCO is thought to be deployed as a Web application by the community at hand, and hence, within the control and management of the community. The Web interface made some students think they were delivering their credentials to a third party (i.e. as if TABASCO was an online service similar to Facebook) which it is certainly not the case.

7.5 Reactive Tag Definition

Beside hiding API intricacies, LDWs used by TABASCO turn obscure and heterogeneous API data into meaningful and uniform semantic data. For instance, LDWs for video manager sites (e.g. *Flickr* and *Vimeo*) could return *schema:VideoObject* items, while LDWs for blog managers (e.g. *Wordpress* and *Blogger*) could return *tsioc:BlogPost*[6] items. This simplifies the definition of reactive tags as transformational rules. TABASCO

[5]*OAuth* (Open Authorization) is an open standard for authorization that allows users to share their private resources (e.g. bookmarks) without having to hand out their credentials. This is achieved by handing out tokens. A token grants access to a specific site (e.g. *Diigo*) for specific resources (e.g. bookmarks at *myReadingList* folder) and for a defined duration (e.g. the next 2 months). See `http://oauth.net/`.

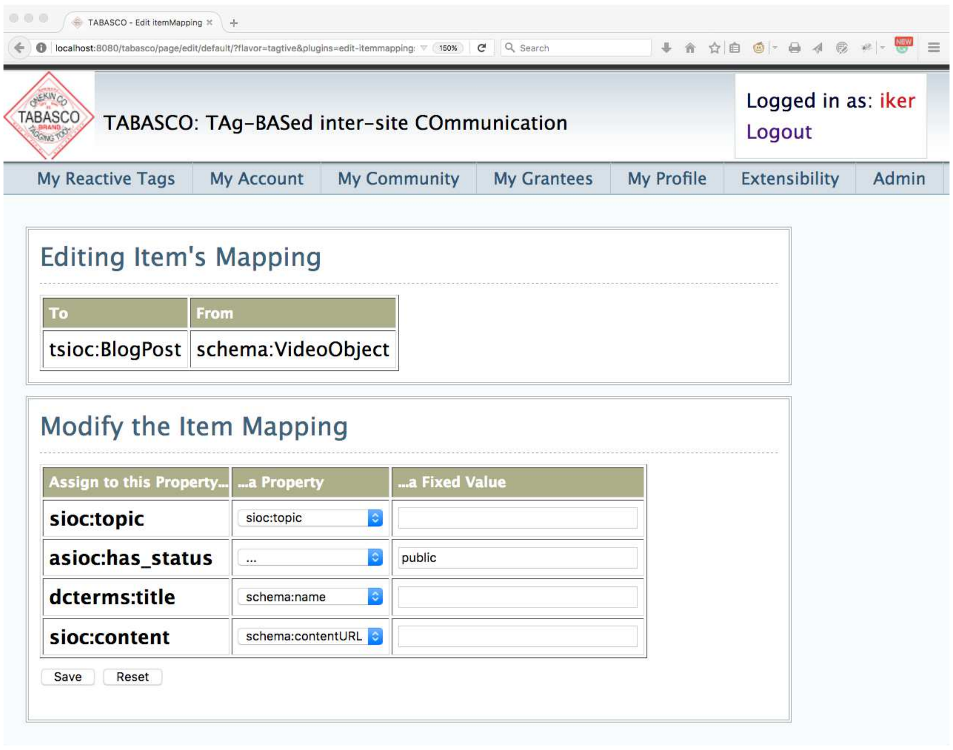
[6]*tsioc* is the prefix for the `http://rdfs.org/sioc/types#` namespace.

administrators create rule templates defining how receiver site items are created from fixed values or from values of the sender site items. In the latter case, instead of setting mappings among API data attributes, administrators set mappings among semantic properties (see Figure 7.6). Semantics of properties help finding related properties and values. In addition, the item uniformity reduces the number of possible combinations. For example, if a set of web site APIs are turned into three types of items (e.g. *BlogPost*, *Bookmark* and *VideoObject*) this implies nine possible transformations: *BlogPost* to *BlogPost*, *BlogPost* to *Bookmark*, *BlogPost* to *VideoObject*, etc[7]. These mappings are already engineered in TABASCO. From this perspective, edges are envisioned as pipes that push items along the *Collaboration Space*.

TABASCO users define reactive tags by instantiating a transformational rule template. In order to do so, users specify (1) the source node, (2) the target node, (3) the label and (4) the operational semantics. In Figure 7.7, the left-hand side panel provides available nodes according to the authorizations held by the current user. Source nodes are restricted to accounts owned by the user. That is, a user cannot define a reactive tag that departs from someone else's account. Target nodes correspond to accounts the user is authorized to operate upon. This includes his own accounts plus those he has been granted authorization. Through drag&drop, the user initializes the middle canvas with the desired nodes. Standing for user accounts, nodes are depicted as a blend of the user picture and the website icon. Edges can now be drawn between user accounts, and in so doing, setting the operational semantics of tags. For instance, users can overwrite property mappings in the rule template.

As an example, let's consider our first scenario, the semantics of the

[7]It could be possible to define a canonical model that factors out the n item types so that the number of combinations would be reduced from $n * n$ to $2 * n$. However, the overlapping among item types is rather small, and hence, the mapping between the type and the canonical model would have been limited to very general properties. By contrast, a direct type-to-type mapping permits to express correspondences beyond general properties.


TABASCO: TAg-BASed inter-site COmmunication

Logged in as: iker
Logout

Editing Item's Mapping

To	From
tsioc:BlogPost	schema:VideoObject

Modify the Item Mapping

Assign to this Property...	...a Property	...a Fixed Value
sioc:topic	sioc:topic	
asioc:has_status	...	public
dcterms:title	schema:name	
sioc:content	schema:contentURL	

Figure 7.6: From a *schema:VideoObject* item to a *tsioc:BlogPost* item

toshare tag (see Figure 7.7): *"on tagging toshare at Oscar's Flickr, do create a post in the* research *category on Oscar's Wordpress"*. The type of both items is set by the participating websites (i.e. *Flickr* handles *schema: VideoObject* items while *Wordpress* manages *tsioc:BlogPost* items). The *sioc:topic* property is bound to the "research" value too.

7.6 Conclusion

TABASCO is a proof of concept to test the viability of SYQL. SYQL supports TABASCO fulfilling its requirements (i.e. interoperability and integrity). In general terms, LD Applications could take benefits not only from the main ideas of this dissertation, i.e. LDW externalization and community curation, but from the LDWs management too. The benefits

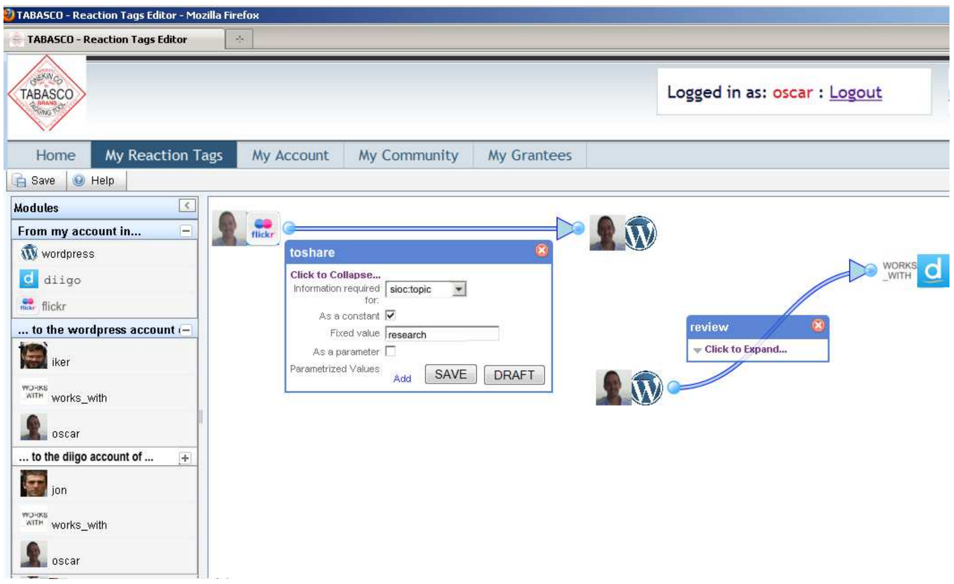


Figure 7.7: Reactive Tag running examples:*"toshare" & "review"*

provided by SYQL are listed below:

- Uniform interface. Instead of calling to heterogeneous APIs, LD Applications benefit from a simple and uniform interaction interface. Namely, to look up a resource dereferencing its URI (GET request), and to create a resource sending an RDF resource to a URI (POST request).

