Facilitating the Exploitation of Linked Open Statistical Data: JSON-QB API Requirements and Design Criteria

Dimitris Zeginis[1,2], Evangelos Kalampokis[1,2], Bill Roberts[3], Rick Moynihan[3], Efthimios Tambouris[1,2], and Konstantinos Tarabanis[1,2]

[1] Information Technologies Institute, Centre for Research & Technology Hellas, Thermi, Greece
[2] University of Macedonia, Thessaloniki, Greece
{zeginis, ekal, tambouris, kat}@uom.gr
[3] Swirrl IT Limited, 20 Dale Street, Manchester, M1 1EZ, United Kingdom
{bill, rick.m}@swirrl.com

Abstract. Recently, many organizations have opened up their data for others to reuse. A major part of these data concern statistics such as demographic and social indicators. Linked Data is a promising paradigm for opening data because it facilitates data integration on the Web. Recently, a growing number of organizations adopted linked data paradigm and provided Linked Open Statistical Data (LOSD). These data can be exploited to create added value services and applications that require integrated data from multiple sources. In this paper, we suggest that in order to unleash the full potential of LOSD we need to facilitate the interaction with LOSD and hide most of the complexity. Moreover, we describe the requirements and design criteria of a JSON-QB API that (i) facilitates the development of LOSD tools through a style of interaction familiar to web developers and (ii) offers a uniform way to access LOSD. A proof of concept implementation of the JSON-QB API demonstrates part of the proposed functionality.

Keywords: Linked data, statistical data, data cube, API, JSON, requirements

1 Introduction

Increasingly, many governments, organisations and companies are opening up their data for others to reuse through *Open Data* portals [12]. These data can be exploited to create added value services, which can increase transparency, contribute to economic growth and provide social value to citizens [9].

A major part of open data concerns statistics (e.g. economical and social indicators) [2]. These data are often organised in a multidimensional way, where a measured fact is described based on a number of dimensions. In this case, statistical data are presented as data cubes.

Linked data has been introduced as a promising paradigm for opening up data because it facilitates data integration on the Web [1]. Concerning statistical data, standard vocabularies such as the RDF data cube (QB) vocabulary[4], SKOS[17] and XKOS[3] enable modelling data cubes as Linked Open Statistical Data (LOSD).

Although LOSD potential is high, their exploitation is low for two reasons. First, using LOSD requires skills and tooling (e.g. RDF, SPARQL) that are less widespread than some other web technologies (e.g. JSON, Javascript). For example, there are many Javascript visualization libraries that consume JSON data (e.g D3.js, charts.js), while there are just a few that consume RDF and their functionality is limited. Second, many portals that use the standard vocabularies often adopt different publishing practices [10], thus hampering their interoperability. As a result it is difficult to create software tools that can be reused across LOSD. Usually, developed tools assume that data are published only in a specific form.

In order to unleash the full potential of LOSD there is a need to standardize the interaction (i.e. input, output and functionality) with LOSD in a way that facilitates the development of reusable software. This paper describes the requirements and design criteria of a JSON-QB API that aims to exploit the advantages of LOSD (e.g. easy data integration) while making data available in a structure and format that is familiar to a larger group of developers. Some of the flexibility, and associated complexity, of linked data is removed, in favour of simplicity and ease of use. Moreover, the API offers a uniform way to access the data, thus enabling the development of generic software tools that can be reused across datasets.

The rest of the paper is organized as follows, section 2 explains the motivation for the development of a JSON-QB API, section 3 presents related work, section 4 defines the requirements and design criteria for JSON-QN API. Section 5 presents a proof of concept implementation of the API. Finally, section 6 draws conclusions.

2 Motivation

Currently, many LOSD have been made available on the Web through official portals. For example Census data of 2011 from Ireland[4] and Italy[5] have been published as linked data. The Department for Communities and Local Government (DCLG)[6] in the UK, the Scottish Government[7], and the Statistics Bureau of Japan[8] opened up their statistics as linked data etc.

