International Journal of Geo-Information

MDPI

Article

Mapping Spatiotemporal Data to RDF: A SPARQL Endpoint for Brussels

Alejandro Vaisman [1,*] and Kevin Chentout [2]

[1] Instituto Tecnológico de Buenos Aires, Buenos Aires 1424, Argentina
[2] Sopra Banking Software, Avenue de Tevuren 226, B-1150 Brussels, Belgium
* Correspondence: avaisman@itba.edu.ar; Tel.: +54-11-3457-4864

Received: 20 June 2019; Accepted: 7 August 2019; Published: 10 August 2019

Abstract: This paper describes how a platform for publishing and querying linked open data for the Brussels Capital region in Belgium is built. Data are provided as relational tables or XML documents and are mapped into the RDF data model using R2RML, a standard language that allows defining customized mappings from relational databases to RDF datasets. In this work, data are spatiotemporal in nature; therefore, R2RML must be adapted to allow producing spatiotemporal Linked Open Data.Data generated in this way are used to populate a SPARQL endpoint, where queries are submitted and the result can be displayed on a map. This endpoint is implemented using Strabon, a spatiotemporal RDF triple store built by extending the RDF store Sesame. The first part of the paper describes how R2RML is adapted to allow producing spatial RDF data and to support XML data sources. These techniques are then used to map data about cultural events and public transport in Brussels into RDF. Spatial data are stored in the form of stRDF triples, the format required by Strabon. In addition, the endpoint is enriched with external data obtained from the Linked Open Data Cloud, from sites like DBpedia, Geonames, and LinkedGeoData, to provide context for analysis. The second part of the paper shows, through a comprehensive set of the spatial extension to SPARQL (stSPARQL) queries, how the endpoint can be exploited.

Keywords: GIS; RDF; Semantic Web; SPARQL; Strabon

1. Introduction

The Semantic Web (a term coined by Tim Berners-Lee) aims at providing a common framework in order to allow data to be shared and reused across applications and to be consumed by machines rather than human beings. The idea behind this is to transform the current "Web of Documents" (the "Syntactic Web") into a "Web of Data" (in the traditional database sense). This requires a technology stack, denoted the "Semantic Web Stack" [1], that enables people to create data stores on the web, build vocabularies, and write rules for handling data. To this end, the World Wide Web Consortium (W3C) (http://www.w3.org/) develops and defines standards with the vision of building a Web of Linked Data. Linked Data (http://www.linkeddata.org) refers to a set of best practices for publishing and interlinking structured data (denoted as resources) on the web, where resources are represented and described using the Resource Description Framework (RDF) [2] (see below). These best practices are known as the Linked Data principles and consist of: (a) using Internationalized Resource Identifiers (IRIs) as names for things; (b) using HTTP IRIs, to make it easy to look up the names in (a); (c) providing useful information in the IRIs, using standards (e.g., RDF for modeling, SPARQL for querying); and (d) including links to other IRIs, to discover further resources. Linked Data are empowered by technologies such as RDF and SPARQL, among others. RDF, the data model for the Semantic Web expresses assertions over resources identified by an IRI. These assertions have

the form of subject-predicate-object triples, where the subject is an IRI-identified resource or a blank node, the predicate is an IRI-identified resource, and the object could be a resource or a string. A blank node is a special kind of node representing an anonymous resource, typically with a structural function. Data values in RDF are called literals. Many formats for RDF serialization exist. This paper adopts Turtle [3] and assumes that the reader is familiar with this notation. SPARQL 1.1 [4] is the W3C standard language for querying RDF data. SPARQL stands for SPARQL Protocol and RDF Query Language. The query evaluation mechanism of SPARQL is based on subgraph matching: RDF triples are interpreted as nodes and edges of directed graphs and the query graph is matched to the data graph, instantiating the variables in the query graph definition. The selection criteria are expressed as a graph pattern in the `WHERE` clause, consisting basically of a set of triple patterns connected by the "." operator. Further, SPARQL 1.1. supports aggregate functions and the `GROUP BY` clause, essential for analytical queries.

In addition to the above, the Open Knowledge Foundation (https://okfn.org/) defines "Open Data" as data that can be freely used, shared, and built-on by anyone, anywhere, for any purpose. Putting together the concepts of Linked Data and Open Data yields the notion of "Linked Open Data" (LOD). Publicly-available linked datasets are depicted in the Linked Open Data Cloud (https://lod-cloud.net/), which is updated and maintained by the Insight Center for Data Analytics (https://www.insight-centre.org/). Crucial issues that arise in an LOD scenario are data acquisition, integration, and exploitation. Most data on the web are obtained from relational databases, although there are other kinds of data sources from which data could be acquired. Several mechanisms exist to produce RDF data from a wide variety of data sources. These mechanisms are studied in this paper. Data on the web can be made available in many different formats and can be accessed in different ways. For example, data represented as XML or JSON documents can be obtained through specialized RESTful APIs, while data represented and stored using the RDF data model can also be published and accessed through SPARQL endpoints (in a nutshell, services that accept SPARQL queries and return results). RDF data could also be extracted from RDF-embedded HTML pages (called RDFa) (http://www.w3.org/TR/xhtml-rdfa-primer/). Further, the Semantic Web is increasingly being populated with geospatial data [5–8], particularly in the fields of Earth and Environmental science. Probably the main example in this sense is LinkedGeoData [9], which exposes data from OpenStreetMap (https://www.openstreetmap.org) in RDF format, to be queried using SPARQL. Furthermore, in the United Kingdom, the Ordnance Survey (https://www.ordnancesurvey.co.uk/) (the British mapping agency) is publishing geospatial datasets. The research community has also produced relevant work in the field [10–13].

In light of the above, this paper addresses the problem of capturing (possibly spatial) data from different sources, e.g., relational databases, XML documents, web APIs, SPARQL endpoints, producing (spatial) RDF data, and integrating and exposing such data in a spatially-enabled SPARQL endpoint. Many challenges appear in this setting and are studied throughout the paper. For this, a real-world case study is used. This case study refers to an LOD platform for the region of Brussels, in Belgium. The goal of this platform is two-fold: On the one hand, data providers are encouraged to publish their data, since the semi-automatic mapping tools described in this paper allow doing this with minimal effort and cost. On the other hand, users can either query the data over the web or applications can consume these data, allowing developers to provide added value services. This case study is part of a larger project funded by the Brussels Capital Region and aims at developing a prototype for integrating data from cultural events in Brussels, provided by two partners, Agenda.be (https://agenda.brussels/en) and Bozar https://www.bozar.be/), with the public transport schedule (provided by the main public transport company in the city, the STIB (http://www.stib.be/)). The integrated data are offered to the public over a SPARQL endpoint and can be consumed for direct analysis or by applications. One use case in this project (whose details are outside the scope of this paper) consists of developing an application that can, for instance, take a picture of a place and, after generating the appropriate SPARQL query, obtain the events taking place there within the next two hours and inform about how to reach them

from the user's current location using public transport. The problem addressed here tackles only the data infrastructure to support applications of this kind. The paper provides an in-depth discussion and description of the deployment and querying of the spatial SPARQL endpoint, backed by the Strabon triple store, a geospatial database management system (DBMS) that supports the stRDF representation language (a spatiotemporal extension to RDF) and comes equipped with a spatial extension to SPARQL, called stSPARQL. The case study introduced above involves several challenges and allows interesting querying possibilities that are explored and discussed in this paper. First, the cultural and public transport data used in the project come in the form of relational tables and XML documents. However, only data from STIB include spatial features like coordinates of places. Therefore, spatial data from the other providers must be produced from alphanumeric data (e.g., addresses). Second, to be openly accessed and linked, data need to be mapped into RDF triples, which in the solution proposed here, is performed using the R2RMLmapping language. (http://www.w3.org/TR/r2rml/) Since R2RML neither directly supports XML data sources, nor spatial data; thus, the R2RML mapping language must be extended in order to tackle both issues, and this is discussed in the paper. Further, to provide context for analysis, data are captured from external sources (DBpedia, Geonames, and LinkedOpenData) and put into stRDF format. Finally, a large part of the paper is devoted to discussing a comprehensive set of stSPARQL queries, which show how the endpoint can be exploited for analysis. This set of queries is classified into different subsets of different characteristics, for example queries requiring aggregation, spatial buffering, and so on.