- Reusable LDWs. Programming effort is reduced since LDWs are available on the web to be reused. SYQL publishes LDWs in *DataHub* so application developers can discover useful data sources. To this end, SYQL creates a Hydra description from the LDW definition and publishes quality measurements through the Health Checker. Even if LDWs do not fulfill completely consumers needs (e.g. structurally, because the lack of properties, or functionally, due to quality issues), they can adapt LDWs.

- Current data. Current Linked Data offers new opportunities for innovative applications. Linked Data on the cloud usually is composed of almost static datasets updated once or twice a year. By contrast, LDWs turn API data into Linked Data on demand. In the case of Social Web sites, User Generated Data is provided as Linked Data. Furthermore, the resource insertion capability may produce a more dynamic data flow in the Linked Data cloud.

- Consumer credentials. LD Applications must manage consumer credentials. SYQL allows applications to provide user credentials which take precedence over the LDW producer credentials. In this way, users do not rely on SYQL but on the application.

Chapter 8

Conclusions

"Anybody who is not like everybody, who does not think like everybody, runs the risk of being eliminated."

– *José Ortega y Gasset*

8.1 Overview

LDWs turn APIs data into semantic data. This has the potential of providing API's current data in the Linked Data cloud. However, LDWs usually are embedded in projects which limit LDWs lifespan. LDWs as separated artifacts need a supporting infrastructure and a community for LDWs maintenance. This dissertation addressed these challenges by establishing requirements for LDW Platforms and by developing a solution on top of the YQL system. A proof of concept application was used to assess the applicability of the presented ideas.

This chapter reviews the main results of this dissertation, assesses its limitations, and suggests works for future research.

8.2 Results

This dissertation developed the content of the research into four main chapters, whose contributions are detailed next:

- Chapter 3 identifies requirements for LDW Platforms. It does so through three stakeholders in the LDW lifecycle: Producers, Consumers and Curators. They are interdependent: if an LDW is not produced it cannot be curated, and if an LDW is not updated it is not consumed. Requirements for LDW Platforms from the stakeholders perspective are established: LDW definition&deployment, LDW discovery&lookup, resource lookup and spot&clean stains.

- Chapter 4 describes the Semantic YQL, an LDW Platform on top of the Yahoo YQL system. The platform aims to fulfill the aforementioned requirements with a focus on LDW externalization and LDW curation. The Health Checker highlights two kind of issues: data quality issues and API issues. Re-engineering tools allow to curate LDW modifying lifting mappings, editing credentials and changing example URIs.

- Chapter 5 evaluates the LDW Platform. Experiments have been performed in order to evaluate the platform from the stakeholders point of view. That is to say, effectiveness, latency, scalability, efficiency and satisfaction have been assessed. In addition, SYQL is compared with other platforms.

- Chapter 7 checks the feasibility of SYQL as a data layer. The TABASCO proof of concept is designed and developed to manage (i.e. read-write) data in the Linked Data cloud. SYQL supports TABASCO fulfilling its requirements (i.e. interoperability and integrity). SYQL, as a data layer, offers some advantages: a uniform interaction interface, reusable LDWs, current data and consumer credentials.

A LDW Platform and a set of LDWs has been developed. The source code is openly available on the Onekin Research Group *GitHub*. Concretely, the LDWs are at `https://github.com/onekin/ldw` and the SYQL platform is at `https://github.com/onekin/ldwServer`.

8.3 Publications

Parts of the work explained in this thesis have been already presented and discussed in distinct peer-reviewed forums. The list of publications to which the author has contributed are listed below:

Journals

- Iker Azpeitia, Jon Iturrioz, and Oscar Díaz. Linked Data Wrapper curation: A platform perspective. Semantic Web, pages 1–27. JCR Impact Factor 2016: 2.889 (Q1). **Accepted in the first round. Under review in the second round**.

International Conferences

- Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. YQL as a platform for Linked-Data wrapper development. In International Conference on Web Engineering (ICWE), pages 355–373. Springer, 2015. Acceptance rate 23.6%. Rank B in the CORE2017 [IAD15b].

- Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Cross publishing 2.0: Letting users define their sharing practices on top of YQL. In International Conference on Web Engineering (ICWE), pages 76–92. Springer, 2014. Acceptance rate 20.0%. Rank B in the CORE2017 [IAD14a].

- Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Generalizing the like button: Empowering websites with monitoring capabilities. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC), pages 743–750. ACM, 2014. Acceptance rate 23.22%. Rank B in the CORE2014 [IAD14b].

- Jon Iturrioz, Oscar Díaz, and Iker Azpeitia. Reactive tags: Associating behaviour to prescriptive tags. In Proceedings of the 22nd ACM conference on Hypertext and hypermedia (HT), pages 191–200. ACM, 2011. Acceptance rate 21.0%. Rank A in the CORE2014 [IDA11b].

Workshops/Posters

- Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Linked Data Wrappers atop Yahoo's YQL. In Workshop Services and Applications over Linked APIs and Data (SALAD@ ESWC), pages 20–21, 2015 [IAD15a].

- Jon Iturrioz, Oscar Díaz, and Iker Azpeitia. A Tool for defining the semantics of prescriptive tags. In The 22nd ACM Hypertext Conference, Posters and Demos HT'11. ACM, 2011.[IDA11a]

8.4 Assessment and Future Research

The work presented in this dissertation introduces, motivates and proposes an LDW Platform for sharing and maintaining LDWs. However, an objective assessment exposes some limitations of this work, which motivates areas of extension and future improvement.

Producers perspective

- **Generic execution engines**: As its name denotes, SYQL is an implementation strongly tied to the Yahoo's YQL system. Besides of breaking dependence on Yahoo, allowing new LDW definition languages and execution engines would increase the number of LDWs available.

- **Automatic LDWs generation**: manual LDW generation requires producer effort. It would be interesting to automatically generate LDWs. The low quality of produced LDW would not be an issue since this dissertation advocates for the LDWs curation through SYQL. The challenge therefore is how to automatically generate a functional LDW. One option is to extend existing LDWs to sibling API's methods. For example, if an LDW is created for the *flickr.photos.info* ODT then it might be cloned to be functional for the *flickr.photos.search* and *flickr.photos.recent* ODTs.

Consumers Perspective

- **Interlinkage quality**: LDWs are implicit Datasets where LD Resources are created on-the-fly. This generates dead interlinkages that are not detectable in advance [RSAS14]. Hence, it is detrimental to data quality. For example, creating the *schema:about* interlink to DBpedia based on a tag (e.g. "covfefe") could produce not existing URIs. A solution to enhance interlinks quality could be to check and hide dead interlinks before LD resources are dispatched. However, the latency would increase considerably and it would be unacceptable. Other solution could be to rate in the Health Checker through the "dead interlinks rate" of a property. That is, a posteriori assessment of the dead interlinks over the total interlinks generated. For example, in an LDW the dead interlinks rate for the *schema:about* property could be 0.2 whilst in other one it could be 0.9. It indicates to Consumers the expected interlinkage quality.

- **Speeding up data processing**: LDW-based applications are limited on the quantity of data retrieved (i.e. usually is dereferenced one resource per call) and on the response speed due to the network latency. These limitations constrain the number of LD Applications able to consume LDWs. Application data cache could help [NS17].

- **Traverse applications**: LD applications could traverse the Linked Data cloud from node to node. Traversing nodes offer the advantage of retrieving current data anytime the path is traversed. The question now is how could the path be described. That is, a path starts in a node, goes through a chain of nodes and reaches the last node. The aim is to retrieved data from the last node, but maybe data from intermediary nodes are interesting too [HÖ16]. Challenges include conditional bifurcations in paths depending on data in intermediary nodes.

Curators perspective

- **Refactoring non-working wrappers**: SYQL focuses on curating LDWs deployed in SYQL. However, this dissertation mentions wrappers that are not working anymore. Could those wrappers be refactored as SYQL LDWs? In doing so, a user guide could be defined to help developers and curators at wrappers recovery.

