A BASILar Approach for Building Web APIs on top of SPARQL Endpoints

Enrico Daga, Luca Panziera, and Carlos Pedrinaci

Knowledge Media Institute (KMI) - The Open University.
Walton Hall, MK76AA Milton Keynes, United Kingdom
{enrico.daga,luca.panziera,carlos.pedrinaci}@open.ac.uk
http://kmi.open.ac.uk

Abstract. The heterogeneity of methods and technologies to publish open data is still an issue to develop distributed systems on the Web. On the one hand, Web APIs, the most popular approach to offer data services, implement REST principles, which focus on addressing loose coupling and interoperability issues. On the other hand, Linked Data, available through SPARQL endpoints, focus on data integration between distributed data sources. The paper proposes BASIL, an approach to build Web APIs on top of SPARQL endpoints, in order to benefit of the advantages from both Web APIs and Linked Data approaches. Compared to similar solution, BASIL aims on minimising the learning curve for users to promote its adoption. The main feature of BASIL is a simple API that does not introduce new specifications, formalisms and technologies for users that belong to both Web APIs and Linked Data communities.

1 Introduction

Nowadays, the World Wide Web became the most effective medium to expose data and information. A growing amount of organisations publish open data in order to provide added value services for stakeholders and customers. The increasing availability of open data sources enable the development of systems that consume distributed data to setup innovative end-user services and new businesses [11, 7]. However, the heterogeneity of methods and technologies to publish open data is still an issue to develop distributed systems on the Web.

The scientific community already identified the concerns for building distributed applications based on online services that provide data and functionalities [8]. The four key aspects to be considered for data services are: (i) support of seamless *data integration* between distributed sources; (ii) *loose coupling* and (iii) *interoperability* between data services and data consumer applications; (iv) *description* of the service interfaces to allow service search and proper data consumption by users and application.

Currently, the most adopted approach for publishing and consuming open data between distributed agents on the Web are Web APIs. Based on REST principles [3], Web APIs put a strong accent on addressing loose coupling and interoperability. More recently, the Semantic Web community started promoting

the adoption of Linked Data principles [5], which focus on integration of open data between distributed Web sources, enabled by RDF and SPARQL endpoints.

Web APIs and SPARQL endpoints have complementary advantages and limits with respect to the concerns listed so far. In the current scenario, the ideal solution is the combination of the two technologies in order to benefit of the advantages from both REST and Linked Data. Nevertheless, to be adopted, an integrated solution has to consider that the developer community behind Web APIs has different background knowledge and skills compared with developers from the Linked Data community. For instance, average Web API users do not know SPARQL or RDF. In this context, this paper aims to answer the following question: *How is it possible to integrate the advantages of the two approaches in a solution that minimises the barrier to adoption for developers from both Web APIs and Linked Data communities?*

The solution that we propose is BASIL (Building Apis SImpLy), an approach to mediate between SPARQL endpoints and applications by generating Web APIs from SPARQL queries stored and managed by an intermediate system. BASIL have been designed to minimise the learning curve for its users. Data consumers, which belong to the Web API community, do not require SPARQL and RDF expertise to benefit of the open data. SPARQL, which is already a data provider skill, is used to tailor Web APIs for data consumers.

The paper is structured as follows. Section 2 provides the motivations of this work based on the real experience on publishing open data through the `data.open.ac.uk` portal. The methodology and the requirements specification for modelling BASIL is available in section 3. The proposed solution is described in Section 4, while section 5 evaluates the benefits of BASIL for data consumers and providers. Related work is discussed in section 6. Finally, section 7 provides conclusions and future work.

2 Motivation and Case Study

The Open University, which the authors are affiliated, provides its own institutional repositories as Linked Open Data through the `data.open.ac.uk` web portal [2]. The portal makes available public data about academic degree qualifications, courses, scholarly publications and educational resources of the University. Several Web applications are currently to obtain official information by exploring linked resources spread across the heterogeneous landscape of systems, websites and repositories of the Open University. Since 2010, the data portal involved more then five developers from different units of the University as data consumers, and two developers as data providers, taking care of the database maintenance, optimisation and evolution. During this experience, developers and providers interacted frequently in order to cope with technical and design issues. From this use case, the authors collected feedback looking backwards to conversations in emails and informal discussions. While this feedback is not intended to be understood as the sole truth of potential views, opinions or problems, indeed it represents the issues that different developers (both consumers and providers)

have been facing in our use case. Nevertheless, we believe that the identified issues are broadly relevant.