The remainder of the paper is organized as follows: Section 2 introduces Strabon and discusses related work on geospatial RDF data stores. Section 3 describes the R2RML mapping language and explains how it is extended to support spatial data present in XML and relational data sources. It also shows how this technique is applied to data provided by the partners in the project. In addition, related work on mapping data to RDF is also briefly discussed. Section 4 shows how data from external sources (DBpedia, Geonames, and LinkedGeoData) were added to the triple store to provide context for querying and for dataset enrichment. Section 5 provides extensive example queries over the endpoint. Section 6 concludes the paper.

Note: A preliminary version of this paper has been published as two extended abstracts [14,15], briefly describing the project's features. The present paper substantially expands and updates such work, providing a full, detailed explanation of each step of the methodology and a comprehensive set of queries that shows the functionality and usefulness of the case study.

2. Geospatial Triple Stores

The availability of geospatial data on the Semantic Web has triggered the interest of the research community on developing spatial (and even spatiotemporal) extensions to SPARQL, which resulted in the GeoSPARQL standard. Other proposals based on this standard exist. One of them is Strabon (and its accompanying language, denoted stSPARQL), developed almost in parallel with GeoSPARQL. Strabon is also the system used for the work presented in this paper. This section discusses the reason for such a decision and briefly comments on the characteristics of other representative geospatial triple stores.

GeoSPARQL

GeoSPARQL (https://www.opengeospatial.org/standards/geosparql) is the Open Geospatial Consortium's (OGC) (http://www.opengeospatial.org/) standard extension to SPARQL. GeoSPARQL supports the representation and querying of geospatial data on the Semantic Web, defining a vocabulary for representing geospatial data in RDF and an extension to SPARQL. GeoSPARQL and Strabon (discussed below) were developed independently at around the same time, resulting in very similar representational and querying constructs. GeoSPARQL represents geometries as literals of a certain data type. These literals may be encoded in various formats like GML, well-known text (WKT), and so on. Furthermore, like Strabon, GeoSPARQL maps spatial predicates and functions

that support spatial analysis to SPARQL extension functions, although GeoSPARQL allows binary topological relations to be used as RDF properties. Note however that GeoSPARQL does not provide aggregate functions, a crucial feature for data analysis, which is the main goal of the work in this paper, as can be seen in Section 5. On the other hand, Strabon was developed based on SPARQL 1.1., which includes (opposite to previous SPARQL versions) SQL-like aggregate functions.

The Strabon Triple Store

Strabon (http://www.strabon.di.uoa.gr/) [16–19] is an RDF data store with spatiotemporal support. That means Strabon can be used to store and query linked geospatial data that change across time. Queries in Strabon can be expressed using two well-known SPARQL extensions: stSPARQL and a subset of GeoSPARQL. The former can be used to query data represented in an extension of RDF called stRDF. Both stRDF and stSPARQL are designed to represent and query geospatial temporal data (e.g., like the reduction of forests in a region over the years due to uncontrolled exploitation). On the other hand, Strabon also supports querying geospatial RDF data using a subset of GeoSPARQL. At the time of writing this paper, this includes the core, geometry extension, and geometry topology extension. However, it is remarked again that stSPARQL provides geospatial aggregate functions, while GeoSPARQL does not. This feature is crucial for data analysis and one of the main reasons for choosing stSPARQL for this project. Further, the temporal dimension is also not addressed by GeoSPARQL. (https://event.cwi.nl/eswc2015-geo/03-stsparql-geosparql.pdf)

Strabon supports spatial data types, allowing representing geometric objects by means of the OGC standards WKT [20] and GML [21]. This is achieved through the definition in stRDF of the `strdf:WKT` and `strdf:GML` data types. WKT stands for well-known text, a text markup language for representing vector geometry objects on a map. GML is the XML grammar defined by the OGC to express geographical features.

Strabon provides spatial and temporal selections and joins and a collection of spatial functions like the ones included in geospatial relational database systems, also supporting many different coordinate reference systems. Further, since Strabon originally extended the RDF store Sesame (currently known as RDF4J) (http://rdf4j.org/), it can handle alphanumerical and spatial RDF data stored in a PostGIS [22] backend.

The stSPARQL query language extends SPARQL with spatial functions. These functions can be used in three parts of a SPARQL query: the `SELECT`, `FILTER`, and `HAVING` clauses. Arguments of these functions are spatial terms, which can be of three kinds: (a) a spatial type literal with data type `strdf:geometry` or its subtypes; (b) a query variable that can be bound to a spatial literal; (c) the result of a set operation on spatial literals (e.g., intersection, union, etc.); (d) The result of a geometric operation on spatial terms (e.g., buffer). Furthermore, a Boolean SPARQL extension function for each topological relation defined in the OGC-SFA(topological relations for simple features) [23] is supported. An example of a spatial join in a `FILTER` clause using two variables as arguments is the expression: `strdf:contains(?geoA, ?geoB)`. An example of a spatial join, but in this case used in the `SELECT` clause of a query, can read:

```
SELECT(strdf:buffer(?river, 0.04) AS ?buffer).
```

The expression above allows drawing a buffer around a geometry representing a river using a variable called `?river`.

Finally, aggregate functions that deal with geospatial data are also supported as follows:

- `strdf:union(set of strdf:geometry a)` returns a geometry that is the union of the set of input geometries.
- `strdf:intersection(set of strdf:geometry a)` returns a geometry that is the intersection of the set of input geometries.

- `strdf:extent(set of strdf:geometry a)` returns a geometry that is the minimum bounding box of the set of input geometries

In addition to the above, insertion, deletion, and update of stRDF triples are supported by stSPARQL. This is however outside the scope of this paper.

Other Geospatial RDF Stores

In addition to Strabon, there are other RDF geospatial RDF data stores. The oldest ones are Parliament [24] and uSeekM [25]. The former supports a large portion of GeoSPARQL. Like Strabon, the WKT and GML serializations are supported. OpenSahara's uSeekM is based on the RDF store Sesame, storing and querying spatial data using PostGIS. Although most GeoSPARQL functionalities are supported, it does not use IRIs for CRS (coordinate reference systems); therefore, it does not satisfy the project's requirements.

On the NoSQL-Big Data side, RDF4J (https://projects.eclipse.org/projects/technology.rdf4j) is a Java-based RDF framework for Linked Data. From the Version 2.4.3, released in late 2018, RDF4J provides geospatial functionality and GeoSPARQL support. Anyway, in spite of not being mature enough, at the time of starting the project described here, this system was not available, and thus, it was not considered as a candidate for the geospatial RDF data backend for the project. The same occurred with Ontotext's GraphDB (http://graphdb.ontotext.com/documentation/free/) v8.6.1 (previously OWLIM), a NoSQL semantic graph database with geospatial support (although it just supports WKT geometry serialization and WGS84 CRS).

Finally, there are other semantically-enabled systems with limited geospatial capabilities, which, for this reason, were not considered as candidates for the project. Examples of these systems are OpenLink Virtuoso (https://virtuoso.openlinksw.com//) and Allegro Graph (https://franz.com/agraph/allegrograph/).

From the analysis of the systems mentioned above, and taking into account functionality, standard support, and its open source condition, Strabon was adopted as the spatial RDF triple store for the work described in the sequel.

3. Extending R2RML Mapping for Spatial Data and XML Support

Since most Semantic Web data come from relational databases, several proposals exist, aimed at producing RDF data from relational data stored in different repositories. Two problems arise here. On the one hand, data have to be translated (mapped) from the relational format to the RDF data model. On the other hand, one can choose between storing RDF data in a data store or producing RDF triples on-the-fly. Several proposals and standards have been developed for this. The most relevant of them are discussed below. Section 3.1 studies the R2RML mapping language, the standard language chosen for this project, while Section 3.2 comments on other proposals and compares them against R2RML. Sections 3.3 and 3.4 present the XML and spatial extensions to R2RML developed for the project.