- **LD cloud Health Checker**: the Health Checker for LDWs on SYQL is a contribution of this dissertation. It is based on the URI examples dereferenciation, so, it seems easy to extend the check service to LD cloud nodes. As an example, it could be interesting to define a *Health Radar* checking the LD cloud nodes in order to create a *Cloud Health Map*. Every day a monitoring application could check selected URIs in DBpedia, Bio2RDF, BBC Music, Eurostat, etc. The challenge here is to determine the health dimensions and granularity. For instance, the API dimension (i.e. API key and API response) is not applicable to DBpedia but value consistency could be a must.

- **Community management**: SYQL is based on a community sharing, consuming and curating LDWs. Hence, ways to promote participation is a future research area. In addition, crowdsourced curation could provoke discrepancies among curators. Deciding which property to remove could be conflictive because each stakeholder has his own interests. We assessed the convenience of enforcing *Backward Compatibility*, that is, new properties can be added and old properties are not removed. However, this would lead to a decreasing data quality since deprecated (Zaveri's CS4 metric) or null-valued (SV2) properties could not be removed and the incremental addition of redundant properties would undermine conciseness.

8.5 Conclusion

This dissertation addresses wrapper curation challenges: LDWs' lifecycles are coupled to those of the breakout projects, LDW maintenance penalty is high, and there is a shortage of people involved in LDWs maintenance. The SYQL platform has been developed to address these challenges. Its feasibility as data provider has been proved for LD Applications such as plugins, Task Automation Services, RDF visualizers and mashup platforms. It has been evaluated from the stakeholders perspective: Consumers, Producers and Curators. The results are promising. However, the presented approach has still to demonstrate that really pays off for real users. It would be demonstrated if LDWs are created by external users, and if the SYQL's source code is improved and installed by third parties. This will certainly imply moving from prototypes to products and from testing students to real stakeholders as the target audience. Developments produced along this dissertation are available on GitHub in order to foster it. In addition, this dissertation has open new research lines worth to be explored.

Appendix A

SYQL Implementation

"Meditation on any theme, if positive and honest, inevitably separates him who does the meditating from the opinion prevailing around him, from that which . . . can be called "public" or "popular" opinion."

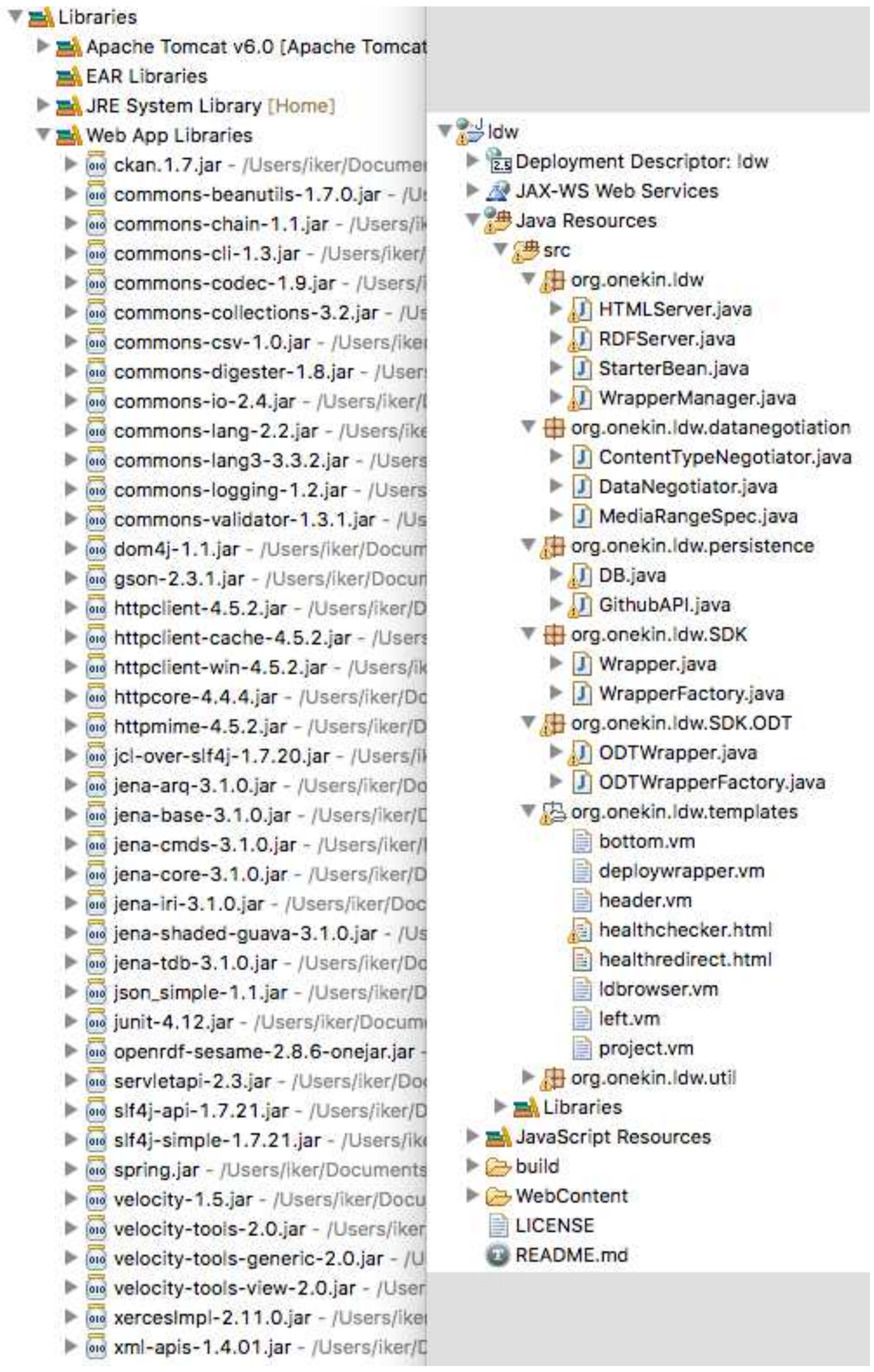
– José Ortega y Gasset

A.1 Java Project

The SYQL Java project has been developed using the Eclipse Java EE IDE for Web Developers (Version: Kepler Service Release 2). Figure A.1 shows the packages, classes and required external libraries. The complete source code and required libraries are publicly available at `https://github.com/onekin/ldwServer` so that other researchers can test, improve and run SYQL instances.