Consumers of `data.open.ac.uk` understand the benefit of RDF and SPARQL for data integration, particularly relating assets from different sources to qualifications, courses and researchers of the University, but the two standards are not a common skill of Web developers. Therefore, data consumers ask support from providers for modelling and embedding SPARQL queries in their applications. This approach has two side effects. First, embedding SPARQL queries in applications violates the principle of loose coupling of distributed systems, because it creates a strong dependency between the source data model and the client software. As consequence, data providers quickly loose a sight on the systems relying on the data, thus loosing the capability of assessing the impact of schema evolutions on existing queries. Second, modelling an efficient and consistent query may require days of hard work. This approach does not allow other consumers to reuse embedded queries. In addition, SPARQL makes harder the implementation of caching solutions for providers, because each requirement can be expressed by queries with same semantics but different syntaxes.

The feedback highlights that Web APIs are well known by Web developers due their high diffusion. The fixed interface of the Web APIs assures the consumer stability in the representation, and implements a proper loose coupling. However, the Open University's dataset evolves frequently. Each dataset evolution may require adaptations of the Web API interface, which are time consuming for the data provider and can break existing client systems. Moreover, Web APIs do not allow consumers to benefit of data integration features provided by SPARQL.

3 Methodology and requirements specification

To face the issues raised by the case study, we perform an analysis by applying the SWOT framework [6] to compare SPARQL endpoints and Web APIs as methods for publishing open data. The analysis starts by (i) collecting the issues and assign each to SPARQL endpoints or Web APIs. Then, (ii) the issues are classified as Strength ($\boldsymbol{S}$), Weaknesses ($\boldsymbol{W}$), Opportunities ($\boldsymbol{O}$) or Threats ($\boldsymbol{T}$). This phase includes removal of duplicate issues, merging of similar concepts and abstraction of related issues. Afterwards, (iii) each issue is known to affect Data Consumers (`C`) and/or Data Providers (`P`). Finally, (iv) issues are classified according to the data services principles: Data Integration (di), Interoperability (io), Loose Coupling (lc), and Description (de). Along with these key aspects, we add a general principle: Adoption (ad), that we intend as the cost of the approach adoption in terms of time, effort or resources for both data providers and consumers. Table 1 shows the results of the analysis. During the development of the analysis, we realised that strengths and opportunities of Web APIs overcome weaknesses and threats of SPARQL endpoints, and viceversa. To help the reader, we ordered the issues accordingly. For instance: $S1_S \rightarrow O1_S \rightarrow W1_A \rightarrow T1_A$.

The analysis provides a clear overview of the practical implications of the two paradigms for both data providers and data consumers. Additionally, it shows that SPARQL endpoint and Web APIs are complementary.

Issue	Description	Affects	Aspect
	SPARQL endpoint		
$S1_S$	SPARQL is a rich query language, capable of selecting any portion of the data and to exploit relations and paths between resources.	P,C	*di*
$S2_S$	The output can be an RDF graph, a semantic meta model that generalizes from specific syntaxes.	C	*di*,*io*
$S3_S$	The interaction is a standard protocol.	P,C	*io*,*de*
$W1_S$	RDF and SPARQL are not widespread technologies, and the related data model may not be optimised for the needs of a dedicated application	C	*di*,*ad*
$W2_S$	Some requests might require too many resources (CPU, RAM).	P,C	*ad*
$W3_S$	Embedded queries in the consumer's code add a dependency between the application and the provider's data schema.	C	*lc*
$O1_S$	The SPARQL query language allows for a deep data exploration and design of task tailored views.	C	*di*
$O2_S$	The RDF output can be integrated with other RDF data with little effort.	C	*di*,*io*
$O3_S$	The data provider maintains a standard infrastructure.	P	*ad*
$T1_S$	Data consumers may decide not to use the service because of a too steep learning curve (both for querying or post-processing of the output).	C	*ad*
$T2_S$	The data provider cannot optimize the infrastructure in advance (or contribute to optimize the query) and the system could crash.	P	*lc*
$T3_S$	Changes in the data schema will break existing embedded queries.	C	*lc*
	Web API		
$S1_A$	Web APIs can be made simple and intuitive, the data models are made ad-hoc for specific tasks and reused by a wide community of developers.	C	*di*,*io*,*de*
$S2_A$	Resources of the underlying infrastructure are controlled.	P	*lc*
$S3_A$	Each resource (request) is fully decoupled from the underlying database schema.	P,C	*lc*
$W1_A$	The set of possible requests and data objects is preordered.	C	*di*
$W2_A$	The output data model cannot be customized to better fit the use case.	C	*io*
$W3_A$	Interfaces and documentation need to be setup and maintained.	P	*de*,*ad*
$O1_A$	Web developers can use the service straight forward to integrate the output in the applications. Data models are reused by different applications.	C	*di*,*ad*
$O2_A$	The infrastructure can be easily optimized.	P	*lc*
$O3_A$	Evolutions in the stored data model in many cases can be reflected in the way the API interacts with the database wihout disrupting existing applications.	P,C	*lc*
$T1_A$	The supported APIs may not cover relevant use cases, and may be hard to extend.	P,C	*di*,*io*,*ad*
$T2_A$	Data consumers cannot easily implement data integration strategies.	C	*di*
$T3_A$	The cost for maintaining infrastructure and documentation increases with the amount of functionalities/data provided.	P	*ad*