Although the above portals use the same standard vocabularies, they often adopt different publishing practices [10]. For example different practices are

[4] http://data.cso.ie
[5] http://datiopen.istat.it
[6] http://opendatacommunities.org/data
[7] http://statistics.gov.scot
[8] http://data.e-stat.go.jp

adopted for the definition of multiple measures, for the definition of popular dimension (i.e. time, geography) and their code lists etc. As a result, generic tools that operate across LOSD datasets cannot be created. However, tools which assume LOSD published only in a specific way have already been developed. Specifically, existing tools enable: i) the browsing of LOSD e.g. Data Cube faceted browser[15], CODE Query wizard[8] ii) the performance of OLAP operations like roll-up/drill-down, slice, dice e.g. OpenCube OLAP Browser [13], QB2OLAP[22] iii) the performance of statistical analysis on LOSD e.g. OpenCube R statistical analysis tool[11] and iv) the visualization of LOSD e.g. CubeViz[16], StatSpace[6].

In addition to the exploitation tools, complete platforms (e.g. PublishMyData[9]) aim both to publish and exploit LOSD. In this case, published data can be consumed only by tools of the same platform, since different publishing practices are adopted. This leads to the creation of LOSD system silos (software & data) that cannot interoperate among each other.

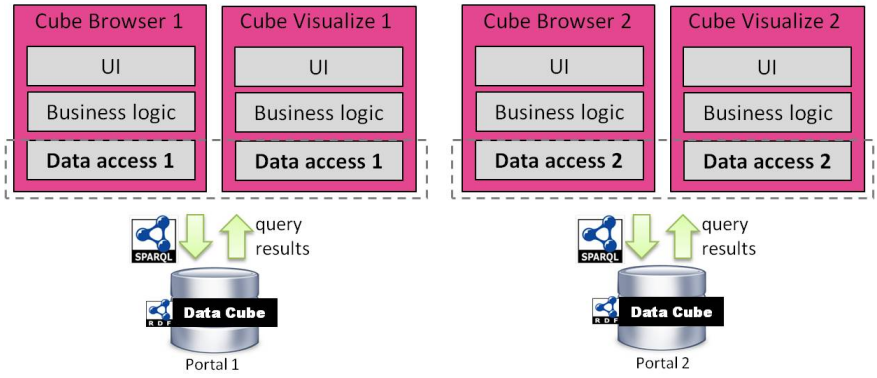

Fig. 1. Traditional architecture for LOSD tools

All the above tools and platforms follow the same traditional architecture (figure 1) where each tool has an integrated access layer. If several tools are created for the same portal (i.e. same publishing practices), then each tool has to develop separately a similar data access layer. In addition, if a tool has to be used at another portal, then a new data access layer has to be created leading to additional costs. More importantly, the development of data access layers requires significant programming expertise in LOSD, a skill that is not widely available between developers.

As a result, there is a need to standardize the interaction with LOSD in a way that hides the LOSD complexity to the developers and offers a uniform way to access the data. To achieve these objectives we adopt the following methodology:

[9] http://www.swirrl.com/

(i) study the related work focusing on APIs that facilitate the interaction with data cubes and statistical data and (ii) collect user requirements from developers that create LOSD applications.

3 Related work

Currently, several APIs that standardize the interaction with multi-dimensional statistical datasets have been developed. For example, SDMX has proposed the SDMX-REST API [20] that offer programmatic access to data and metadata disseminated in an SDMX-compliant source. This API is currently used by several organisations including the Eurostat, OECD and World Bank. Eurostat offers also the "JSON & UNICODE Web Services"[10] to allow access to its data. This service is complementary to the SDMX-REST API since it supports different output formats (i.e. JSON and UNICODE). Another REST API that enables the accessing of multidimensional databases is offered by PX-Web [11] internet server application. The API is used by many National Statistics Offices including Finland, Sweden, Estonia and Switzerland. Table 1 presents a summary of the offered functionality of the above three APIs. The functionality is separated to three main categories, namely "search", "get meta-data" and "get data".

These APIs focus on supplying metadata and data about a specific dataset or cube. They do not address requirements regarding the combination of data from multiple datasets or cubes. Although two of the APIs in Table 1 support search functionalities, the search provides limited filtering options such as free text search and search based on the category of the indicator. As a result they do not support the discovery of datasets that are structural compatible to integrate.

Additionally, APIs that support advanced OLAP operations on data cubes, such as aggregation, slice and roll-up/drill-down, have been proposed. For example, the Oracle OLAP Java API [19] allows users to select, explore, aggregate and perform analytical tasks on data stored in an Oracle data warehouse. Olap4j[12] is another Java API for accessing data cubes stored at OLAP servers. It supports Multidimensional Expressions (MDX) that is the query language for OLAP.