A.2 SYQL Storage

SYQL stores information in different places:



```
▼ Libraries
  ▶ Apache Tomcat v6.0 [Apache Tomcat
    EAR Libraries
  ▶ JRE System Library [Home]
  ▼ Web App Libraries
    ▶ ckan.1.7.jar - /Users/iker/Docume
    ▶ commons-beanutils-1.7.0.jar - /U
    ▶ commons-chain-1.1.jar - /Users/i
    ▶ commons-cli-1.3.jar - /Users/iker/
    ▶ commons-codec-1.9.jar - /Users/i
    ▶ commons-collections-3.2.jar - /U
    ▶ commons-csv-1.0.jar - /Users/ike
    ▶ commons-digester-1.8.jar - /User
    ▶ commons-io-2.4.jar - /Users/iker/
    ▶ commons-lang-2.2.jar - /Users/ike
    ▶ commons-lang3-3.3.2.jar - /Users
    ▶ commons-logging-1.2.jar - /Users
    ▶ commons-validator-1.3.1.jar - /Us
    ▶ dom4j-1.1.jar - /Users/iker/Docum
    ▶ gson-2.3.1.jar - /Users/iker/Docum
    ▶ httpclient-4.5.2.jar - /Users/iker/D
    ▶ httpclient-cache-4.5.2.jar - /Users
    ▶ httpclient-win-4.5.2.jar - /Users/ik
    ▶ httpcore-4.4.4.jar - /Users/iker/Do
    ▶ httpmime-4.5.2.jar - /Users/iker/D
    ▶ jcl-over-slf4j-1.7.20.jar - /Users/i
    ▶ jena-arq-3.1.0.jar - /Users/iker/Do
    ▶ jena-base-3.1.0.jar - /Users/iker/D
    ▶ jena-cmds-3.1.0.jar - /Users/iker/
    ▶ jena-core-3.1.0.jar - /Users/iker/D
    ▶ jena-iri-3.1.0.jar - /Users/iker/Doc
    ▶ jena-shaded-guava-3.1.0.jar - /Us
    ▶ jena-tdb-3.1.0.jar - /Users/iker/Do
    ▶ json_simple-1.1.jar - /Users/iker/D
    ▶ junit-4.12.jar - /Users/iker/Docum
    ▶ openrdf-sesame-2.8.6-onejar.jar -
    ▶ servletapi-2.3.jar - /Users/iker/Do
    ▶ slf4j-api-1.7.21.jar - /Users/iker/D
    ▶ slf4j-simple-1.7.21.jar - /Users/ik
    ▶ spring.jar - /Users/iker/Documents
    ▶ velocity-1.5.jar - /Users/iker/Docu
    ▶ velocity-tools-2.0.jar - /Users/iker
    ▶ velocity-tools-generic-2.0.jar - /U
    ▶ velocity-tools-view-2.0.jar - /User
    ▶ xercesImpl-2.11.0.jar - /Users/ike
    ▶ xml-apis-1.4.01.jar - /Users/iker/D
```

```
▼ ldw
  ▶ Deployment Descriptor: ldw
  ▶ JAX-WS Web Services
  ▼ Java Resources
    ▼ src
      ▼ org.onekin.ldw
        ▶ HTMLServer.java
        ▶ RDFServer.java
        ▶ StarterBean.java
        ▶ WrapperManager.java
      ▼ org.onekin.ldw.datanegotiation
        ▶ ContentTypeNegotiator.java
        ▶ DataNegotiator.java
        ▶ MediaRangeSpec.java
      ▼ org.onekin.ldw.persistence
        ▶ DB.java
        ▶ GithubAPI.java
      ▼ org.onekin.ldw.SDK
        ▶ Wrapper.java
        ▶ WrapperFactory.java
      ▼ org.onekin.ldw.SDK.ODT
        ▶ ODTWrapper.java
        ▶ ODTWrapperFactory.java
      ▼ org.onekin.ldw.templates
          bottom.vm
          deploywrapper.vm
          header.vm
          healthchecker.html
          healthredirect.html
          ldbrowser.vm
          left.vm
          project.vm
      ▶ org.onekin.ldw.util
    ▶ Libraries
  ▶ JavaScript Resources
  ▶ build
  ▶ WebContent
    LICENSE
    README.md
```

Figure A.1: Required libraries at the lefthand and packages/classes at the righthand

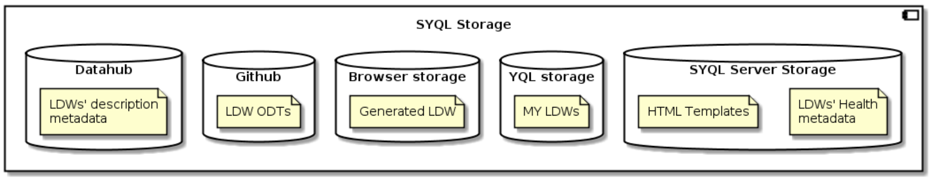


Figure A.2: SYQL storage component diagram

- Provisional LDWs (those in edition process) are initially stored in the browser and then in the YQL storage.
- LDWs are finally stored in GitHub.
- LDWs are described in DataHub for discovery purposes.
- LDW's quality assessments are stored in the SYQL server.
- Velocity[1] templates are used to generate HTML web pages. These templates are stored in the server.

Figure A.2 depicts databases in the SYQL storage component.

A.3 SYQL Front-end

Stakeholders interact with the SYQL front-end (see Figure A.3) to be aware of LDW's health and to curate LDWs. The former is performed through the *Health Checker Window* which lists all deployed LDWs. The SYQL server dispatches LDW's health information (i.e. CheckHealth interface). The latter is achieved by the re-engineering windows: Lifting mapping, Lowering mapping and Credentials editor. *LDWs Editor* complements the Lifting mapping. Both are implemented as a Greasemonkey augmentation over the YQL console and editor respectively. The augmentation script is available at `https://github.com/onekin/ldw/blob/master/odt2ldw.user.js`. Re-engineering windows read LDWs

[1]http://velocity.apache.org/

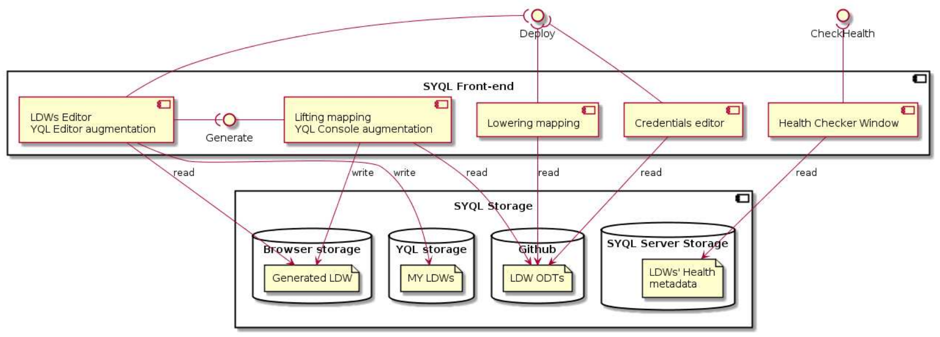


Figure A.3: SYQL front-end components diagram (simplified SYQL storage component)

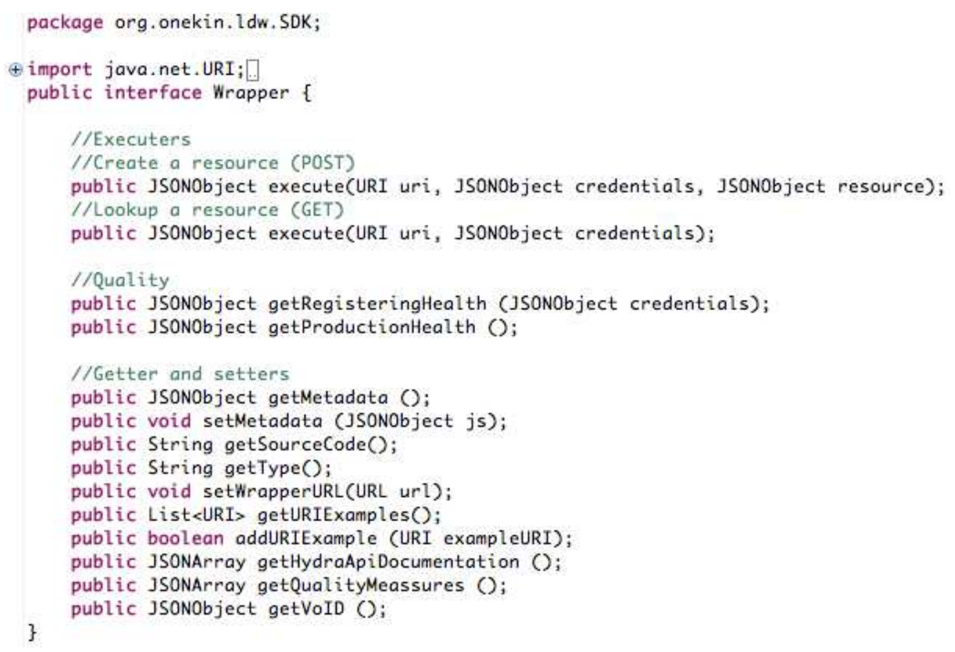
from GitHub and show annotations or credentials to be edited. Changes are added to LDWs and the SYQL server deploys them.

It is worth to note that this YQL-based front-end is an implementation to take advantage of the YQL system. SYQL is designed to be extended with other systems. To support a new type of wrappers the Wrapper interface (see Figure A.4) and the WrapperFactory interface (see Figure A.5) have to be implemented. For example, a new type of wrappers could interpret PHP and execute them through an online PHP engine. It would result in the Semantic PHP-platform (SPHP-platform)! So far it is not tested. Any front-end requires to access the *CheckHealth* and the *Deploy* interfaces provided by the SYQL back-end.