3.1. The R2RML Mapping Language

The W3C standard for mapping relational to RDF data, denoted R2RML [26], is a customized mapping whose result is a collection of RDF triples in Turtle syntax that represents all or a portion of a relational database. The main object of an R2RML mapping is the "triples map". Each triples' map yields a collection of triples, and it is composed of a "logical table", a "subject map", and zero or more "predicate object maps". A logical table can be either a table name (defined using predicate `rr:tableName`) or an SQL query (defined using predicate `rr:sqlQuery`). A predicate object map is composed of a predicate map and an object map. Subject maps, predicate maps, and predicate object maps can be constants (using predicate `rr:constant`), column-based maps (using predicate `rr:column`), or template-based maps (predicate `rr:template`).

The example in Listing 1 depicts the mapping into RDF of a table called Professor. This table contains attributes id, name, department, and gender. For this mapping, the R2RML language is used. The triples representing a given instance of this table are produced applying the mapping shown in the figure. For example, applying the mapping to the tuple <1,Professor, A. Vaisman, Computer Science, M>, produces the triples (prefixes are omitted for brevity):

```
<http://myexample.org/professor/1> rdf:type prof:Professor.
<http://myexample.org/professor/1> prof:name "A. Vaisman".
<http://myexample.org/professor/1> prof:worksIn "Computer Science".
<http://myexample.org/professor/1> prof:gender "M".
<http://myexample.org/professor/2> rdf:type prof:Professor.
```

Listing 1: Mapping a Table using R2RML

```
<#TriplesMap>
rr:logicalTable [
rr:tableName "Professor"
];

rr:subjectMap [
rr:template "http://myexample.org/professor/{ID}";
rr:class prof:Professor;
];

rr:predicateObjectMap [
rr:predicate prof:name;
rr:objectMap [
rr:column "Name"
];
];
rr:predicateObjectMap [
rr:predicate prof:worksIn;
rr:objectMap [
rr:column "Department"
];
];
rr:predicateObjectMap [
rr:predicate prof:gender;
rr:objectMap [
rr:column "Gender"
];
].
```

In summary, given a relational database, to translate it into RDF triples, a mapping file containing the R2RML specification is created. This mapping is then applied to a given instance of the relational database.

In this paper, the mappings are generated using a common vocabulary and an ontology for the cultural domain, defined using the GOSPLtool [27]. GOSPL is a collaborative ontology evolution methodology (it has also an associated tool), which supports stakeholders in interpreting and modeling their common ontologies in their own terminology and context, also feeding back results to the owning community. In this case, the ontology was created in a collaborative way by all the project's participants, in an evolving manner. To illustrate the need for this ontology, consider for example that a key element in the cultural domain, namely an event, is denoted Event in the Agenda.be data and Activity in the Bozar database. Further, the cultural ontology is extended to support elements not particularly belonging to the cultural domain (defined mainly by the partners involved in these issues). This way, for instance, references to places and locations of an event must be included in the ontology. The same occurs with dates, event categories, and others. Table 1 shows some examples.

Finally, a mapping engine takes the database instance and the mapping file to produce the RDF document. Given that the standard specification of R2RML does not support spatial data, in order to satisfy the requirements of the problem at hand, the scheme commented on above was extended

as explained in Section 3.4. For completeness and to put this decision in context, other mapping options are discussed in the next section.

Table 1. Mapping of elements in the Bozar database to concepts in the cultural ontology.

Column	Concept
activity.date_end	gospl:DateTimeSpecification_valid_until_Date
activity.date_start	gospl:DateTimeSpecification_valid_from_Date
activity_category	gospl:Category
activity_lng.field.title	gospl:Event_has_Title
personality	foaf:Person
opus	gospl:Opus
...	...

3.2. Other Relational to RDF Mapping Tools and Languages

R2RML is not the only language for mapping relational to RDF data. Therefore, other options are discussed next, to understand the rationale for using R2RML in this project.

RML

The RDF Mapping Language (RML) [28,29] has been proposed to override R2RML's limitation on the kind of supported data sources. For this, RML is designed as a superset of R2RML whose main feature consists of a vocabulary that defines a generic data source instead of the three kinds allowed in R2RML. RML extends the notion of the logical table, defined in R2RML, and introduces the notion of the logical source, with the property `rml:logicalsource`, which can be any data input. It also defines the properties `rml:source`, `rml:reference`, and `rml:referenceFormulation`, instead of R2RML's `rr:tableName`, `rr:column`, and `rr:SQLQuery`, respectively. It also defines an iterator, denoted `rml:iterator`. However, opposite R2RML, RML is not a standard recommendation; thus, for this work, R2RML was chosen.

Direct Mapping

Direct mapping (DM) [30] was the first W3C standard for mapping relational to RDF data sources. It is the simplest mapping method for such purposes. A direct mapping takes as input a relational schema and an instance of such a relational schema and produces an RDF graph as the output. Each table defines a class, using the `rdf:type` predicate. Each row in a table produces a set of triples with a common subject, which is an IRI formed from the concatenation of the base IRI, the table name, the primary key column name, and a primary key value. To indicate a foreign key, a reference triple is produced with a predicate formed by the concatenation of the table name, a "ref" string, and the referencing column name. As an example, the triples in Listing 2 were produced by mapping two tables, namely `Laboratories` and `Researchers`.

DM is very easy to learn and use and produces as the output an RDF graph that reflects exactly the same structure as the one of the source database. However, compared to R2RML, the main drawback of DM is that it does not provide a mapping vocabulary; therefore, it does not allow using existing vocabularies to describe the data to be translated, since the name of all the predicates are the column names of the relational tables. Therefore, it would not be possible to take advantage of existing ontologies, which rules out DM as a candidate to be used in the work presented in this paper.

Listing 2: An example of a Direct Mapping

```
@base <http://ulb.ac.be/db/> .
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
...

<Laboratories/LID -1> rdf:type <Laboratories > ;
<Laboratories#LID > 1 ;
<Laboratories#Lname > "Web and Information Technologies" ;
<Laboratories#Llocation> "Building A.5";
<Laboratories#Head_ID > 5 ;
<Laboratories#ref -Head_ID <RESEARCHERS/RID -5> .
##...
<Researchers/RID -5> rdf:type <Researchers > ;
<Researchers#RID > 5;
<Researchers#Rname > "Jack" ;
<Researchers#Lab_ID > 1 ;
<Researchers#ref-Lab_ID > <Laboratories/LID -1>.
```

The D2RQ Platform

The D2RQPlatform (http://d2rq.org/) is a system for accessing relational databases as virtual, read-only RDF graphs. It offers RDF-based access to the content of relational databases without having to replicate it into an RDF store. D2RQ allows querying a non-RDF database using SPARQL, accessing the content of the database as Linked Data over the Web, and creating custom RDF dumps. The platform consists of three parts: (a) the D2RQ Mapping Language, with similar goals as R2RML; (b) the D2RQ Engine, which uses the mappings to rewrite Jena API calls as SQL queries against the database, passing the query results up to the outer layers of the frameworks; (c) the D2R Server, which provides, among others, the SPARQL endpoint. Requests from the web are translated on-the-fly into SQL queries using the D2RQ mapping. Therefore, it is possible to query the sources in SPARQL without using an RDF triple store. Furthermore, as mentioned above, an RDF dump could be produced. From the above, it follows that R2RML and D2RQ could work together in both ways, on-the-fly, and as an RDF dump of the triples produced by the mapping. Regarding the project described in the present work, it was considered that the D2RQ server for spatiotemporal data support would, at least, be a very hard task. Further, the solution adopted here makes it easier to support XML data sources.

3.3. Extending R2RML Sources for XML Support

As mentioned above, R2RML was originally designed to receive relational data as input, through three mechanisms: a table name, an SQL query, and an R2RML view. Therefore, a different mechanism should be used to cope with XML documents. The trivial solution would be to transform the XML document into a table. This solution may work for simple XML documents. However, when this is not the case (like in the Agenda.be data used in this project), this approach would not suffice. Therefore, the approach discussed here enriches the R2RML specification with special terms that account for XML documents. Although this solution actually modifies the standard in some sense, the modification proposed here is just a vocabulary extension, which does not prevent any standard mapping from being defined. The idea is to allow the user to provide as input a list of XPath expressions, such that the first element in this list gives the context node for all the expressions that follow. To extend R2RML, a new source `rr:xpathQuery` was defined in the R2RML namespace used in the `rr:logicalTable` blank node, and taking a string literal as associated object. This string contains a set of valid XPath expressions separated by the "~" character (chosen since it is not allowed in XPath expressions). It is worth remarking that, although the new predicate belongs to the R2RML namespace, a local namespace could have been used as well (e.g., http://ulb.ac.be/ontology/), to remain independent of possible future R2RML updates. The new predicate looks as follows:

```
rr:xpathQuery """//document~
  ./professor~
  ./name~
  ./worksIn""";
```

As an example, consider the data provided for this project by Agenda.be, a service that informs about cultural and artistic events in Belgium's capital city and the institutions where such events take place. Data were provided by means of two XML documents: events.xml, containing the events and their description, and institutions.xml, containing the institutions mentioned above. Both documents were related to each other in a way such that the place and organizer appearing in events.xml correspond to an institution appearing in institutions.xml. For example, consider the event with ID 234217 in events.xml, corresponding to the presentation by mezzo-soprano Guillemette Laurens, indicated in Listing 3. The event is organized by an institution with ID 233287.