Table 1: SWOT Analysis results.

	Description	**Enforce**	**Limits**
di	Explorable data, to be extracted and reused as RDF as well as non-standard formats.	$S1_S$ $O1_S$ $O1_S$ $O2_S$ $S1_A$ $O1_A$	$W1_S$ $T1_A$ $T2_A$
io	Customizable and reusable data models, relevant for both consumers and providers, and formally specified.	$S2_S$ $S3_S$ $O2_S$ $S1_A$	$W2_A$ $T1_A$
lc	Do not introduce dependencies between the systems, both sintactically and semantically	$S2_A$ $S3_A$ $O2_A$ $O3_A$	$S3_S$ $T2_S$ $T3_S$
de	Described for both human and agents with small effort	$S3_S$ $S1_A$	$W3_A$
ad	No additional technologies, specifications or formalisms that introduce learning effort for both data consumers and providers. Provide opportunities for better sustainability (monitoring, caching).	$O3_S$ $O1_A$	$W1_S$ $W2_S$ $T1_S$ $W3_A$ $T1_A$ $T3_A$

Table 2: Requirements

From this analysis, we extract the requirements listed in Table 2. These requirements highlight that the ideal solution would integrate the strengths of both approaches by reducing and possibly eliminate the weaknesses.

4 A BASILar approach

BASIL is designed as middleware system that mediates between SPARQL endpoints and applications. The architecture of BASIL is provided in figure 1. Three actors are involved: *data providers*, *data consumers* and *Web API tailors*. Data providers focus on maintenance and evolution of their SPARQL endpoints and data provided. Data consumers use tailored Web APIs as mean to access data. BASIL introduces Web API tailors as a new actor of the process of publishing and consuming data. Tailors model Web APIs for data consumers through *API specifications*. An API specification defines: a data source, and the portion of data to be returned by the API, and the input parameters. A SPARQL query is the formalisms used to specify input parameters and output data of a single API. Queries are stored in the BASIL middleware. Each time an API is consumed, its query is executed on the related endpoint, then the query result is returned. In addition, tailors can specify *views* for each Web API. A view is an alternative presentation of an API results based on a template or script[1]. In a concrete scenario, the Web API tailor can be a member of the data provider organisation that builds APIs for data consumers, which are not SPARQL experienced, or a data consumer who has SPARQL knowledge and prefers to benefit of the advantages of Web APIs.

[1] E.g., a view can be a HTML representation of the results defined by a XSLT or a transformation of the API output by maintaining the data format. The advantage of views is to adapt the output of a Web API to applications with special needs. For instance, a view can be handy to develop snippets to be embedded in web pages with no further processing. Our definition of *view* is inherited from the popular Model-View-Controller (MVC) pattern.