A.4 SYQL Back-end

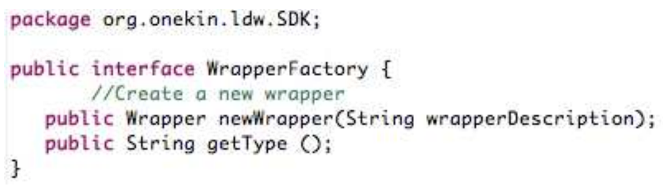
The SYQL back-end is a part of the SYQL server. That means that interfaces are implemented as a web API to interact with. Three interfaces are provided: the *Deploy* interface, the *CheckHealth* interface and the *Execute* interface (see Figure A.6).

The *CheckHealth* and *Deploy* interfaces are implemented by the HTML server. The *Wrapper Manager* (re)deploys LDWs. It retrieves the LDW definition from the *YQL storage*, checks it, and updates the *Github storage*



```
package org.onekin.ldw.SDK;

import java.net.URI;
public interface Wrapper {

    //Executers
    //Create a resource (POST)
    public JSONObject execute(URI uri, JSONObject credentials, JSONObject resource);
    //Lookup a resource (GET)
    public JSONObject execute(URI uri, JSONObject credentials);

    //Quality
    public JSONObject getRegisteringHealth (JSONObject credentials);
    public JSONObject getProductionHealth ();

    //Getter and setters
    public JSONObject getMetadata ();
    public void setMetadata (JSONObject js);
    public String getSourceCode();
    public String getType();
    public void setWrapperURL(URL url);
    public List<URI> getURIExamples();
    public boolean addURIExample (URI exampleURI);
    public JSONArray getHydraApiDocumentation ();
    public JSONArray getQualityMeassures ();
    public JSONObject getVoID ();
}
```

Figure A.4: Wrapper interface



```
package org.onekin.ldw.SDK;

public interface WrapperFactory {
        //Create a new wrapper
   public Wrapper newWrapper(String wrapperDescription);
   public String getType ();
}
```

Figure A.5: WrapperFactory interface

and the *Datahub storage*.

The *CheckHealth* interface is managed by the *Wrapper Manager* too. When the *Health Checker Window* is requested via the *HTML Server* the manager asks wrappers for their health. Finally, the *HTML Server* dispatch the health information using an HTML template.

The RDF Server dispatches RDF data, that is, LDW lookup, resource lookup and resource creation requests (i.e. *Execute interface)*. Wrappers send *YQL Statements* to the YQL engine in order to dereferenciate or create resources. The YQL Engine component interprets LDWs as ODTs and calls underlying APIs.

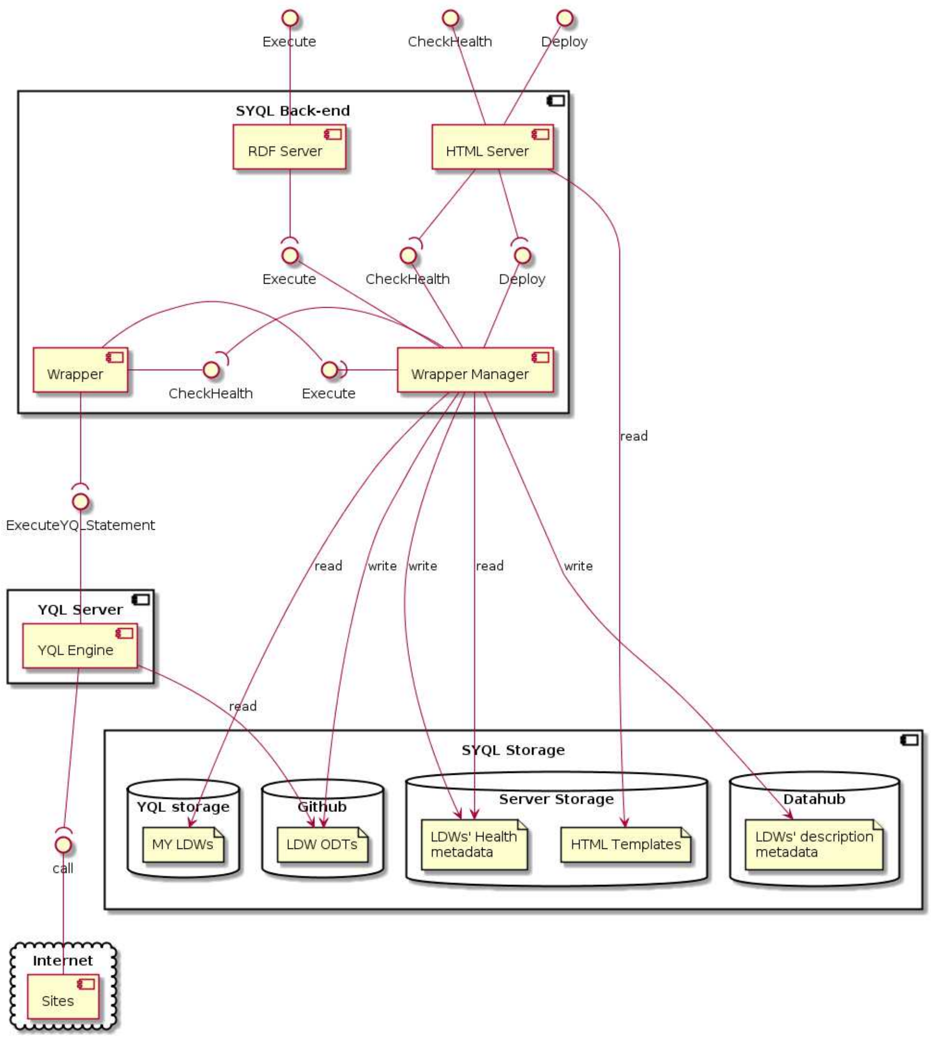


Figure A.6: SYQL back-end components diagram (simplified SYQL storage component)

Appendix B

Demonstration

"Man is a substantial emigrant on a pilgrimage of being, and it is accordingly meaningless to set limits to what he is capable of being."
– José Ortega y Gasset

B.1 Overview

This appendix guides readers through a demonstration in order to check the SYQL system in creating, consuming and curating LDWs.

B.2 Installation

Following steps set up dependencies of SYQL:

1. Open an account in Yahoo. Alternatively, you can use the sample account: User: *ana.fiss@yahoo.es* Password: *ldw-onekin*.

2. Install the Greasemonkey Firefox add-on https://addons.mozilla.org/en-US/firefox/addon/greasemonkey/.

3. Install the SYQL plug-in available at https://raw.githubusercontent.com/onekin/ldw/blob/master/odt2ldw.user.js. SYQL client-side is supported as a Firefox 52.0 plug-in on top of the YQL Console/Editor.

B.3 LDW Definition

Once logged in Yahoo, go to the YQL Editor at `https://developer.yahoo.com/yql/editor/` and click on the "LDW template (select)" link. In doing so, an incomplete wrapper appears in the editor. Complete the URI pattern and example, the credentials if required, the lowering part, and the lifting part. For example, you can copy the LDW in Figure 4.5. Alternatively, to speed up the LDW edition you can copy the code for the running example from `https://github.com/onekin/ldw/blob/master/flickr.videodemo.ldw`. Finally, click on the *Save* button and give a name to your LDW.

B.4 LDW Deployment

After clicking the *Deploy* button the Verification window appears displaying the result of the verification process. If verification issues are arisen, they must be solved on the editor. Syntactic issues indicate the lack of essential parts (e.g. the URI example is lost). Dereferencing issues appear if there are errors on either the API call or the lifting process. To check the XML parser, you could remove the URIExample from the current LDW and next click the "Deploy" button. The registration window should alert about the lack of the URIExample. Similarly, a dereference error could be produced by introducing an incorrect JavaScript code into the lifting function. For example, write the "5 = 6" incorrect assignment at the beginning of the lifting function. This will result in the creation of a void LD resource. Once corrected, acknowledgement messages should pop up in the

Verification Console.

B.5 LDW Discovery

Go to the `https://datahub.io/dataset/flickr_videodemo` URL for consulting information about the recently deployed demo LDW. Explore metadata and click on links to retrieve LDW's information.

