Linked Open Statistical Data API: requirements and design criteria

xxx ddd and yyy sss

No Institute Given

Abstract.

1   Introduction

Recently, many governments, organisations and companies are opening up their data for others to reuse through *Open Data* portals [6]. These data can be exploited to create added value services, which can increase transparency, contribute to economic growth and provide social value to citizens [4].

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 compared to data cubes. Thus, we onwards refer to statistical multidimensional data as *data cubes* or just *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[3], SKOS[1] and XKOS[2] enable modelling data cubes as Linked Open Statistical Data (LOSD). In this way they facilitate data integration across portals and enable the creation of added value services and applications on top of them.

Although LOSD potential is high, their exploitation is currently low for two reasons. First, using LOSD requires skills and tooling for use of linked data (e.g. RDF, SPARQL) are less widespread than some other web technologies (e.g. JSON, REST). For example, there are many visualization libraries that consume data in JSON format (e.g D3.js, charts.js), while there are just a few that consume RDF and their functionality is limited. That's one of the reasons that there are limited application (e.g. visualisation) which exploit LOSD.

Second, many portals that use the standard vocabularies (QB, SKOX, XKOS) often adopt different publishing practices [5], thus hampering their interoperability. As a result it is not easy to create generic software tools that operate across LOSD. Usually, case specific software [**?**] are created which 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 with LOSD and hide all the complexity. Towards this end, in

---

[1] https://www.w3.org/TR/skos-reference/
[2] http://rdf-vocabulary.ddialliance.org/xkos.html

this paper we describe the requirements and design criteria of an API that standardizes the interaction (i.e. input, output and functionality) with LOSD in a way that facilitates the development of generic software. The API aims to exploit the advantages of linked data (e.g. easy data integration) but hide all the complexity. Specifically, it supports developers use linked statistical data stored in the form of an RDF Data Cube, while assuming minimal knowledge of linked data technologies. Moreover, the API offers a uniform way to access the underlying data by hiding any data discrepancies, thus enabling the development of generic software tools that operate across datasets.

The rest of the paper is organized as follows ... +++

2 Methodology

In order to achieve the objectives of the paper we adopt the following methodology:

- Study the related work. We focus on: i) APIs that facilitate the interaction with data cubes and ii) data formats that can be exploited to represent the result (i.e. output) of the API.
- Collect user requirements from developers that currently create applications for linked data cubes.

Currently, there exist many APIs that facilitate the interaction with data cubes. These APIs offer basic functionality that covers the cube's logical model, but they also support more advanced OLAP operations including aggregations, slicing, roll-up/drill-down etc. For example, the Oracle OLAP Java API [7] allows users to select, explore, aggregate, calculate, and perform other analytical tasks on data stored at Oracle data warehouse. Olap4j[3] is another Java API for accessing data cubes, which is compatible with many OLAP servers (e.g. Mondrian, Palo and SAP Business Warehouse). It enables the browsing of meta-data including the cubes, dimensions, hierarchies and members in an schema. Olap4j also supports Multidimensional Expressions (MDX) that is the query language for OLAP.

There are also some REST APIs with similar functionality. The Data Brewery[4] offers a set of Python tools, including a REST API, for processing and analysing data cubes stored at a relational data base (e.g. MySQL, PostgreSQL). Apache Lens[5] is an analytics platform that integrates Hadoop with traditional data warehouses (e.g. Apache Hive, Amazon Redshift). Lens provides a REST API to handle data cubes, that also supports "OLAP Cube QL" - a high level SQL like language to query data organized as data cubes.

All the above APIs handle data cubes that are stored at traditional databases or data-warehouse. However, none of them can handle data cubes stored as linked data using the QB vocabulary.

---

[3] http://www.olap4j.org

[4] http://databrewery.org/

[5] lens.apache.org

Regarding the output of the REST APIs, JSON is a commonly used simple format. Existing REST APIs use case-specific JSON responses, however JSON extension formats have already been proposed to model data cubes and linked data. Specifically, JSON-LD[6] offers a method for encoding linked data using JSON. While, JSON-stat[7] is a JSON format for modelling statistical data (i.e. data cubes), however the structure is too complicated when it comes to simple visualizations (e.g. maps). Moreover, it has some limitations e.g. does not support pagination of results.