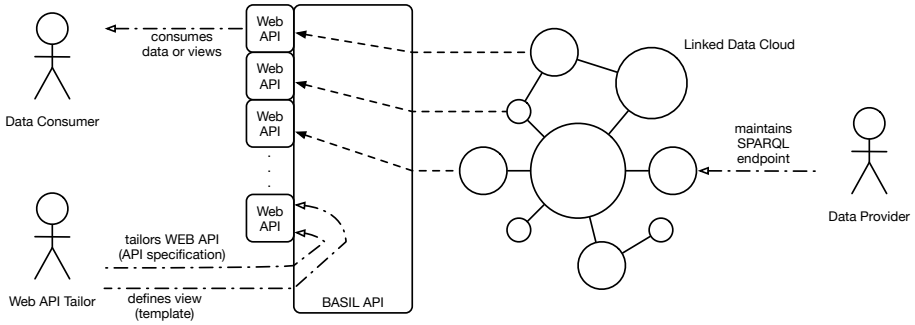

Fig. 1: The BASIL architecure

BASIL[2] provides as a CRUD API over HTTP[3]. To create a new API, tailors define a SPARQL query. For instance, listing 1.1 provides a real query modelled by a maintainer of `data.open.ac.uk` to support a developer requesting to extract a list of open educational resources related to a given qualification[4]. The example query returns extracts, video, text, audio of current courses that are related to qualifications provided by the Open University (e.g., Master degree in Computer Science). Each qualification has an ID code (e.g., q18). In order to make the qualification ID an input parameter of the tailored API, the variable `?_qid` has been defined. BASIL considers a mapping between API parameters and SPARQL variables by adding a underscore in the begin of the variable name as convention. Details on variable name conventions for parameters mappings are provided in Table 3. The creation of the API is performed with a HTTP `PUT` request to `http://basil.kmi.open.ac.uk/basil?endpoint=http://data.open.ac.uk/sparql`.

The query parameter `endpoint` defines the SPARQL endpoint and the request body contains a SPARQL query that defines the view on the dataset. This operation triggers the generation of a set of resources:

`/basil/x68shwt3Qw` → base resource, redirects to `/spec`
`/basil/x68shwt3Qw/api` → to retrieve the data
`/basil/x68shwt3Qw/spec` → to get and update the stored query
`/basil/x68shwt3Qw/explain` → to inspect the query after variables substitution
`/basil/x68shwt3Qw/view` → to manage views
`/basil/x68shwt3Qw/api-docs` → to access the Swagger description

In order to support data consumers, BASIL makes available the Swagger[5] documentation of the API as subresource of the API specification. Swagger is chosen because it provides interactive documentation which allow developers to test the API before the integration in the application. A Web API can be consumed

[2] The source code is available at `https://github.com/the-open-university/basil`
[3] The documentation is available at `http://basil.kmi.open.ac.uk/docs`
[4] The complete SPARQL query is available at `https://gist.github.com/enridaga/3b71423df42328ea2110`
[5] `https://github.com/swagger-api/swagger-spec`

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
...
SELECT DISTINCT
 (?related as ?identifier) ?type (STR(?label) AS ?title) (STR(?location) AS ?link)
 FROM <http://data.open.ac.uk/context/youtube>
 FROM <http://data.open.ac.uk/context/audioboo>
 FROM <http://data.open.ac.uk/context/podcast>
 FROM <http://data.open.ac.uk/context/openlearn>
 FROM <http://data.open.ac.uk/context/course>
 FROM <http://data.open.ac.uk/context/qualification>
 FROM <http://data.open.ac.uk/context/xcri>
WHERE {
 BIND(IRI(CONCAT("http://data.open.ac.uk/qualification/",?_qid)) AS ?qualification)
 { # related video podcasts
   ?related podcast:relatesToQualification ?qualification .
   ?related a podcast:VideoPodcast .
   ?related rdfs:label ?label .
   optional { ?related bazaar:download ?location }
   BIND( "VideoPodcast" as ?type ) .
 } UNION { # related audio podcasts ...
 } UNION { # related audioboo posts ...
 } UNION { # related openlearn units ...
 } UNION { # related youtube videos
   ?related a schema:VideoObject .
   ?related yt:relatesToQualification ?qualification .
   BIND("YoutubeVideo" AS ?type) .
   ?related media:download ?location .
   ?related rdfs:label ?label .
 }
}
```

Listing 1.1: SPARQL query that specifies a tailored API

SPARQL variable name	**Description**
?_<name>	The variable specifies the API parameter <name> (mandatory, by default). The value is encorporated in the query as plain literal.
?__<name>	The parameter <name> is optional.
?_<name>_iri	The variable is substituted with the parameter value as a IRI.
?_<name>_<lang>	The parameter value is considered as literal with the language <lang> (e.g., en,it,es, etc.).
?_<name>_<xsd-datatype>	The specification includes an XSD data type (e.g., integer,date, etc.) .
?_<name>_<prefix>_<type>	The specification includes a custom data type: <prefix>:<type> (e.g., rdf:HTML).

Table 3: SPARQL variable name convention for Web API parameters mapping

through a HTTP `GET` in different ways:

`/basil/x68shwt3Qw/api?qid=q18` with content negotiation
`/basil/x68shwt3Qw/api.json?qid=q18` or .xml, .rdf, .jsonld, .csv, .nt, .ttl, ...
`/basil/x68shwt3Qw/api.html-list?qid=q18` preprocess the output using the *html-list* view script

When the BASIL API receives a tailored API invocation, the value of the parameter `qid` substitutes the variable `?_qid` in the specification query that will be executed. Then, the result of the query is returned to the data consumer, according to a data format specified through content negotiation. The supported response formats are plain XML, JSON and CSV without namespaces, for data consumers that are not familiar with Linked Data, and Semantic Web Standards (such as, RDF+XML, N3 and Turtle), for SPARQL experts. For instance, listing 1.2 shows the JSON output of the query in the previous example (Listing 1.1) Moreover, users can customize the output with user-defined views using template or scripting languages (Mustache[6] and JavaScript[7], in the reference implementation).