B.6 LDW Lookup

Let's inspect the *flickr.videodemo* wrapper's different descriptions. The VoID description is retrieved dereferencing the Wrapper's URI (i.e. `http://rdf.onekin.org/flickr/videodemo/(photo_id)`). It will show information similar to that in Figure 4.9 extended with all the measurements' URIs (e.g. P2, IN3, T7, CM4, ...). Dereferencing one of them assessment value will appear. For instance, dereferencing http://rdf.onekin .org/flickr/videodemo/(photo_id)/dqv/P2measure could show:

> {"@type":"dqv:QualityMeasurement", "@id":"http://rdf.onekin.org /flickr/videodemo/(photo_id)/dqv/P2measure", "dqv:value":"831", "dqv:computedOn":"http://rdf.onekin.org/flickr/videodemo /(photo_id)"}

On the other hand, the Hydra description in the `http://rdf.onekin.org/flickr/videodemo/(photo_id)/apidocumentation` URI depicts the supported class description, the supported operation, the entry point and credentials. Credentials indicate what data is required for resource lookup:

{"@id":"http://rdf.onekin.org/flickr/videodemo/(photo_id)/ apidocumentation/credentials", "@type":"hydra:Class", "hydra:description": "Required credentials", "hydra:title":"Credentials", "hydra:supportedProperty": [{"@type": "hydra:SupportedProperty", "hydra:required":"true", "hydra:title": "api_key"}

B.7 Resource Lookup

Consumers do not need to install anything in order to dereference resources in SYQL. Resources are directly dereferenceable in the browser, or in any JSON viewer. For example, go to the *Health Checker* `http://rdf.onekin.org/ldw/page/healthchecker/`, unfold the *flickr .videodemo* LDW, and click on the URI Example for this resource to be displayed using *Online JSON Viewer*[1]

This example illustrates resource lookup using the producer's API key. Alternatively, the API key can be programmatically provided by the *LDW consumer*. This requires the HTTP request to hold a lookup URI and an *Authorization* header providing the API key value (i.e. *schema:value*) along the Hydra credential description:

> *GET* *`http://rdf.onekin.org/flickr/videodemo/27376196615`*
>
> *Authorization* = {"hydra:supportedProperty": [{"*hydra:title*": "api_key", "schema:value": "2c894ba749b413 7b6f7ab127c86890ec"}]}

For security reasons, the *Authorization header* should be encrypted but for the sake of a better understanding it is not encrypted. SYQL recovers the *api_key* from the header and embeds it in the API call. This API key takes precedence over the one embedded in the LDW. Readers can check this out through Hurl[2].

B.8 Spotting Stains

Go to the Health Checker console available at `http://rdf.onekin.org/ldw/page/healthchecker/`. Spot one LDW worth curating. For instance, *flickr.videodemo* has some data-quality issues. Some of them

[1] `http://jsonviewer.stack.hu`

[2] `https://www.hurl.it/`

are not directly solvable because they depend on the net performance (e.g. latency). Other issues can be solved editing the LDW, for example, a broken interlink. Let's solve this out. Click on the *Maintain the LDW* button and the YQL console will open. This moves us to next demo section.

B.9 Cleaning up Stains

Once in the YQL console (`https://developer.yahoo.com/yql/console/`) select the *Community LDWs* radio button. Click on the *flickr .videodemo* LDW and the *Annotation View* tab will appear. Here, annotation interlays will show up after re-engineering *flickr.videodemo* code. Go to the annotation that accounts for the broken interlink: property with path *photo/location/locality /content*. Update the *Association mapping's* URI to *http://rdf.onekin.org/geo/place/{VALUE}*. A preview of the impact on the instance resource can be obtained by moving to the *Semantic View* tab. Once satisfied with the output, click the *Generate* button to obtain the code counterpart. This moves you to the YQL Editor Console where you can click on the *Redeploy* button. Finally, go to the Health Checker console `http://rdf.onekin.org/ldw/page/healthchecker/` to check the interlink works.

Bibliography

[ACHZ11] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets with the VoID vocabulary. *Working draft, W3C*, March 2011.

[AI16] Riccardo Albertoni and Antoine Isaac. Data on the Web Best Practices: Data Quality Vocabulary. *Working draft, W3C*, December 2016.

[AMB$^+$] Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch, and Richard Cyganiak. Linking Open Data cloud diagram 2017. http://lod-cloud.net/ [accessed June 2017].

[AZS$^+$16] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Fabian Flöck, and Jens Lehmann. Detecting Linked Data quality issues via crowdsourcing: A DBpedia study. *Semantic Web*, (Preprint):1–33, 2016.

[BB] Christian Bizer and Christian Becker. Flickr wrappr: precise photo association. http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/ [accessed June 2017].

[BBFD08] U. Bojars, J. G. Breslin, A. Finn, and S. Decker. Using the Semantic Web for linking and reusing data across Web 2.0 communities. *Web Semantics*, 6(1):21–28, 2008.

[BCG07] Christian Bizer, Richard Cyganiak, and Tobias Gauß. The RDF Book Mashup: From web APIs to a Web of Data. In

Proceedings of the ESWC'07 Workshop on Scripting for the Semantic Web (SFSW), volume 248. CEUR Workshop Proceedings, 2007.

[BK11] Florian Bauer and Martin Kaltenböck. Linked Open Data: The essentials. *Edition mono/monochrom, Vienna*, 2011.

[BL06] Tim Berners-Lee. Linked Data. *Design issues, W3C*, 2006. https://www.w3.org/DesignIssues/LinkedData.html [accessed July 2017].

[BPM$^+$08] Diego Berrueta, Jon Phipps, Alistair Miles, Thomas Baker, and Ralph Swick. Best practice recipes for publishing RDF vocabularies. *Working draft, W3C*, August 2008.

[BRB$^+$14] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: a uniform way of publishing other people's dirty data. In *International Semantic Web Conference (ISWC)*, pages 213–228. Springer, 2014.

[CB] Richard Cyganiak and Christian Bizer. D2RQ. Accessing Relational Databases as Virtual RDF Graphs. http://d2rq.org/ [accessed June 2017].

[CCTAD13] Alison Callahan, José Cruz-Toledo, Peter Ansell, and Michel Dumontier. Bio2RDF release 2: Improved coverage, interoperability and provenance of life science Linked Data. In *Extended Semantic Web Conference (ESWC)*, pages 200–212. Springer, 2013.

[Dat] Datanyze. Social sharing market share table. https://www.datanyze.com/market-share/social-sharing/ [accessed June 2017].

[DBp16] DBpedia. DBpedia mappings wiki, 2016. http://mappings.dbpedia.org [accessed June 2017].

[Dii] Diigo. Homepage. https://www.diigo.com/ [accessed June 2017].

[DLA16] Jeremy Debattista, Christoph Lange, and Sören Auer. Luzzu – A framework for Linked Data quality assessment. In *International Conference on Semantic Computing (ICSC)*, pages 124–131. IEEE, 2016.

[DLE^{+}11] Li Ding, Timothy Lebo, John S Erickson, Dominic DiFranzo, Gregory Todd Williams, Xian Li, James Michaelis, Alvaro Graves, Jin Guang Zheng, Zhenning Shangguan, Johanna Flores, Deborah L. McGuinness, and Jim Hendler. TWC LOGD: A portal for linked open government data ecosystems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(3):325–333, 2011.

[dMS^{+}08] Mathieu d'Aquin, Enrico Motta, Marta Sabou, Sofia Angeletou, Laurian Gridinoc, Vanessa Lopez, and Davide Guidi. Toward a new generation of semantic web applications. *Intelligent Systems*, 23(3):20–28, 2008.

[DP17] Aba-Sah Dadzie and Emmanuel Pietriga. Visualisation of Linked Data – Reprise. *Semantic Web*, 8(1):1–21, 2017.

[DR11] Aba-Sah Dadzie and Matthew Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2):89–124, 2011.

[DSC12] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. *Recommendation, W3C*, September 2012.

[DV17] Milan Dojchinovski and Tomas Vitvar. Linked Web APIs Dataset: Web APIs meet Linked Data. *Semantic Web*, (Preprint):1–10, 2017.

[Eil11] Eldad Eilam. *Reversing: Secrets of reverse engineering*. John Wiley & Sons, 2011.

[EM10] Orri Erling and Ivan Mikhailov. Virtuoso: RDF support in a native RDBMS. In *Semantic Web Information Management: A Model-Based Perspective*, pages 501–519. Springer, 2010.

[Eri13] Bayo Erinle. *Performance Testing with JMeter 2.9*. Packt Publishing Ltd, 2013.

[Fac] Facebook. Like Button for the Web. https://developers.facebook.com/docs/plugins/like-button [accessed June 2017].