Finally, to collect the user requirements, we established a continuous discussion with developers that currently create applications for linked data cubes. The discussion mainly occurs within the EU funded project OpenGovIntelligence[8], that aims to exploit linked data cubes for improving the public services. To facilitate the collection of requirements we organized a dedicated workshop at Manchester with participation of relevant developers.

3   Solution overview

A large part of statistical data is isolated, meaning that exists in different portals as different datasets. The technology of Linked data and RDF Data Cube vocabulary solves the problem of distributed data sources through their integration, creating interoperable linked statistical data portals. These portals are used for the design and creation of a Linked Open Statistical Data API.

Except from data integration, main aim of API is the implementation of data access layers.As figure 1 shows, the traditional architecture is based on monolithic, vertical applications where sofware is developed for layers. Although, the data access layers are similar in both tools, each tool uses different data access. More importantly, the development of the data access layers requires significant programming expertise in LOSD, SPARQL and RDF Data Cube vocabulary skills and knowledge of the different data publish practices. As a result, data software silos are created, being a significant barrier for the exploitation of LOSD. The architecture of the API is developed in JSON format offering benefits which are not exist until now. The implementation of API abolishes the need to implement different data access layers for each tool is created. In the traditional architecture data access layers had to be coded separately leading to additional costs. The API can be installed on top of any RDF repository and offers basic and advanced operations on RDF Data cubes.

Moreover, using JSON for accessing RDF Stores is an easier way to build software applications. Developers need to know and use neither RDF Data Cube vocabulary nor expert programming skills at LOSD. They have full access at LOSD just using JSON, a much more common and understandable programming language. Through SPARQL queries the API has access at data cubes returning

[6] https://json-ld.org
[7] https://json-stat.org/
[8] http://www.opengovintelligence.eu/

the asked data in the re-used format of JSON. By this way, all LOSD-related programming and its complexity is now hidden.

The architecture of the API is relatively simple, it is developed as a middle-ware between the Linked Open Statistical Data (stored at RDF repositories) and the applications that consume the data. In the traditional architecture, data access layers had to be coded separately leading to additional costs. However, the implementation of the API abolishes the need to implement different data access layers for each application.

In addition, there is a need to standardise the API specification and try to get broad support for it, so that many data publishers can provide data in a compatible way, making tools interoperable. As well as making data easier to consume, using the API as the main method of delivering machine readable extracts of data would remove or greatly reduce the need for data publishers to provide public SPARQL endpoints. By this way cost can be reduced and reliability of open services could be improved.
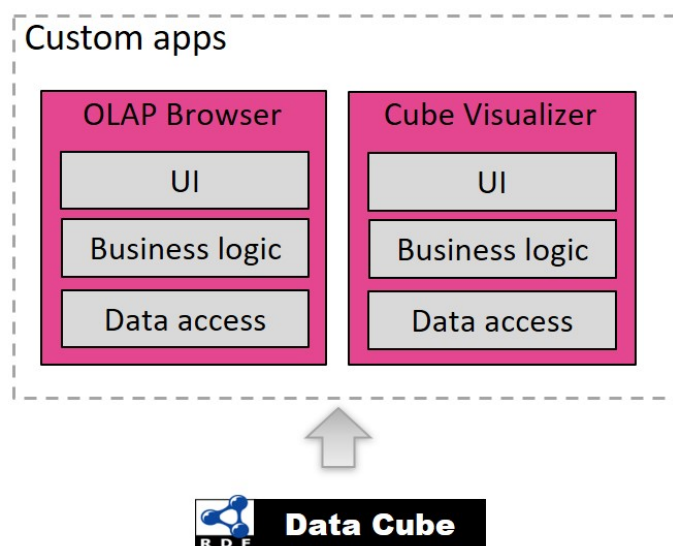