Regarding the output, the APIs support many formats including, SDMX-JSON [21], SDMX-ML, JSON-stat[13], CSV, Unicode, PC-Axis etc. An interesting JSON extension for encoding linked data is JSON-LD[14]. JSON-LD is not currently used by existing APIs, however it is a candidate for a JSON-QB API.

Finally, APIs that hide the complexity of SPARQL endpoints have been proposed. For example OpenPHACTS [7] and BASIL [5] propose approaches to build Web APIs on top of SPARQL endpoints. Grlc is a lightweight server that takes SPARQL queries curated in GitHub repositories, and translates them to Linked Data APIs on the fly. These API, succeed in hiding the complexity of

[10] http://ec.europa.eu/eurostat/web/json-and-unicode-web-services/
[11] http://www.stat.fi/tup/pcaxis/px_web_ominaisuudet_en.html
[12] http://www.olap4j.org
[13] https://json-stat.org/
[14] https://json-ld.org

SPARQL to web developers, however they are generic and do not provide cube related operations.

Table 1. Existing APIs used by national and international organisations

Functionality		**SDMX-REST**	**JSON & Unicode Web Service**	**PX-Web**
Search	Get cubes	Filtering options: – data provider – API version	No	Filtering options: – free text search – category search – geography search
Get meta-data	Data structure	Yes	No	Yes
	Concept scheme (qb:concept)	Yes	No	No
	Code List	Yes (including hierarchical code lists)	No	Yes
	Data publisher	Yes	No	No
	Data category	Yes	No	Yes
Get data	Observations	Filtering options: – dimension values (multiple values, AND/OR, ALL) – start/end date – updateAfter – data provider	Filtering options: – dimension values (multiple values, AND) – unit – precision	Filtering options: – dimension values (multiple values, AND)
	Time series	Yes	No	No
	Compatible data	Yes	No	No

4 Requirements and design criteria

The API should follow patterns and practices familiar to "mainstream" web developers, to facilitate the creation of data-driven visualisations and interactive applications. Moreover, it should be suitable for use by a wide range of statistics publishing organisations, so that data users can have a standard interface to LOSD. This will put constraints on the way that publishers manage their data, however those constraints should be reasonable and manageable.

To collect the requirements, we established an ongoing interaction with developers that currently create applications for LOSD. The interaction mainly occurs within the EU funded project OpenGovIntelligence[15], which aims to exploit LOSD for improving the public services. To facilitate the collection of requirements we organized a dedicated workshop in Manchester with participation of relevant developers.

[15] http://www.opengovintelligence.eu/

4.1 Search data cubes

The LOSD cloud currently contains many data cubes and their number still increases. Thus, applications need to search for cubes based on some criteria. For example, get cubes that measure unemployment, or get cubes for Greece. The search criteria can be even more complex e.g. get cubes about unemployment in Greece after 2010. Thus, the API should provide a flexible way to express complex data queries. A parameter that should also be taken into consideration is the support of multiple natural languages, for example in helping to match search terms with concepts that could have multi-lingual labels.

The search functionality can also be extended to support not only user specific data queries, but also support the "automatic" search of compatible cubes that could be processed together. For example, having a cube at hand search for other cubes that are compatible for combined statistical analysis, for visualisation or for browsing. The compatibility search needs to access both the structure and the data of the cube. However, the compatibility criteria is still an open issue and is out of the scope of this paper.

4.2 Get cube meta-data

Once a cube has been identified (e.g. through the search functionality) the processing application (e.g. cube browser) needs to initialize the user interface or the analysis with information related to the cube structure. For example, populate drop-down menus with the cube dimensions and measures. The QB vocabulary clearly identifies the main elements of the structure that should be accessed through the JSON-QB API:

- Dataset meta-data. They include information like the label, description, issue date, publisher and license.
- Dimensions. They include all the dimension properties of the cube (e.g. reference area, reference period).
- Measures. They include all the measure properties of the cube (e.g. unemployment, poverty)
- Attributes. They include all the attribute properties of the cube (e.g. unit of measure)
- Dimension values. They include all the values of a dimension (e.g. male, female) that appear at the cube.
- Dimension levels. In the case of hierarchical data, dimension values are organized to hierarchical levels (e.g. region, district).
- Attribute values. They include all the values of an attribute (e.g. euro, dollar) that appear at the cube.