[Fli] Flickr. API documentation. https://www.flickr.com/services/api/ [accessed June 2017].

[FMH17] Michael Färber, Carsten Menne, and Andreas Harth. A Linked Data wrapper for CrunchBase. *Semantic Web*, (Preprint):1–11, 2017.

[fS11] International Organization for Standardization. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.

[GBM16] Ramanathan V Guha, Dan Brickley, and Steve Macbeth. Schema.org: Evolution of structured data on the web. *Communications of the ACM*, 59(2):44–51, 2016.

[Gdb12] Christophe Guéret and Victor de boer. Openaid IATI parser and API, 2012. http://api2lod.appspot.com/oipa [accessed June 2017].

[GGL+14] Alasdair JG Gray, Paul Groth, Antonis Loizou, Sune Askjaer, Christian Brenninkmeijer, Kees Burger, Christine Chichester, Chris T Evelo, Carole Goble, Lee Harland, et al. Applying Linked Data approaches to pharmacology: Architectural decisions and implementation. *Semantic Web*, 5(2):101–113, 2014.

[GS10] Lars Grammel and Margaret-Anne Storey. A survey of mashup development environments. In *The Smart Internet*, pages 137–151. Springer, 2010.

[Gué11] Christophe Guéret. GoogleArt – Semantic data wrapper (technical update). *DATAVERSITY.net*, March 2011. http://www.dataversity.net/googleart-semantic-data-wrapper-technical-update/ [accessed May 2017].

[HB11] Tom Heath and Christian Bizer. Linked Data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.

[HHJ16] Aidan Hogan, Pascal Hitzler, and Krzysztof Janowicz. Linked Dataset description papers at the Semantic Web Journal: A critical assessment. *Semantic Web*, 7(2):105–116, 2016.

[HÖ16] Olaf Hartig and M Tamer Özsu. Walking without a map: Ranking-based traversal for querying linked data. In *International Semantic Web Conference*, pages 305–324. Springer, 2016.

[HUH+12] Aidan Hogan, Jürgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of Linked Data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14–44, 2012.

[IAD14a] Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Cross publishing 2.0: Letting users define their sharing practices on top of YQL. In *International Conference on Web Engineering (ICWE)*, pages 76–92. Springer, 2014.

[IAD14b] Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Generalizing the like button: Empowering websites with monitoring capabilities. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)*, pages 743–750. ACM, 2014.

[IAD15a] Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Linked Data Wrappers atop Yahoo's YQL. In *Workshop in Services and Applications over Linked APIs and Data (SALAD@ ESWC)*, pages 20–21, 2015.

[IAD15b] Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. YQL as a platform for Linked-Data Wrapper development. In *International Conference on Web Engineering (ICWE)*, pages 355–373. Springer, 2015.

[IDA11a] Jon Iturrioz, Oscar Díaz, and Iker Azpeitia. A Tool for defining the semantics of prescriptive tags. In *The 22nd ACM Hypertext Conference, Posters and Demos. HT'11*. ACM, 2011.

[IDA11b] Jon Iturrioz, Oscar Díaz, and Iker Azpeitia. Reactive tags: Associating behaviour to prescriptive tags. In *Proceedings of the 22nd ACM conference on Hypertext and hypermedia (HT)*, pages 191–200. ACM, 2011.

[IFT] IFTTT. The IFTTT website. https://ifttt.com/ [accessed June 2017].

[Ini13] Dublin Core Metadata Initiative. FAQ/DC and DCTERMS Namespaces, 2013.

http://wiki.dublincore.org/index.php/FAQ/DC_and_DCTERMS_Namespaces [accessed June 2017].

[IS15] Bala Iyer and Mohan Subramaniam. The strategic value of APIs. *Harvard Business Review*, January 2015. https://hbr.org/2015/01/the-strategic-value-of-apis [accessed June 2017].

[JP14] Paul Johannesson and Erik Perjons. *An introduction to Design Science*. Springer, 2014.

[KAU$^+$13] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing Linked Data dynamics. In *Extended Semantic Web Conference (ESWC)*, pages 213–227. 2013.

[KC93] Jurek Kirakowski and Mary Corbett. SUMI: The software usability measurement inventory. *British Journal of Educational Technology*, 24(3):210–212, 1993.

[KDN08] John Krumm, Nigel Davies, and Chandra Narayanaswami. User-Generated Content. *IEEE Pervasive Computing*, 7(4):10–11, 2008.

[KHS12] Magnus Knuth, Johannes Hercher, and Harald Sack. Collaboratively patching Linked Data. *arXiv preprint arXiv:1204.2715*, 2012.

[KK15] Aikaterini K Kalou and Dimitrios A Koutsomitropoulos. Towards semantic mashups: Tools, methodologies, and state of the art. *International Journal of Information Retrieval Research*, 5(2):1–25, 2015.

[Kyt05] Thomas Kyte. Invoker and definer rights. In *Expert Oracle*, pages 981–1026. A-Press, 2005.

[KZAL13] Dimitris Kontokostas, Amrapali Zaveri, Sören Auer, and Jens Lehmann. TripleCheckMate: A tool for crowdsourcing the quality assessment of Linked Data. In *International Conference on Knowledge Engineering and the Semantic Web*, pages 265–272. Springer, 2013.

[LBC17] Bernadette Farias Lóscio, Caroline Burle, and Newton Calegari. Data on the web best practices. *Recommendation, W3C*, January 2017.

[LG13] Markus Lanthaler and Christian Gütl. Hydra: A vocabulary for hypermedia-driven web APIs. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web (LDOW)*, volume 996. CEUR Workshop Proceedings, 2013.

[LGdSN10] Stefania Leone, Michael Grossniklaus, Alexandre de Spindler, and Moira C Norrie. Synchronising personal data with Web 2.0 data sources. In *Web Information Systems Engineering – WISE 2010*, pages 411–418. Springer, Springer Berlin Heidelberg, 2010.

[LIJ$^+$15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. DBpedia – a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[lin] Linked Widgets Platform - Smart data exploration and data integration with Linked Widgets. http://linkedwidgets.org/ [accessed June 2017].

[LLSL16] David Lizcano, Genoveva López, Javier Soriano, and Jaime Lloret. Implementation of end-user development success factors in mashup development environments. *Computer Standards & Interfaces*, 47:1–18, 2016.

[lod] SZTAKI LODmilla. http://lodmilla.sztaki.hu/ [accessed June 2017].

[Mag] PC Mag. Definition of: wrapper. https://www.pcmag.com/encyclopedia/term/54886/wrapper [accessed June 2017].

[MF06] David R Millen and Jonathan Feinberg. Using social tagging to improve social navigation. In *Workshop on the Social Navigation and Community Based Adaptation Technologies (SNC-BAT)*, 2006.

[MGCEG13] Nandana Mihindukulasooriya, Raul Garcia-Castro, and Miguel Esteban-Gutierrez. Linked Data Platform as a novel approach for enterprise application integration. In *Proceedings of the Fourth International Conference on Consuming Linked Data (COLD)*, volume 1034, pages 146–157. CEUR Workshop Proceedings, 2013.

[MPD10] Maria Maleshkova, Carlos Pedrinaci, and John Domingue. Semantic annotation of web APIs with SWEET. In *Proceedings of the ESWC'10 Workshop on Scripting for the Semantic Web (SFSW)*, volume 699. CEUR Workshop Proceedings, 2010.

[Neta] Yahoo! Developer Network. Yahoo! GeoPlanet guide. https://developer.yahoo.com/geo/geoplanet/guide/ [accessed June 2017].

[Netb] Yahoo! Developer Network. YQL console. http://developer.yahoo.com/yql/console/ [accessed June 2017].

[NKH$^+$16] Ciro Baron Neto, Dimitris Kontokostas, Sebastian Hellmann, Kay Müller, and Martin Brümmer. Assessing quantity and quality of links between Linked Data datasets. In *Proceedings of the WWW2016 Workshop on Linked Data on the Web (LDOW)*, volume 1593. CEUR Workshop Proceedings, 2016.

[NKMF10] Barry Norton, Reto Krummenacher, Adrian Marte, and Dieter Fensel. Dynamic Linked Data via Linked Open Services. In *Proceedings of the Workshop on Linked Data in the Future Internet at the Future Internet Assembly (LDFI)*, volume 700. CEUR Workshop Proceedings, 2010.