```
{
    "vars": ["identifier", "type", "label", "link"],
    "items": [
        {
            "link": "https://audioboo.fm/boos/1695040",
            "label": "Holism:␣the␣whole␣truth␣(2\/10)@en",
            "type": "AudiobooPost",
            "identifier": "http://data.open.ac.uk/audioboo/post/1695040"
        },
        {
            "link": "https://audioboo.fm/boos/1695018",
        ...
        ]
}
```

Listing 1.2: Tailored API output as JSON

5 Evaluation of Benefits

BASIL would be extremely beneficial in a setting like `data.open.ac.uk`, even more when deployed as a cloud service for the Web of Data. Through its approach, BASIL fulfils the four requirements of a data service as follows.

Data integration is guaranteed by design, relying on SPARQL and RDF. Use cases include vocabulary rewriting, inferences materialisation, data refactoring, cleaning, and patching. In addition, the output can be customised to better fit the use case of web developers, including ready made HTML snippets to be directly embedded in web sites.

[6] `https://mustache.github.io/`
[7] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

Interoperability towards consumer applications is enforced by relying on Web APIs, and towards publishing systems using the SPARQL specification. Generated APIs can be shared and reused by different consumers, and API specifications (queries) can be exploited by data providers to analyse usage and perform optimisations on the underlying infrastructure.

BASIL guarantees **loose coupling**. Client applications do not depend directly on the remote data schema. Evolutions in the remote data service can be reflected in API specifications without changing the consumer application. Similarly, the requirements of the client application can evolve without requesting the provider to enhance the publishing infrastructure. By decoupling the data requirement specification from the retrieval operation, the middleware can implement sophisticated solutions to improve efficiency, response time, availability.

BASIL support the service **description**, by providing Swagger specifications for each tailored API. The collection of API descriptions can be published as a data catalogue. The semantics of the queries could be used to boost search and discovery of APIs or implement explanation services. With BASIL, the effort made on query design can be directly exploited by similar use cases. Data providers can access a collection of queries, and contribute to optimise them. Data providers could allocate appropriate computational resources for requests coming from a trusted middleware, and reduce the ones offered to unknown visitors. Organisations can invest in maintaining BASIL descriptions and reduce the cost of the infrastructure by simply specifying queries instead of developing Web APIs from scratch,

The BASIL approach brings the opportunity of boost a sustainable **adoption** of open data published as RDF and SPARQL, without the need for additional technologies, specifications or formalisms for both data consumers and providers, as proposed by existing approaches.

6 Related Work

Encapsulated views and stored procedures are common in relational databases [10] as methods to improve efficiency, enforce security, data integrity, and to decouple the application logic from the database schema. This tradition inspired our approach. However, distributed applications based on open data are dissimilar under some key aspects, particularly the unpredictability of the consumer's use cases. This difference lead to many of the concerns addressed in this paper, like the need for decoupling the data specification from the data consumer's application logic, but also from the data provider's system.

The Linked Data Platform[8] is a W3C recommendation to perform CRUD operations on resources exposed as Linked Data. The specification enables consuming or modifying linked data resources through REST, by packaging a single Web API serving RDF data. However, the way data is provided is full RDF, and the specification does not give recommendation on how to customise the data

[8] `http://www.w3.org/TR/ldp/`