Fig. 1. AS-IS overview

The API offers a uniform way to handle data cubes. It receives the API calls and translates them to SPARQL queries that are executed at Linked Open Statistical Data. Then, the API gets the result, expressed using the RDF Data cube vocabulary, and transforms it to a JSON representation that can easily be consumed by applications.
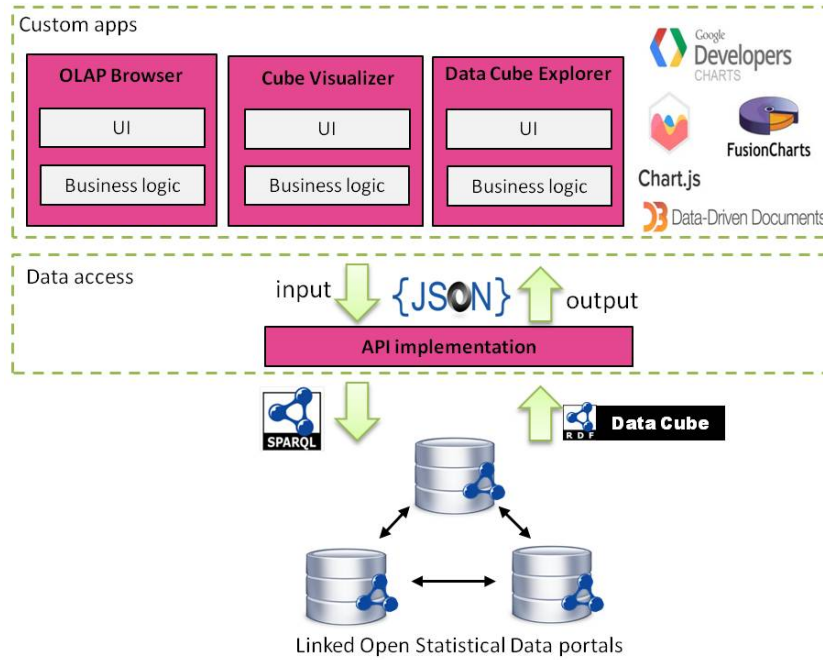
Fig. 2. Solution overview

4 Requirements and design criteria

This section presents the functional and non-functional requirements and design criteria related to the Linked Open Statistical Data API. A summary of the requirements is presented at table 1.

4.1 Search data cubes

The linked data web currently contains many published 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 can be even more complex e.g. get cubes about unemployment in Greece after 2010. To fully exploit the cube searching functionality, the API ideally should search over the linked data web and not be limited to a single RDF store.

The search functionality can also be extended to support not only user specific criteria (as the previous examples), 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 of for browsing. The cube compatibility criteria is still an open issue and is out of the scope of this paper.

Table 1. Summary of API requirements

Type	Name	Description
functional	Search data cubes	Search for data cubes at linked statistical data portals based on specified criteria
	Explore data cube structure	Having a cube at hand, get information about its structure (e.g. dimensions, measures, attributes)
	Slicing and filtering	Get a sub-set (slice) of cube observations based on user-defined criteria
non-functional	Easy of use	Provide a style of interaction that is familiar to web developers
	Uniform data access	Hide any discrepancies of published data cubes
	High performance	Provide responses fast even for demanding requests
	Extensibility	Enable the addition of new or the modification of existing functionality

4.2 Explore data cube structure

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 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 API should allow applications to take exactly the data they want by defining constrains (i.e. filters) to the dimension values. The API should support many filtering options 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 API should support paging and ordering of the results. The ordering of the results can be in ascending or descending order based on one or more dimensions. However, in some cases lexicographical ordering is not appropriate (e.g. for the days of the week), thus other types of ordering should be applied.

4.4 Easy 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 creating many obstacles at their adoption. The purposes of the 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.

The easy of use of an API is related both to the input of the API (API calls) and the out put. Regarding the input of the API there are mainly two design options: i) use a separate parameter for each required input and ii) model all the required input as a JSON object. The first option is post popular at existing APIs, while the second is more flexible and expressive. For example, it enables the expression of relations other than equality e.g. greater than (see Table 2).