Regarding the last three elements, the QB vocabulary does not offer a way to retrieve the values / levels directly from the structure. Thus the API should iterate over the cube observations, which is a time consuming task.

4.3 Slicing and filtering

There are already methods available for downloading entire data cubes but people often want just small parts. Whole cubes are often too big to be well-suited to interactive applications, and if the data updates frequently, then it's important for people to be able to retrieve up-to-date extracts of the data, rather than keeping their own copies of full datasets up to date. The JSON-QB API should provide a flexible way to applications to take exactly the data they need by defining constrains (i.e. filters). For example it should support many filtering options to the dimension values including:

- Single values e.g. refPeriod=2010.
- Multiple values e.g. refPeriod=[2010, 2011, 2012]
- Ranges e.g refPeriod=[2010 ... 2015]
- Greater/smaller than e.g. refPeriod>2010
- Hierarchical data filtering e.g. refArea="all council areas in Scotland"

In many cases, applications do not need all the requested data at once, because they process them at bunches. For example, a cube browser shows a part of the data allowing the user to navigate to the previous/next page of data. Thus, the JSON-QB API should support paging and ordering of the results. The ordering of the results can be in ascending or descending order based on a dimension. However, in some cases this is a complicated task e.g. ordering based on 2 ore more dimension. Moreover, lexicographical ordering is not always appropriate (e.g. for the days of the week), thus other types of ordering should be applied.

4.4 Ease of use

Linked data offer many benefits to web developers, including the easy integration on the web. However, linked data technologies (i.e. RDF, SPARQL) are unfamiliar to many developers, thus hindering their adoption. The purpose of the JSON-QB API is to exploit the advantages of linked data through a style of interaction that is familiar to web developers, thus helping them create data visualisations and applications. It is not necessary for the API to be a complete "round-trippable" representation of the data, it is acceptable to lose some information in favour of greater ease of use.

The ease of use of an API is related both to the input and the output. Regarding the input of the API there are mainly two design options: i) use a separate REST parameter for each input and ii) model all the input as a JSON object. The first option was traditionally used by APIs, while the second is recently becoming popular since it is more flexible and enables the creation of a data query language for the API. For example, using JSON objects is easier to express relations other than equality e.g. greater than, while using parameters is more awkward as custom encoding conventions should be used which require extra processing on part of the developer.

Regarding the output of the API, JSON is a popular, easy to use format. Usually, applications and visualizations do not require an n-array/ tabular response

(e.g. JSON-stat); an array of observations is sufficient and more straightforward. In case that a tabular response is required, then it can easily be constructed from the observations. While JSON-QB API aims in hiding some of the complexity of linked data, responses should include URIs as identifiers of key entities (e.g. JSON-LD), to retain the connection to data on the web and to support reliable combining of data from different sources within a data consuming application.

4.5 Uniform data access

Currently many LOSD have been published, however a lot of them adopt different publishing practices. The JSON-QB API should work on top of any of these data, offering uniform access to the data. Obviously, this will require separate implementations to comply with the different publishing practices.

Ideally, the standardization of the JSON-QB API specification will also contribute to the formulation of an application profile for the QB vocabulary. The profile will include best practices that can be used by data publishers to provide data in a compatible way, facilitating in this way the development of generic LOSD tools. This will add some constraints on the way that publishers manage their data, however those constraints will lead to greater exploitation of the data.

4.6 High performance

The volume of LOSD is big, reaching the magnitude of million triples per cube. Thus, SPARQL queries that iterate over all the observations tend to be slow. For example, a query to get all the dimension values that appear at a cube needs to iterate over all the observations.

The JSON-QB API can improve performance of demanding SPARQL queries through efficient caching of the responses. The caching policy (e.g. Least Recently Used, Least Frequently Used) plays an important role at the performance improvement. Note that caching of API responses is much easier that caching of arbitrary SPARQL queries. Allowing a SPARQL query to run on a collection of data means that if any of the data changes, it is possible that the query response changes. It is complex to analyse which queries touch a particular data cube or particular part of the data, thus making cache clearing difficult. With the API call, most requests will return data from individual data cubes, so it is easier to know which cached responses must be invalidated when data is updated.