Listing 3: An event in the events.xml file.

```
<Event_Instance>
<EventID>234217</EventID>
<Event_Title_FR>Guillemette Laurens (mezzo-soprano), Fuoco e cenere.</Event_Title_FR>
...
<Event_Place>
<Event_Place_InstitutionID>471</Event_Place_InstitutionID>
<Event_Place_Name_FR>Conservatoire Royal de Bruxelles</Event_Place_Name_FR>
...
</Event_Place>
<Event_Organisator>
<Event_Organisator_InstitutionID>233287</Event_Organisator_InstitutionID>
<Event_Organisator_Name_FR>Opus 3</Event_Organisator_Name_FR>
...
</Event_Organisator>
...
</Event_Instance>
```

Furthermore, consider the R2RML mapping file in Listing 4, which indicates how the triples are generated from the XML file.

Listing 4: A portion of the R2RML mapping file for the events.xml file.

```
...
rr:logicalTable [
rr:xpathQuery """//Event_Instance~
./EventID~
./Event_Place/Event_Place_InstitutionID
""";
];
...
...
rr:subjectMap [
rr:template "http://www.agenda.be/db/Event/{EventID}";
rr:class gospl:Event; ];

rr:predicateObjectMap [
rr:predicate gospl:Event_organized_by_Institution;
rr:objectMap [
rr:template "http://www.agenda.be/db/Institution/{Event_Organisator_InstitutionID}";
rr:termType rr:IRI;];
];

rr:predicateObjectMap [
rr:predicate gospl:Event_taking_place_at_Institution;
rr:objectMap [
rr:template "http://www.agenda.be/db/Institution/{Event_Place_InstitutionID}";
rr:termType rr:IRI;];
];
```

The mapping produces the RDF triples in Listing 5, which inform that the event is organized by an institution with ID 233287, which is called Opus 3, and takes place at the Conservatoire Royal

de Bruxelles, which has ID = 471. This semantics that is used in the translation is given by the ontology created using the GOSPL tool.

Listing 5: Triples produced by the mapping in Listing 4.

```
1
2 <http://www.agenda.be/db/Event/234217>
3 <http://starpc18.vub.ac.be/gospl/ontology/2#Event_organized_by_Institution>
4 <http://www.agenda.be/db/Institution/233287>;
5
6 <http://starpc18.vub.ac.be/gospl/ontology/2#Event_taking_place_at_Institution>
7 <http://www.agenda.be/db/Institution/471>.
```

Data in the XML files also contain spatial information. The mapping of spatial data in XML documents is addressed in the next section.

3.4. Mapping Spatial Data

This section describes the mapping of spatial data into RDF aimed at providing LOD for Brussels. Although there are proposals to map geospatial data to RDF automatically [12,13], based on R2RML and RML, in the remainder, it will become clear that the problem is too complex to be solved just using automated tools. Further, as explained above, RML is not a standard, and a design decision was to work with standard tools.

Three datasets were used for this purpose, available from three Belgian companies: Agenda.be, Bozar, and STIB. Different places of interest (PoI) were included in these datasets. In order to be able to geolocate these PoIs, longitude and latitude were needed. Therefore, a conversion from full addresses of the PoIs (containing street, number, city, and postal code) into geographic coordinates was needed, a process known as geocoding. Many open-source geocoding tools are available, for example MapQuest Open (https://developer.mapquest.com/documentation/open/nominatim-search/), OpenCage Geocoder https://opencagedata.com/), LocationIQ (https://locationiq.com/), and OpenStreetMap's Nominatim (https://nominatim.openstreetmap.org/).

Section 3.3 showed how XML data in the events.xml file provided by Agenda.be were mapped into RDF. However, the other XML document, namely institutions.xml, contained also spatial data (the institutions' addresses). Mapping this spatial information coming from an XML document requires a previous step, explained next.

Institutions and their addresses in the source XML file are represented as shown in Listing 6.

Listing 6: An institution in the institutions.xml file.

```
1 <InstitutionID>232502</InstitutionID>
2 <Institution_Name_FR>Maison de la Poesie</Institution_Name_FR>
3 <Institution_Name_NL>Maison de la Poesie</Institution_Name_NL>
4 <Institution_Street_FR>r. Fumal 28</Institution_Street_FR>
5 <Institution_Street_NL>r. Fumal 28</Institution_Street_NL>
6 <Institution_HouseNumber>28</Institution_HouseNumber>
7 <Institution_ZipCode>5000</Institution_ZipCode>
8 <Institution_City_FR>Namur</Institution_City_FR>
9 <Institution_City_NL>Namur</Institution_City_NL>
```

The institutions' addresses must be transformed into spatial points, given that the geolocation API must receive a string representing the full address. Thus, to produce the full addresses that are the input to the geocoding process, the information in Listing 6 must be concatenated, for example, to get something like: "r. Fumal 28 5000 Namur". To avoid transforming XML data into relational tables, XPath (http://www.w3.org/TR/xpath/) was used instead of SQL to define the logical tables in the mapping document, as explained in Section 3.3. However, XPath does not allow creating a new node containing the full address, which is needed in order to obtain the corresponding spatial coordinates. Thus, XSLT(http://www.w3.org/TR/xslt) was used to produce a temporary file with such full addresses. The XSLT code is shown in Listing 7. The code browses the XML file, and each node is simply copied in a new file. When a node like "Institution_Street_FR" is reached, the zip code and city

terms are concatenated, and a new node is created, with the name "Institution_Full_Address" (although data come in Dutch and French, the latter was used to find out the coordinates). The temporary file is then used as the original file on which the mapping is performed.

Listing 7: XSL code for creating a node.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:template match="/">
<xsl:apply-templates select="*"/>
</xsl:template>
<xsl:template match="node()">
<xsl:copy><xsl:apply-templates select="node()"/></xsl:copy>
</xsl:template>
<xsl:template match="Institution_Street_FR">
<xsl:copy><xsl:apply-templates select="node()"/></xsl:copy>
<Institution_Full_Address>
<xsl:apply-templates select="node()"/>
<xsl:text> </xsl:text>
<xsl:value-of select="../Institution_ZipCode"/>
<xsl:text>, </xsl:text>
<xsl:value-of select="../Institution_City_FR"/>
</Institution_Full_Address>
</xsl:template>
</xsl:stylesheet>
```

After running this code, the full address for the institution above will look as depicted in Listing 8

Listing 8: Full institution address resulting from the XSLT program in Listing 7.

```
<InstitutionID>3</InstitutionID>
<Institution_Full_Address>r. Fumal 28, 5000 Namur
</Institution_Full_Address>
```

After this process, a spatial mapping file can be produced and applied over the new file. The mapping file is shown in Listing 9. Note that only the part relevant to the spatial mapping is shown. An XPath query was used as the logical table.

Listing 9: (Spatial) Mapping file for an institution.