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. An example of a JSON code that could represents an array of observations is presented below.

```
{ "observations": [
    {"gender"       : "male",
```

Table 2. Input of API: separate parameters vs JSON object

Separate parameters	Input as JSON

GET /slice?dataset=home-care&
         gender=male
	{"dataset": "home-care",
	 "filter": {
	    "gender": "male"
	}}

Cannot be expressed	{"dataset": "home-care",
	 "filter": {
	    "gender": "male"
	    "age": { "greater-than": 50}
	}}

```
    "age"          : "55-65",
    "refPeriod"    : "2016",
    "refArea"      : "Scotland",
    "unemployment" : "4.3",
    "unit"         : "percent"},
  {"gender"       : "female",
    "age"          : "55-65",
    "refPeriod"    : "2016",
    "refArea"      : "Scotland",
    "unemployment" : "4.7",
    "unit"         : "percent"},
    ...
]}
```

Finally, JSON-LD representation can be used both at the input and output of the API format in order to represent linked data.

4.5 Uniform data access

Currently many linked data cubes have been published through official data portals, however a lot of them adopt different publishing practices. As a result, only case specific applications can be created that assume data published in a specific way. The API should work on top of any of these data, offering uniform access to the data and hide any discrepancies. Obviously, this will require different API implementations to comply with the different publishing practices. However, using the API as the main way to deliver data, will reduce the need for data publishers to maintain public SPARQL endpoints.

Ideally, the standardization of the API specification will also lead 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 linked data cube applications.

4.6 High performance

The volume of data published as linked data cubes 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. As a result, applications that use such queries seem to be non-responsive.

The 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.

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 need to compute the aggregations on-the-fly when requested.

4.7 Extensibility

Finally, a requirement of the API is to be extensible, thus take future growth into consideration minimizing the effort required for the extension. Extensions can be implemented through the :

- 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, enabling flexibility and innovation
- modification of existing functionality e.g. support modified search (see section 4.1) of filtering options (see section 4.3).

5 Implementation

The API implementation[9] aims to provide an easy to use API for web developers that use statistical data stored in the form of RDF Data cubes. It can be installed on top of any RDF repository and offer basic and advanced operations on RDF Data cubes assuming that: i) they are stored using the RDF Data Cube Vocabulary, ii) they follow a specific application profile (common practices) and iii) are accessible through a SPARQL endpoint. All the functionality is offered through simple REST calls. The API translates each call to a SPARQL query that is executed at the linked data cubes. The returned results are always transformed in JSON format

The implementation of the API uses the following technologies:

---

[9] https://github.com/OpenGovIntelligence/json-qb-api-implementation

- Jersey framework for the implementation of the RESTful services. Jersey is an implementation of the JAX-RS Reference Implementation.
- SPARQL for querying the RDF repositories.
- Rdf4j Java framework for processing RDF data
- Gson Java library to serialize Java Objects into their JSON representation

Table 3. Example API calls

Dimensions	
API call	`GET /dimensions?dataset=http://example.com/cube/unemployment`

SPARQL query	`PREFIX qb: http://purl.org/linked-data/cube#`
	`PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#`
	`PREFIX ex: http://example.com/cube/`
	`select distinct ?dim ?label where {`
	`    ex:unemployment qb:structure ?dsd.`
	`    ?dsd qb:component  ?cs.`
	`    ?cs qb:dimension ?dim.`
	`    ?dim rdfs:label ?label.}`

JSON result	`{"dimensions":[`
	`  {"@id":  "http://example.com/dimension/refArea",`
	`   "label":"Reference area"},`
	`  {"@id":  "http://example.com/dimension/refPeriod",`
	`   "label":"Reference period"},`
	`  ....`
	` ]`
	`}`

Slice	
API call	`GET /slice`
SPARQL query	`XXX`
JSON result	`YYY`

++ easy app creation++ image cube visualiser!

6 Conclusion

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. Cyganiak, R., Reynolds, D.: The RDF data cube vocabulary: W3C recommendation. Tech. rep., W3C (January 2014)
4. 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

5. Kalampokis, E., Roberts, B., Karamanou, A., Tambouris, E., Tarabanis, K.: Challenges on developing tools for exploiting linked open data cubes. In: Proceedings of the 3rd International Workshop on Semantic Statistics (SemStats2015) within the 14th International Semantic Web Conference (ISWC2015). vol. 1551. CEUR-WS (2015)
6. 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
7. Oracle: Oracle olap developer's guide to the olap api, 10g release 2 (10.2) (2006)