Another task that can improve the performance of the API is the pre-computation of aggregations: i) across a dimension of the cube e.g. compute the SUM of the sales over time and thus ignore the time dimension of the cube and ii) across a hierarchy e.g. if a cube contains the election results at municipality level, then aggregations can be computed at region and at country level. The pre-computation of the aggregations facilitates the execution of queries, because there in not the need to compute the aggregations on-the-fly when requested.

Finally, the performance and network traffic can be improved by returning exactly the data requested. This can be achieved through a flexible data query

language. In this way web applications can be fast and stable because they control the data they get, not the server.

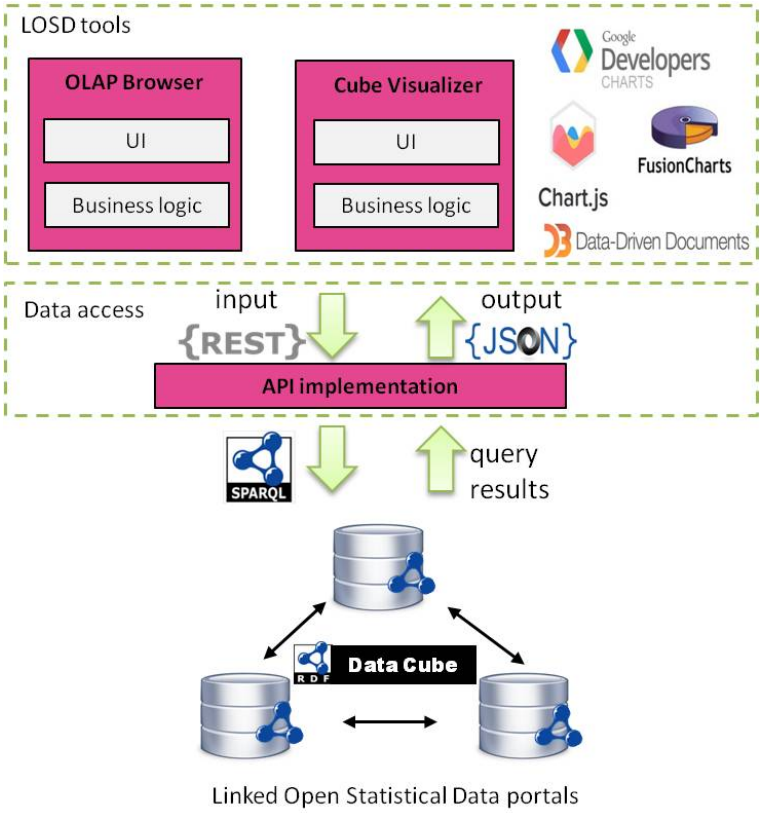

Fig. 2. JSON-QB API architecture overview

4.7 Access control

Many organisations have lots of not "fully" open data use cases. For example, ethical and legal restrictions exist to the access of health and fitness data [14]. The restrictions may derive from: i) strict regulations that protect personal data, ii) agreements that are specified in consent forms and iii) policies of stakeholders owing the data. Thus there is a need for an access control mechanism that ensures the availability of data only to authorized persons and prevent the unauthorized and unintended withholding of data.

4.8 Extensibility

Finally, the JSON-QB API should be extensible, thus take future growth into consideration minimizing the effort required for the extension. Extensions can be implemented through the: i) addition of new functionality e.g. while the initial aim is to build an API on top of RDF databases, other kinds of database could be used, ii) modification of existing functionality e.g. support modified filtering options and iii) the harmonisation with deployed solutions e.g. SDMX-REST.

5 Proof of concept implementation

The architecture of the JSON-QB API (figure 2) is simple and is developed as a middle-ware between LOSD and the applications that consume the data. The API receives the REST calls and translates them to SPARQL queries which are executed at LOSD portals. Then, the returned results are transformed to JSON format that can easily be consumed by applications.

We have developed a proof of concept implementation of the JSON-QB API[16] which can be installed on top of existing RDF repositories that store data using the QB vocabulary. Two options are currently examined for the input of the API (see sec. 4.4). The first option is to use a separate REST parameter for each input and the second to model the input as a JSON object. Results so far show that the second option seems promising since it is more flexibile and extensibile. Specifically, it enables the expression of complex search and filtering data queries limiting the transmitted data to exactly what requested. Towards this direction the implementation uses GraphQL[17], which is a data query language proposed by Facebook. Other technologies used by the API include the Jersey framework[18] for the implementation of the RESTful services, the Rdf4j[19] for processing RDF data and the Gson[20] library to serialize Java Objects into their JSON representation.