[NS17] Chifumi Nishioka and Ansgar Scherp. Keeping linked open data caches up-to-date by predicting the life-time of RDF triples. In *Proceedings of the International Conference on Web Intelligence*, pages 73–80. ACM, 2017.

[NVCM13] Luís Eufrasio T Neto, Vânia Maria P Vidal, Marco A Casanova, and José Maria Monteiro. R2RML by assertion: A semi-automatic tool for generating customised R2RML mappings. In *Extended Semantic Web Conference (ESWC)*, pages 248–252. Springer, 2013.

[ODKM15] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, and Michele Marchesi. Measuring and understanding the effectiveness of JIRA developers communities. In *Proceedings of the Sixth International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pages 3–10. IEEE Press, 2015.

[ODT] YQL Open Data Tables. GitHub repository. https://github.com/yql/yql-tables [accessed June 2017].

[Ova14] Steven Ovadia. Automate the Internet with "If This Then That" (IFTTT). *Behavioral & Social Sciences Librarian*, 33(4):208–211, 2014.

[PDRM11] Kevin R Page, David C De Roure, and Kirk Martinez. REST and Linked Data: a match made for domain driven development? In *Proceedings of the Second International Workshop on RESTful Design*, pages 22–25. ACM, 2011.

[Pru] Eric Gordon Prud'hommeaux. W3C's RDF validation service. *W3C*. https://www.w3.org/RDF/Validator/ [accessed June 2017].

[PS09] Sotirios Paroutis and Alya Al Saleh. Determinants of knowledge sharing using Web 2.0 technologies. *Journal of Knowledge Management*, 13(4):52–63, 2009.

[Red] Redlink. Vafu, another Linked Data validator. http://vafu.redlink.io/ [accessed June 2017].

[Rod08] Alex Rodriguez. RESTful web services: The basics. *IBM developerWorks*, 2008.

[RSAS14] Enayat Rajabi, Salvador Sanchez-Alonso, and Miguel-Angel Sicilia. Analyzing broken links on the web of data: An experiment with dbpedia. *Journal of the Association for Information Science and Technology*, 65(8):1721–1727, 2014.

[SAD$^+$14] Elena Simperl, Maribel Acosta, Marin Dimitrov, John Domingue, Peter Haase, Maria Maleshkova, Alexander Mikroyannidis, Barry Norton, and Maria Esther Vidal. EUCLID: EdUcational Curriculum for the usage of LInked Data. Chapter 5: building Linked Data applications, 2014. http://euclid-project.eu/modules/chapter5.html [accessed June 2017].

[SAM15] Steve Speicher, John Arwe, and Ashok Malhotra. Linked Data Platform 1.0. *Recommendation, W3C*, February 2015.

[SBP14] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. Adoption of the Linked Data best practices in different topical domains. In *International Semantic Web Conference (ISWC)*, pages 245–260. Springer, 2014.

[SCAV08] Leo Sauermann, Richard Cyganiak, Danny Ayers, and Max Völkel. Cool URIs for the Semantic Web. *Working draft, W3C*, December 2008.

[SGL07] Amit P Sheth, Karthik Gomadam, and Jonathan Lathem. SA-REST: Semantically interoperable and easier-to-use services and mashups. *IEEE Internet Computing*, 11(6):91–94, 2007.

[SH10] Sebastian Speiser and Andreas Harth. Taking the LIDS off data silos. In *Proceedings of the 6th International Conference on Semantic Systems (SEMANTiCS)*, pages 44:1–44:4. ACM, 2010.

[SH11] Sebastian Speiser and Andreas Harth. Integrating Linked Data and services with Linked Data Services. In *The Semantic Web: Research and Applications*, pages 170–184. Springer, 2011.

[SHH+11] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization techniques for federated query processing on Linked Data. In *International Semantic Web Conference (ISWC)*, pages 601–616. Springer, 2011.

[Sim11] Julien Simon. What are Rich Snippets and when to use them?, September 2011.

http://www.6smarketing.com/blog/what-are-rich-snippets-and-when-to-use-them/ [accessed June 2017].

[SLHA12] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. LinkedGeoData: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.

[SLK$^+$17] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. JSON-LD 1.1: A JSON-based serialization for Linked Data. *Working draft, W3C*, June 2017.

[TKSA12] Mohsen Taheriyan, Craig A Knoblock, Pedro Szekely, and José Luis Ambite. Rapidly integrating services into the Linked Data cloud. In *International Semantic Web Conference (ISWC)*, pages 559–574. Springer, 2012.

[Twi] Twitter Wrapper. http://km.aifb.kit.edu/services/twitterwrap/ [accessed June 2017].

[VV] Pierre-Yves Vandenbussche and Bernard Vatant. Linked Open Vocabularies (LOV). http://lov.okfn.org [accessed June 2017].

[Wea] Weather Underground. A weather API designed for developers. https://www.wunderground.com/weather/api/ [accessed June 2017].

[Wen17] Santos Wendell. ProgrammableWeb API directory eclipses 17,000 as API economy continues surge. *ProgrammableWeb*, March 2017.

[Wor] Wordpress. https://wordpress.org/ [accessed June 2017].

[YHM11] S Hassas Yeganeh, Oktie Hassanzadeh, and Renée J Miller. Linking semistructured data on the web. *14th International Workshop on the Web and Databases (WebDB)*, June 2011.

[Zap] Zapier. The Zapier website. https://zapier.com/ [accessed June 2017].

[ZRM$^+$16] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for Linked Data: A survey. *Semantic Web*, 7(1):63–93, 2016.

Acknowledgments

I would like to thank to all the people who has "suffered" this thesis.

First of all, I thank my supervisors, Prof. Oscar Díaz and Prof. Jon Iturrioz, for their patience, guidance and support.

I owe gratitude to Jesús Ibáñez, Oscar Barrera and Patxi Echarte, the mates with whom I started this journey.

Onekin members and my sister reminded me that finishing the thesis is possible. I always remember Maider Azanza cheering me up with the "de derrota en derrota hasta la victoria final" motto. My sister, Maider Azpeitia, told me "hay vida después de la tesis", and I added "pues nadie ha vuelto para confirmarlo". I express my gratitude to the present and past members of Onekin for their accompaniment (I hope not to forget anyone): Arantza Irastorza, Maider Azanza, Iñigo Aldalur, Luis M. Alonso, Cristóbal Arellano, Oscar Barrera, Josune De Sosa, Jokin García, Felipe Ibáñez, Felipe Martín, Haritz Medina, Leticia Montalvillo, Itziar Otaduy, Iñaki Paz, Juanan Pereira, Sandy Pérez, Jeremías Pérez and Gorka Puente.

Thank you.

My family deserves an especial appreciation. My parents and my sister because they allowed me to occupy their houses. My brother Asier and my sister-in-law Zita because ... they were there. Above all, I must thank Gloria, Beñat and Alba because they really suffered the consequences of my anxiety, but nevertheless they motivated me to finish the thesis.

Bereziki zuri pottola. Eskerrik asko. Alba, ¡te quiero!

Iker Azpeitia teaches MATLAB
in the Degree in Renewable Energies.
Faculty of Engineering of Gipuzkoa in Eibar.

Summary

Linked Data Wrappers (LDWs) turn Web APIs into RDF end-points, leveraging the LOD cloud with current data. This potential is frequently undervalued, regarding LDWs as mere by-products of larger endeavors, e.g. developing mashup applications. However, LDWs are mainly data-driven, not contaminated by application semantics, hence with an important potential for reuse. If LDWs could be decoupled from their breakout projects, this would increase the chances of LDWs becoming truly RDF end-points. But this vision is still under threat by LDW fragility upon API upgrades, and the risk of unmaintained LDWs. LDW curation might help. Similar to dataset curation, LDW curation aims to clean up datasets but, in this case, the dataset is implicitly described by the LDW definition, and “stains” are not limited to those related with the dataset quality but also include those related to the underlying API. This requires the existence of LDW Platforms that leverage existing code repositories with additional functionalities that cater for LDW definition, deployment and curation.
This dissertation contributes to this vision through:
(1) identifying a set of requirements for LDW Platforms;
(2) instantiating these requirements in SYQL, a platform built upon Yahoo's YQL;
(3) evaluating SYQL through a fully-developed proof of concept; and
(4) validating the extent to which this approach facilitates LDW curation.