model or distribute the different roles in the design flow. Approaches based on storing SPARQL queries on the server side have been proposed by the Linked Data API[9] specification, which have been implemented by ELDA[10] and Open PHACTS [4]. A similar facility is provided by The Data Tank[11]. As well as BASIL, both attempts hide the complexity of the SPARQL specification to the data consumer through a Web API. Nevertheless, the two approaches introduce additional formalisms for API specification, which highly increase the learning curve of potential adopters. Less recent approaches include implementation of *ad hoc* APIs to bridge the gap between Semantic Web URIs and well known codes. One example is The RDF book mashup [1]. The relation between Web services and Linked Data has been analysed in [9]. In this context, approaches to bridge the gap between services and linked data have been proposed. In [12], the authors propose a method to publish existing Web APIs as Linked Data. The same issue has been addressed by introducing functional descriptions of hypermedia services in [13]. Compared to [9, 12, 13], this paper addresses the opposite issue. BASIL exploits the benefits of Web APIs on top of SPARQL endpoints as simple and intuitive bridge between the Semantic Web and the Web developer communities.

7 Conclusion and Future Work

Web APIs and SPARQL endpoints are two complementary approaches to provide open data on the Web. The advantages of the two methods have been combined in BASIL, an approach for *Building Apis SImpLy* on top of SPARQL endpoints. With our approach data consumers, that belong to the Web API community, can access the Linked Data cloud by adopting well-known paradigms. On the other side, data providers can benefit of queries stored in BASIL for a better maintenance, optimisation and evolution of their own data services. BASIL envisage a new role in the open data consumption life-cycle: the Web API tailor, a SPARQL expert from the data provider organisation or third-party that fills the gap between Linked Data and Web API worlds.

For the future, we will apply BASIL on `data.open.ac.uk` in order to perform a user-based evaluation with developers to explore pros and cons of BASIL. We want to provide meta level RDF descriptions of APIs, including aspects like provenance (with PROV-O). The tailored APIs could be exposed also as a data catalogue, for example with DCAT. BASIL can give us the opportunity to study methods to improve availability, for example integrating an approach like linked data fragments [14], and research on new methods for query federation by composing APIs seamlessly using SPARQL from, graph or service clauses.

[9] `https://github.com/UKGovLD/linked-data-api`
[10] `http://www.epimorphics.com/web/tools/elda.html`
[11] `http://docs.thedatatank.com/4.3/spectql` and `http://docs.thedatatank.com/4.3/sparql`

References

1. Bizer, C., Cyganiak, R., Gauss, T.: The RDF Book Mashup: From Web APIs to a Web of Data. In: Proc. of the Workshop on Scripting for the Semantic Web (2007)
2. Daga, E., d'Aquin, M., Adamou, A., Brown, S.: The Open University Linked Data - data.open.ac.uk. Semantic Web Journal (2015), to appear.
3. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California - Irvine (2000)
4. Groth, P., Loizou, A., Gray, A.J., Goble, C., Harland, L., Pettifer, S.: API-centric Linked Data integration: The Open PHACTS Discovery Platform case study. Web Semantics: Science, Services and Agents on the WWW 29(0), 12 – 18 (2014)
5. Heath, T., Bizer, C.: Linked data. Synthesis Lectures on the Semantic Web: Theory and Technology 1(1), 1–136 (2011)
6. Hill, T., Westbrook, R.: SWOT analysis: it's time for a product recall. Long range planning 30(1), 46–52 (1997)
7. Huijboom, N., Van den Broek, T.: Open data: an international comparison of strategies. European journal of ePractice 12(1), 4–16 (2011)
8. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: a Research Roadmap. International Journal of Cooperative Information Systems 17(2), 223–255 (2008)
9. Pedrinaci, C., Domingue, J.: Toward the next wave of services: Linked Services for the Web of data. Journal of Universal Computer Science 16(13), 1694–1719 (2010)
10. Ramakrishnan, R., Gehrke, J.: Database management systems. Osborne/McGraw-Hill (2000)
11. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: The Semantic Web–ISWC 2014, pp. 245–260. Springer (2014)
12. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: The Semantic Web: Research and Applications, pp. 170–184. Springer (2011)
13. Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Vallés, J.G., Van de Walle, R.: Functional descriptions as the bridge between hypermedia apis and the semantic web. In: Proc. of the WS-REST workshop. pp. 33–40. ACM (2012)
14. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-scale querying through linked data fragments. In: Proceedings of the 7th Workshop on Linked Data on the Web (2014)