Table 2 presents an example API call that returns a cube slice by filtering the dimension values. Both options for API input are considered. The example presents also the corresponding SPARQL query and the returned JSON result.

The second option (GraphQL) is more flexible e.g. it enables the request of the title and description of the cube except from the filtered observations. However, the GraphQL approach raises some challenges that need to be addressed. For example it does not support namespaces, so all schemas exist in a single global namespace. Thus, it's hard to be sure that an extension added to the schema doesn't conflict with another extension. Another challenge is related with the conversion of URIs into fields whilst retaining uniqueness, since the characterset in GraphQL is very small. Finally, there isn't really a standardised schema serialisation for GraphQL, so you can't detect that an endpoint supports the data types you're looking for.

[16] https://github.com/OpenGovIntelligence/json-qb-api-implementation
[17] http://graphql.org/
[18] https://jersey.github.io/
[19] http://rdf4j.org/
[20] https://github.com/google/gson

Table 2. Example JSON-QB API call: get cube slice

Option 1: REST parameters	`GET /slice?dataset=http://example.com/cube/unemployment&`
	`http://example.com/dimension/refPeriod=2016&`
	`http://example.com/dimension/refArea=Greece`

Option 2: GraphQL	`{dataset_unemployment{`
	`  title`
	`  description`
	`  observations(dimensions:{refPeriod:2016 refArea:Greece}){`
	`  ageGroup`
	`  unemploymentRate`
	`}}}`

SPARQL query	`PREFIX qb: http://purl.org/linked-data/cube#`
	`PREFIX ex: http://example.com/cube/`
	`select ?obs ?ageGroup ?unemploymentRate where {`
	`?obs qb:dataSet ex:unemployment.`
	`?obs ex:refPeriod "2016".`
	`?obs ex:refArea ex:Greece.`
	`?obs ex:ageGroup ?ageGroup.`
	`?obs ex:unemploymentRate ?unemploymentRate.`
	`}`

JSON result	`{"title": "Unemployment", //only at option 2`
	`"description": "Unemployed rate in EU", //only at option 2`
	`"observations":[`
	`{"@id":"http://example.com/observation/1",`
	`"ageGroup" : "25-34",`
	`"unemploymentRate": 0.34`
	`},`
	`{"@id":"http://example.com/observation/2",`
	`"ageGroup" : "35-44",`
	`"unemploymentRate": 0.29`
	`},.... ]}`

The API is still under development and the existing version implements only a subset of the proposed functionality. A complete list of the currently implemented functionality can be found at [18].

6 Conclusion

Currently, the LOSD cloud contains many datasets and their number still increases. However, their exploitation remains low due to two reasons. First, skills and tooling for linked data are not widespread among developers and second, existing portals adopt different publishing approaches, thus hindering the development of tools that can operate across LOSD datasets.

In this paper we describe the requirements and design criteria of a JSON-QB API that standardises the interaction with LOSD aiming at their broader exploitation. Specifically, the API facilitates the development of LOSD tools

through a style of interaction familiar to web developers. It also provides uniform access to LOSD data. However, in order to achieve the uniform data access, either different implementation of the API should be created (one for each set of publishing practices) or a set of best practices should be widely adopted by publishers to provide data in a compatible way. We anticipate that the standardization of the JSON-QB API specification will contribute towards the formulation of these best practices.

The proof of concept implementation of the JSON-QB API raises many issues that need to be clarified, including: i) whether GraphQL covers the needs of the API, ii) the part of JSON-LD that will be considered (currently only limited functionality is included), iii) the relation with JSON-stat and whether it can be used as an output format of the API and iv) technical details e.g. content negotiation, status codes, success/error responses. Moreover, there are open issues related to the cube compatibility criteria and the ordering of the API results.