```
...
rr:logicalTable [
rr:xpathQuery """//Institution_Instance~
./InstitutionID~
./Institution_Street_FR~
./Institution_Street_NL~
./Institution_HouseNumber~
./Institution_ZipCode~
./Institution_City_FR~
./Institution_City_NL~
./Institution_Full_Address; """;
];

rr:subjectMap [
rr:class gospl:Address;
rr:template "http://www.agenda.be/db/Address_of_Institution/{InstitutionID}";];
...
rr:predicateObjectMap [
rr:predicate geo:geometry;
rr:objectMap [
rr:column "Institution_Full_Address";
rr:termType rr:Literal;
rr:data type virtrdf:Geometry;];];.
```

The spatial mapping produced triples in stRDF format, where the predicate is <http://www.w3.org/2003/01/geo/wgs84_pos#geometry>, and the object is a geometry in the WKT stRDF format. The following URL: http://www.opengis.net/def/crs/EPSG/0/4326 represents the spatial reference system used.

Listing 10 depicts the mapping of the full address for the institution with ID 24, obtained applying the process explained above. The workflow of the whole process is given in Figure 1. The coordinates

were obtained applying geocoding to the element `Institution_Full_Address` (note that the ''a'' in Turtle means `type`, for example, institution is of type address). For example, the address above can be obtained using the OpenStreetMap's `search` API as follows: https://nominatim.openstreetmap.org/search/r.Fumal%2028%20Namur?format=json&addressdetails=1&limit=1&point_geojson.

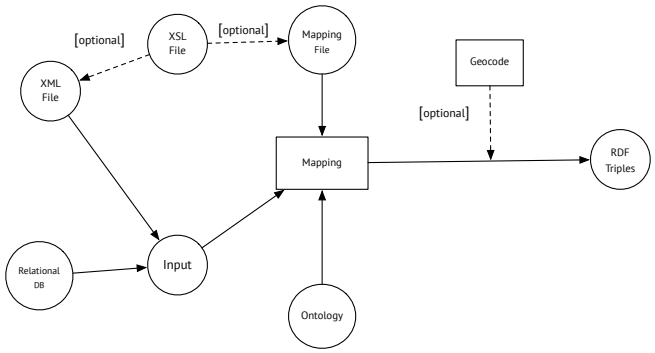

Figure 1. R2RML mapping workflow.

Listing 10: A triple produced by the spatial mapping for an institution address.

```
....
<http://www.agenda.be/db/Address_of_Institution/24>
a <http://starpc18.vub.ac.be/gospl/ontology/2#Address>;
<http://starpc18.vub.ac.be/gospl/ontology/2#Address_of_Institution>
<http://www.agenda.be/db/Institution/24>;
<http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
"POINT(4.34926,50.8486);http://www.opengis.net/def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/ontology#WKT> .
```

The second dataset to be mapped was the one belonging to Bozar. Bozar is Brussels' Center of Fine Arts, hosting cultural events, concerts, and exhibitions all year long. The logical table in the Bozar mapping file was an aggregation of several tables dealing with activities. Addresses were stored in a table denoted location_lng, with schema (id, lng, field, content). The field attribute contained the types of the components of a certain address (e.g., zip code, city, etc.). Table 2 shows an example of the location_lng table.

Table 2. Sample of the `location_lng` table of bozar database.

id	lng	Field	Content
231	fr	zip	1050
231	fr	name	Mussé d'Ixelles
231	fr	address	Jean Van Volsem 71
231	fr	country	Belgique
231	fr	city	Ixelles

The five tuples in Table 2 represent a single address. In order to perform the mapping, the five tuples in Table 2 must be concatenated. This concatenation is performed by an SQL query and yields the tuple $\langle Mussé\ d'Ixelles\ 71\ JeanVanVolsem\ 1050\ Ixelles\ Belgique \rangle$, over the schema (fullAddress,id), as shown in Table 3. Over this result, the mapping in Listing 11 is applied, and the result is shown in Listing 12.

Table 3. Concatenation of tuples in Table 2.

fullAddress	id
Mussé d'Ixelles 71 Jean Van Volsem 1050 Ixelles Belgique	231

Listing 11: Mapping file for locations from Bozar.

```
rr:subjectMap [
rr:template "http://bozar.be/db/Locations/{id}";
rr:class gospl:Location; ];

rr:predicateObjectMap [
rr:predicate geo:geometry;
rr:objectMap [
rr:column "fullAddress";
rr:termType rr:Literal;
rr:data type virtrdf:Geometry;];];
```

Listing 12: Bozar: triples corresponding to location ID 231.

```
<http://bozar.be/db/Locations/231>
a <http://starpc18.vub.ac.be/gospl/ontology/2#Location>;
<http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
"POINT(4.37238 50.83169); http://www.opengis.net/def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/ontology#WKT>.
```

The third dataset to be mapped corresponded to STIB, the main company of public transport in Brussels. Data from STIB were stored in an SQL database that consisted of four tables: Block, Stop, Trip and Tripstop. A Block represents the time elapsed between the moments when a vehicle leaves the warehouse and returns to it. The Stop table contains the name (in French and Dutch) and location of a bus, tram, or metro stop. A Trip is a part of a block, defined as the path between the starting point and the ending point on a particular route. Finally, Tripstop tells the time during a trip when the vehicle reaches a stop. Figure 2 shows the interaction between components and describes the process in more detail. Tables 4–6 show instances of the dataset.

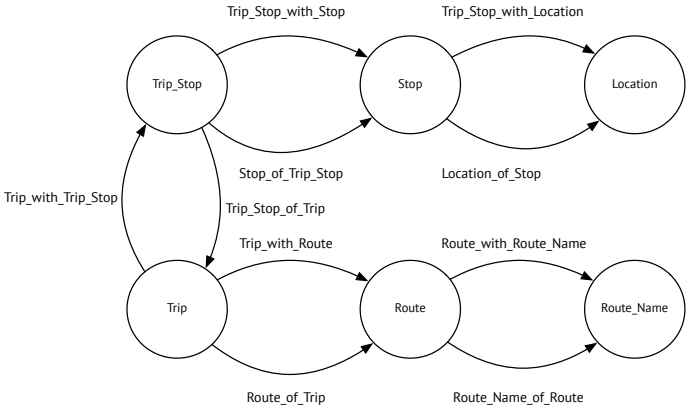

Figure 2. Relation between STIBroute components.

Table 4. Sample of the Stop table.

stp_Identifier	stp_Description	ud_stp_desc_Flam	stp_Longitude	stp_Latitude
6306	SAINTE-MARIE	SINT-MARIA	4.3681	50.8580
6307	BOTANIQUE	KRUIDTUIN	4.36569	50.8532

Table 5. Sample of the Trip table.

trp_Block	trp_Time_Start	trp_Time_End	trp_Route_Public
001901	09:06:33	09:34:24	1
001901	10:14:03	10:40:51	1

Table 6. Sample of the TripStop table.

tstp_Stop_id	tstp_Passing_Time	Block_Number
8733	09:06:33	001901
8282	09:10:36	001901
8032	09:16:20	001901
8052	09:18:34	001901
8042	10:24:06	001901
8092	10:31:37	001901
8132	10:36:34	001901

Table 7 shows how the concepts in the ontology map to the columns in the tables of the STIB database.

Table 7. Mapping STIB database columns to concepts in the ontology.

Table	Column	Concept
stop	stp_identifier	gospl:Stop
	stp_description	gospl:Stop_with_Description
	ud_stp_desc_flam	gospl:Stop_with_Description
	stp_longitude	gospl:Location_with_Longitude
	stp_latitude	gospl:Location_with_Latitude
trip	trp_time_start	gospl:Trip_with_start_Time
	trp_time_end	gospl:Trip_with_end_Time
	trp_route_public	gospl:Route_with_Name
tripstop	tstp_passing_time	gospl:Trip_with_passing_time

The spatial mapping in the STIB case was straightforward, because longitude and latitude of a stop were given; therefore, each point could be computed through an SQL query. Listing 13 shows a portion of the mapping file for a stop.

Listing 13: STIB mapping for a stop.

```
<#TM_stop>
a rr:TriplesMap ;
rr:logicalTable [
rr:sqlQuery """
SELECT *
FROM stop
"""; ];

rr:subjectMap [
rr:template "http://www.stib.be/stop/{stp_identifier}";
rr:class gospl:Stop; ];

rr:predicateObjectMap [
rr:predicate gospl:Stop_with_Description;
rr:objectMap [
rr:column "stp_description";
rr:termType rr:Literal;
rr:language "fr-BE"; ];];
...
rr:predicateObjectMap [
rr:predicate gospl:Stop_with_Location;
rr:refObjectMap [
rr:parentTriplesMap <#TM_location>;
rr:joinCondition [
rr:child "stp_identifier";
rr:parent "stp_identifier";
```

```
];];];
.
```

Listing 14 depicts the R2RML mapping document for STIB's stop locations. Note, in the logical table, the concatenation in the SQL query mentioned above.

Listing 14: Mapping for a STIB stop location.

```
rr:logicalTable [
rr:sqlQuery """
SELECT stp_identifier, stp_longitude, stp_latitude, CONCAT('POINT(',stp_longitude,' ',stp_latitude,');
http://www.opengis.net/def/crs/EPSG/0/4326') as point
FROM stop """; ];

rr:subjectMap [
rr:template "http://www.stib.be/location/{stp_identifier}";
rr:class gospl:Location; ];
...
rr:predicateObjectMap [
rr:predicate gospl:Location_with_Latitude;
rr:objectMap [
rr:column "stp_latitude";
rr:termType rr:Literal;
rr:data type xsd:double; ];];

rr:predicateObjectMap [
rr:predicate geo:geometry;
rr:objectMap [
rr:column "point";
rr:termType rr:Literal;
rr:data type virtrdf:Geometry; ];];.
```

Finally, Listing 15 shows the resulting triple for the location of Stop 6306 in Table 4.

Listing 15: An RDF triple for stop #6306.

```
<http://www.stib.be/location/6306>
a <http://starpc18.vub.ac.be/gospl/ontology/60#Location> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Location_of_Stop>
<http://www.stib.be/stop/6306> ;
...
...
<http://starpc18.vub.ac.be/gospl/ontology/60#Location_with_Latitude>
"50.8580"^^<http://www.w3.org/2001/XMLSchema#double> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Location_with_Longitude>
"4.3681"^^<http://www.w3.org/2001/XMLSchema#double> ;
<http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
"POINT(4.3681 50.8580);http://www.opengis.net/def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/ontology#WKT> .
...
```

It can be seen that the POINT geometry was represented in WKT. Furthermore, the latitude and longitude coordinates were produced using the mapping.

To avoid redundancy, the mapping files for the trip stops and for the trips are not shown. However, to allow the reader to be fully able to understand what will finally be at the endpoint, the triples produced for the trip stops and for the trips are shown in Listings 16 and 17, respectively.

Listing 16: RDF triples for a trip stop.

```
<http://www.stib.be/trip_stop/8733/at/09:06:33>
a <http://starpc18.vub.ac.be/gospl/ontology/60#Trip_Stop> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_Stop_of_Trip>
<http://www.stib.be/trip/1/from/09:06:33/to/09:34:24> ;

<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_Stop_with_Stop>
<http://www.stib.be/stop/8733> ;

<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_with_passing_time>
"09:06:33"^^<http://www.w3.org/2001/XMLSchema#time> .
```

Listing 17: RDF triples for a trip between 09:06:33 and 09:34:24.

```
<http://www.stib.be/trip/1/from/09:06:33/to/09:34:24>
a <http://starpc18.vub.ac.be/gospl/ontology/60#Trip> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_with_Route>
<http://www.stib.be/route/1> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_with_Trip_Stop>
<http://www.stib.be/trip_stop/8733/at/09:06:33> ,
<http://www.stib.be/trip_stop/8282/at/09:10:36> ,
<http://www.stib.be/trip_stop/8032/at/09:16:20> ,
<http://www.stib.be/trip_stop/8052/at/09:18:34> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_with_start_Time>
"09:06:33"^^<http://www.w3.org/2001/XMLSchema#time> ;
<http://starpc18.vub.ac.be/gospl/ontology/60#Trip_with_end_Time>
"09:34:24"^^<http://www.w3.org/2001/XMLSchema#time> .

...
```

4. Adding External Datasets

The Agenda.be, Bozar, and STIB datasets were enriched with PoIs located in the city of Brussels, including restaurants, theaters, schools, and the like. These PoIs were obtained from the DBpedia, GeoNames, and LinkedGeoData sites described next and added to the endpoint.

DBpedia contains structured information extracted from Wikipedia, which can be queried via a SPARQL endpoint (http://dbpedia.org/sparql). Data from DBpedia were extracted building a Java program that queries a DBpedia-provided service and returns RDF triples. The property "`is dbpedia-owl:location of`" was used to find places of interest in Brussels. For example, the SPARQL query in Listing 18 retrieved all places in Brussels for which there was a latitude and a longitude and, optionally, a name and a Wikipedia page. For such places, the query returned the reference to a DBPedia page (in variable `?location`), the location's latitude and longitude (in variables `?lat` and `?long`, respectively), and the name of the place (`?name`), as well as the Wikipedia web page (`?wiki`) if the latter two existed. To represent the triples' predicates, the DBPedia properties were used, except for <`http://xmlns.com/foaf/0.1/name`>, where `http://www.w3.org/2000/01/rdf-schema#label` was used.

Listing 18: SPARQL query to retrieve all places in Brussels from DBPedia.

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT ?location ?lat ?long ?name ?wiki
WHERE {
?location dbo:location :Brussels .
?location geo:lat ?lat .
?location geo:long ?long .
OPTIONAL { ?location foaf:name ?name }.
OPTIONAL { ?location foaf:isPrimaryTopicOf ?wiki }.
}
```

An example of the extracted triples is displayed in Listing 19.

Listing 19: Result of triples for DBPedia data.

```
<http://dbpedia.org/resource/Church_of_Saint_Jacques-sur-Coudenberg>
<http://www.w3.org/2000/01/rdf-schema#label> "Church_of_Saint_Jacques-sur-Coudenberg@en";
<http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
"POINT(4.36028 50.8419; http://www.opengis.net/def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/ontology#WKT>;
<http://www.w3.org/2003/01/geo/wgs84_pos#lat> "50.8419^^http://www.w3.org/2001/XMLSchema#float";
<http://www.w3.org/2003/01/geo/wgs84_pos#long> "4.36028^^http://www.w3.org/2001/XMLSchema#float";
<http://xmlns.com/foaf/0.1/isPrimaryTopicOf> "http://en.Wikipedia.org/wiki/Church_of_Saint_Jacques-sur-Coudenberg".
```

GeoNames (http://geonames.org) is a geographical database of the world that contains over 25 million geographical names and more than 11 million features. GeoNames provides a Java library allowing one to access its web services through the code in Listing 20.

Listing 20: Piece of code for accessing GeoNames data.

```
WebService.setUserName(userName);
ToponymSearchCriteria searchCriteria = new ToponymSearchCriteria();
searchCriteria.setQ("Brussels");
searchCriteria.setCountryCode("BE");
searchCriteria.setAdminCode1("BRU");
...
ToponymSearchResult searchResult = WebService.search(searchCriteria);
...
```

The result of the search is then parsed, and the triples are built and put into stRDF format. Over 114 different places for Brussels were obtained in this way. For example, Listing 21 shows the information for the Brussels Central Square (Grand Place).

Listing 21: Triples resulting applying the code in Listing 20.

```
<http://geonames.org/6930484>
<http://www.w3.org/2000/01/rdf-schema#label> "Grand Place Brussels" ;
<http://www.geonames.org/ontology#countryCode> "BE" , "Belgium" ;
<http://www.geonames.org/ontology#feature> "L" , "parks,area, ..." ;
<http://www.geonames.org/ontology#featureCode> "PRK" ;
<http://www.w3.org/2003/01/geo/wgs84_pos#geometry> "POINT(4.35189 50.84681);
http://www.opengis.net/def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/ontology#WKT> ;
<http://www.w3.org/2003/01/geo/wgs84_pos#lat> "50.84681" ;
<http://www.w3.org/2003/01/geo/wgs84_pos#long> "4.35189" .
```

LinkedGeoData (http://linkedgeodata.org/) is a spatial database derived from OpenStreetMap, a project that aims at building a free editable world map. LinkedGeoData can be accessed and queried via a SPARQL endpoint. As in the case of DBpedia, a service was queried and RDF triples were constructed, transforming the results. Since LinkedGeoData does not contain information about the city, the user must check for the places actually in Brussels. This was done through a SPARQL query, which included a FILTER clause that looked for places within a radius of 8 km around Brussels' Grand Place (of course, this radius length was chosen arbitrarily), which is considered the city center (see Figure 3). Over 2191 new places were obtained in this way, running the query in Listing 22. This query retrieved all places with their labels, their type (e.g., restaurant, cafe, school), and their coordinates (longitude and latitude). The FILTER clause at the end verified that the place was in Brussels, as mentioned above. Examples of the resulting RDF triples are presented in Listing 23.