Acknowledgments. Part of this work was funded by the European Commission within the H2020 Programme in the context of the OpenGovIntelligence project (http://OpenGovIntelligence.eu) under grant agreement no. 693849.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
2. Capadisli, S., Auer, S., Ngonga Ngomo, A.C.: Linked sdmx data. Semantic Web 6(2), 105–112 (2015)
3. Cotton, F.: XKOS an skos extension for representing statistical classifications (unofficial draft). Tech. rep., DDI Alliance (January 2017)
4. Cyganiak, R., Reynolds, D.: The RDF data cube vocabulary: W3C recommendation. Tech. rep., W3C (January 2014)
5. Daga, E., Panziera, L., Pedrinaci, C.: A BASILar approach for building web APIs on top of SPARQL endpoints. In: Services and Applications over Linked APIs and Data SALAD2015 (ISWC 2015),vol. 1359. CEUR Workshop Proceedings (2015)
6. Do, B.L., Wetz, P., Kiesling, E., Aryan, P.R., Trinh, T.D., Tjoa, A.M.: Statspace: A unified platform for statistical data exploration. In: OTM Confederated International Conferences, Rhodes, Greece, October 24-28. pp. 792–809 (2016)
7. Groth, P., Loizou, A., Gray, A., Goble, C., Harland, L., Pettifer, S.: API-centric linked data integration: The Open PHACTS discovery platform case study. Web semantics 29(1), 12–18 (2014)
8. Hoefler, P., Granitzer, M., Veas, E.E., Seifert, C.: Linked data query wizard: A novel interface for accessing sparql endpoints. In: Workshop on Linked Data on the Web (LDOW) (2014)
9. Janssen, M., Charalabidis, Y., Zuiderwijk, A.: Benefits, adoption barriers and myths of open data and open government. Information Systems Management 29(4), 258–268 (2012), http://dx.doi.org/10.1080/10580530.2012.716740
10. Kalampokis, E., Roberts, B., Karamanou, A., Tambouris, E., Tarabanis, K.: Challenges on developing tools for exploiting linked open data cubes. In: 3rd International Workshop on Semantic Statistics (SemStats2015) co-located with ISWC2015. vol. 1551. CEUR-WS (2015)

11. Kalampokis, E., Nikolov, A., Haase, P., Cyganiak, R., Stasiewicz, A., Karamanou, A., Zotou, M., Zeginis, D., Tambouris, E., Tarabanis, K.: Exploiting linked data cubes with OpenCube toolkit. In: ISWC 2014 Posters and Demos Track, vol. 1272. CEUR-WS (2014)
12. Kalampokis, E., Tambouris, E., Tarabanis, K.: A classification scheme for open government data: towards linking decentralised data. Int. J. Web Eng. Technol. 6(3), 266–285 (Jun 2011), http://dx.doi.org/10.1504/IJWET.2011.040725
13. Kalampokis, E., Tambouris, E., Tarabanis, K.: Ict tools for creating, expanding, and exploiting statistical linked open data, statistical. IAOS 33(2), 503–514 (2017)
14. Kamateri, E., Kalampokis, E., Tambouris, E., Tarabanis, K.: The linked medical data access control framework. Journal of Biomedical Informatics 50, 213 – 225 (2014), special Issue on Informatics Methods in Medical Privacy
15. Maali, F., Shukair, G., Loutas, N.: A dynamic faceted browser for data cube statistical data. In: W3C workshop on Using Open Data (2012)
16. Martin, M., Abicht, K., Stadler, C., Ngonga Ngomo, A.C., Soru, T., Auer, S.: Cubeviz: Exploration and visualization of statistical linked data. In: Proceedings of the 24th International Conference on World Wide Web. pp. 219–222 (2015)
17. Miles, A., Bechhofer, S.: SKOS simple knowledge organization system. Tech. rep., W3C (August 2009)
18. OpenGovIntelligence: D3.2: Opengovintelligence ict tools - 1st release (2016)
19. Oracle: Oracle olap developer's guide to the olap api, 10g release 2 (10.2) (2006)
20. SDMX: Guidelines for the use of web services (version 2.1) (2013)
21. SDMX: Sdmx-json data message: syntax and documentation (2014)
22. Varga, J., Etcheverry, L., Vaisman, A.A., Romero, O., Pedersen, T.B., Thomsen, C.: Qb2olap: Enabling olap on statistical linked open data. In: 32nd International Conference on Data Engineering (ICDE). pp. 1346–1349. IEEE (May 2016)