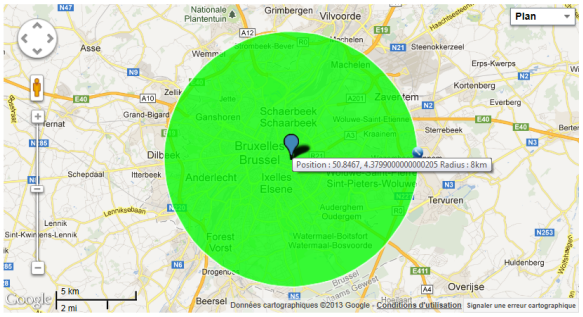

Figure 3. Defining the Brussels region.

Listing 22: SPARQL query to obtain places in Brussels from LinkedGeoData.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT *
From <http://linkedgeodata.org>
WHERE {
?s rdfs:label ?label .
?s geo:geometry ?geo .
?s rdf:type ?type .
?s geo:lat ?lat .
?s geo:long ?long~.

Filter (bif:st_intersects(?geo,<bif:st_point>(4.3799,50.8467),8)) .
}
```

Listing 23: Triples in Brussels obtained by the query in Listing 22.

```
<http://linkedgeodata.org/triplify/node319762946>
a <http://linkedgeodata.org/ontology/Node> ,
<http://linkedgeodata.org/ontology/Landuse> ,
<http://linkedgeodata.org/ontology/Railway> ,
<http://linkedgeodata.org/ontology/TramStop> ;
<http://www.w3.org/2000/01/rdf-schema#label> "ULB" ;
<http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
"POINT(4.38437 50.8136);http://www.opengis.net/def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/
ontology#WKT> ;
<http://www.w3.org/2003/01/geo/wgs84_pos#lat> "50.8136121"^^<http://www.w3.org/2001/XMLSchema#double> ;
<http://www.w3.org/2003/01/geo/wgs84_pos#long> "4.3843685"^^<http://www.w3.org/2001/XMLSchema#double> .
```

In total, 2355 places were collected from the three external sources. The sizes and number of triples of the data in the triple store are given in Table 8.

Table 8. Size of the datasets in the endpoint.

Dataset	Size (MB)	Size (Triples)
Agenda.be	35.2	222,207
Bozar	7.1	52,776
STIB	201	1,358,222
DBpedia	0.039	273
GeoNames	0.082	791
LinkedGeoData	1.6	10,950
Total	245.02	1,644,719

5. Querying the SPARQL Endpoint

With the data obtained as explained in previous sections, a Strabon triple store was built and deployed as an stSPARQL endpoint. This section shows how this endpoint can be exploited using the stSPARQL language. Figure 4 shows the user interface. Next, examples of queries that can be run over the endpoint (http://eao4.ulb.ac.be:8080/strabonendpoint/) are given. For the clarity of the presentation, these queries are classified according to their type. That is, this classification is not aimed at representing a query taxonomy. Nevertheless, the queries cover the characteristics of the queries included in the benchmark for geospatial RDF stores proposed by Garbis et al. [19] and more recently Ioannidis et al. [31]. The prefixes in Listing 24 were used in the queries, and they are not repeated for the sake of space.

Listing 24: Prefixes used in the SPARQL queries.

```
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX lont:<http://linkedgeodata.org/ontology/>
PREFIX gn:<http://www.geonames.org/ontology#>
PREFIX strdf:<http://strdf.di.uoa.gr/ontology#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX gospl:<http://starpc18.vub.ac.be/gospl/ontology/2#>
PREFIX stib:<http://starpc18.vub.ac.be/gospl/ontology/60#>
PREFIX postgis:<http://postgis.net/>
```

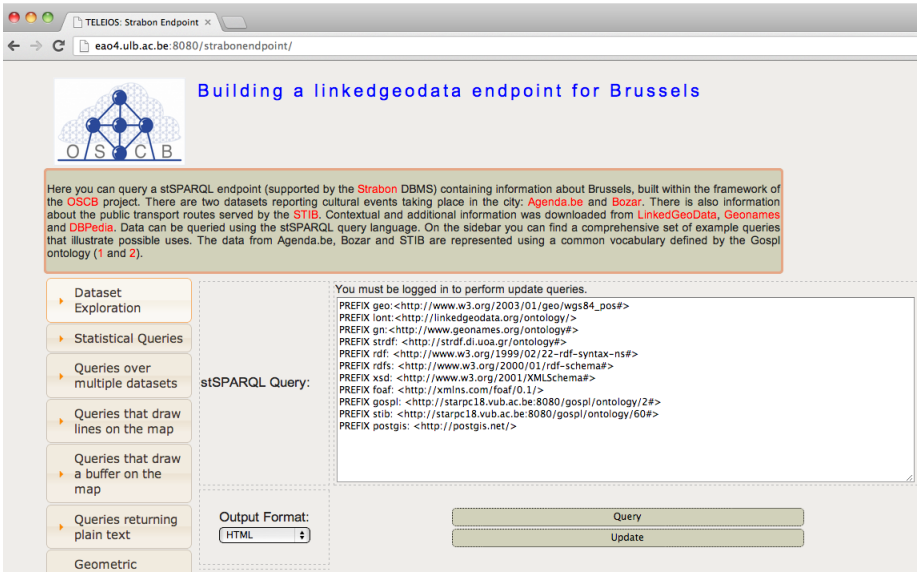

Figure 4. The stSPARQL endpoint's interface.

The endpoint prototype was installed on an OpenVZ container on a Debian server, with one core from a Xeon X3350, 1 GB of dedicated RAM, and RAID10 storage. The heap size is currently 5 GB. Most of the queries ran very fast, as readers can themselves check over the endpoint, and most of the queries executed on the sub-second timescale. However, experimental results are not reported here since this paper's focuses is on usability and query expressiveness.

5.1. Queries that Explore the Datasets

The first group of queries were posed over just one dataset and allowed exploring the dataset's data and metadata. Therefore, the stSPARQL queries contained only one `FROM` clause. Although these queries were quite simple, they were aimed at introducing the language.

Query 1

"List the Institutions in the Agenda.be dataset that located at less than 200 m from Brussels' Grand Place"
The query reads in stSPARQL:

```
SELECT ?name ?geo
FROM <http://agenda.be>
WHERE {
?node a gospl:Address.
?node gospl:Address_of_Institution ?ins .
?ins gospl:Institution_with_Name ?name .
?node geo:geometry ?geo.
filter(strdf:distance(?geo, "POINT (4.3525 50.8467);
http://www.opengis.net/def/crs/EPSG/0/4326", <http://www.opengis.net/def/uom/OGC/1.0/meter>) < 200)}
```

In this query, variables `?name` and `?geo` returned the name of an institution and the point coordinates of its location, which allowed displaying the result on a map. Note that the `FILTER` clause was used to keep just the institutions (represented by `?geo`) located less than 200 meters from "Grand Place" (it was assumed that the coordinates were known by the user or given by an application). The function `strdf:distance` computed the distance between the Grand Place and the corresponding institution. The result is shown in Figure 5.

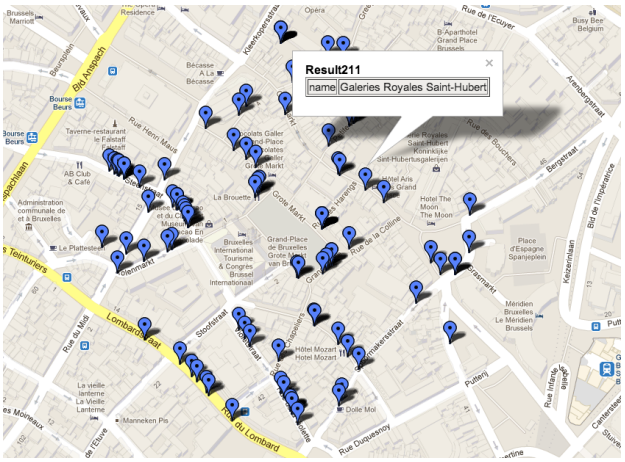

Figure 5. Institutions <200 m from Grand Place.

Query 2

"List all STIB stop locations of route 25"

```
SELECT DISTINCT ?stop ?geo ?dsc
FROM <http://stib.be>
WHERE {
?name stib:Route_with_Name "25" .
?rName stib:Route_with_Route_Name ?name.
?trip stib:Trip_with_Route ?rName .
?trip stib:Trip_with_Trip_Stop ?tripSt .
?tripSt stib:Trip_Stop_with_Stop ?stop .
?stop stib:Stop_with_Description ?dsc .
?node stib:Location_of_Stop ?stop .
?node geo:geometry ?geo.}
```

The query returned a link to the triple corresponding to the stop's location. The stop's location was returned in variable `?stop`. The geometry was returned in variable `?geo` (that is, the point coordinates, which allowed displaying the result on a map). The name of the stop was returned in variable `?dsc`. This is shown in the portion of the result as a JSON document, displayed below. The result in graphic form is shown in Figure 6.

```
{
"stop_loc": { "type": "uri", "value": "http:\/\/www.stib.be\/stop\/6212" },
"description": { "type": "literal", "xml:lang": "fr-be", "value": "STATION MEISER" },
"geo": { "type": "typed-literal", "data type": "http:\/\/strdf.di.uoa.gr\/ontology#WKT",
"value": "POINT(4.39477 50.8557);http:\/\/www.opengis.net\/def\/crs\/EPSG\/0\/4326" }
},
{
"stop_loc": { "type": "uri", "value": "http:\/\/www.stib.be\/stop\/5200" },
"description": { "type": "literal", "xml:lang": "nl-be", "value": "ROFFIAEN" },
"geo": { "type": "typed-literal", "data type": "http:\/\/strdf.di.uoa.gr\/ontology#WKT",
"value": "POINT(4.38263 50.8199);http:\/\/www.opengis.net\/def\/crs\/EPSG\/0\/4326" }
}
```

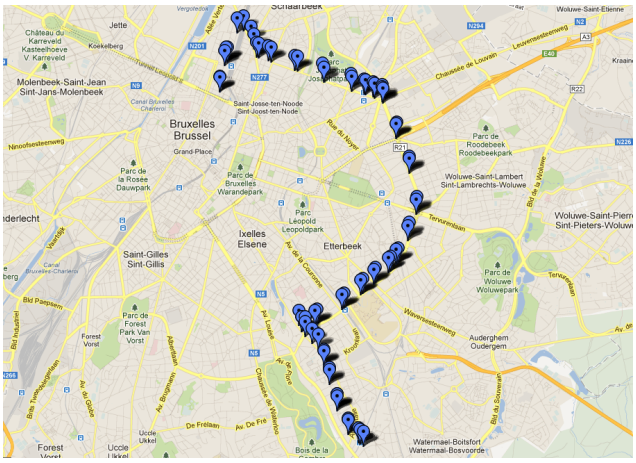

Figure 6. Stops of Tram 25.

5.2. Queries that Include Aggregation

This kind of query contained a `GROUP BY` clause. The first query of this kind computed the twenty stops closest to the Brussels Central Square.

Query 3

"List the twenty STIB stops closest to Brussels' Grand Place"

```
SELECT distinct ?stop (GROUP_CONCAT(distinct ?line) AS ?someLine) (SAMPLE(?dsc) AS ?someDescription)
?geo (strdf:distance(?geo, "POINT (4.3525 50.8467);http://www.opengis.net/def/crs/EPSG/0/4326",
<http://www.opengis.net/def/uom/OGC/1.0/meter>)as ?dist )
FROM <http://stib.be>
WHERE {
?stopLoc a stib:Location .
?stopLoc geo:geometry ?geo.
filter(strdf:distance(?geo, "POINT (4.3525 50.8467);http://www.opengis.net/def/crs/EPSG/0/4326",
<http://www.opengis.net/def/uom/OGC/1.0/meter>) < 1000)
?stopLoc stib:Location_of_Stop ?stop .
?stop stib:Stop_with_Description ?dsc~.

?trStop stib:Trip_Stop_with_Stop ?stop .
?trStop stib:Trip_Stop_of_Trip ?trip~.

?trip stib:Trip_with_Route ?rName .
?rName stib:Route_with_Route_Name ?name.
?name stib:Route_with_Name ?line . }
GROUP BY ?stop ?geo ?dist
ORDER BY ?dist
LIMIT 20
```

In the `SELECT` clause, the `GROUP_CONCAT` concatenated all the lines stopping at the same location (e.g., the ULBstop was used by Lines 25, 71, 72, and 94). The distance between a stop (`?geo`) and the "Grand Place" (`POINT (4.3525 50.8467)`) was computed and stored in variable `?dist`. In order to get the 20 closest stops, first the distance (`?dist`) between each stop and the "Grand Place" was computed. Then, the results were sorted by distance in ascending order, and finally, the closest 20 stops were kept, using the `LIMIT` keyword. The `GROUP BY` clause was used to ensure that a stop was only displayed once since it could have several descriptions (in French and Dutch) for the same location. Figure 7 displays the result.

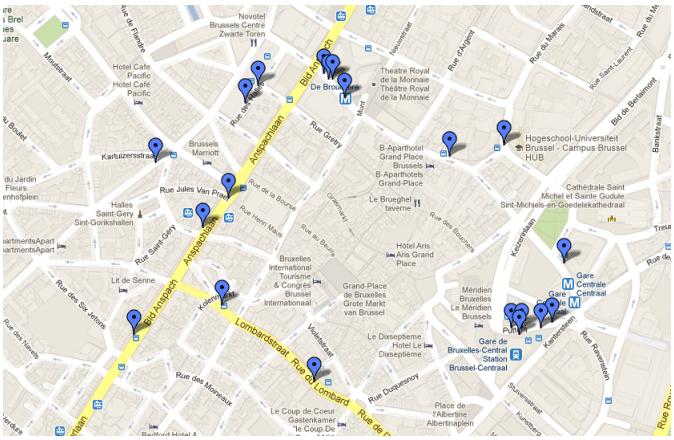

Figure 7. Stops closest to Grand Place.

Query 4

"For each STIB route, compute the number of stops"

```
SELECT (str(?res) as ?line) (count(*) as ?total)
FROM <http://stib.be>
WHERE {
?name stib:Route_with_Name ?line .
?route stib:Route_with_Route_Name ?name.
?route stib:Route_with_Stop ?stop .
BIND( xsd:integer(?line) as ?res)}
GROUP BY ?res
ORDER BY ?res
```

The aggregate function `count()` in this query counted the number of triples for each instantiation of the `?res` variable. The stops are displayed in ascending order, by using the `ORDER BY` clause. However, since the name of a route is a string, a casting into an integer had to be performed, using the `BIND` function. Then, it was converted back into a string in the `SELECT` clause, in order to avoid the data type "`xsd:integer`". With this little trick, the lines can be displayed in ascending order like "1", "2", "3" ... "98", instead of "1", "11" ... "2", "21" ... "98", which would be the case if they were strings. Table 9 shows the result.

Table 9. Number of stops per route.

Line	Total
"1"	"44"^^xsd:integer
"2"	"40"^^xsd:integer
"3"	"58"^^xsd:integer
"4"	"42"^^xsd:integer
...	...
"98"	"24"^^xsd:integer

5.3. Queries over More than One Dataset

Queries in this class may have more than one graph in the `FROM` clause.

Query 5

"Find the institutions in Brussels that host at least ten events"

```
SELECT ?institname ?geo?street ?zipcode ?cityname (GROUP_CONCAT(?title; separator = ' - / - ') as ?events)
FROM <http://bozar.be>
FROM <http://agenda.be>
WHERE {
```

```
?event gospl:Event_taking_place_at_Institution ?institname .
?instit gospl:Institution_with_Address ?addr .
?addr geo:geometry ?geo1 .
FILTER(strdf:distance(?geo1, "POINT (4.3799 50.8467); http://www.opengis.net/def/crs/EPSG/0/4326",
<http://www.opengis.net/def/uom/OGC/1.0/meter>) < 8000)

?event gospl:Event_has_Title ?eventname .
FILTER(langMatches(lang(?eventname ), "FR"))
?instit gospl:Institution_with_Name ?institname .
FILTER(langMatches(lang(?institname), "FR"))

?addr gospl:Address_with_Street ?street .
FILTER(langMatches(lang(?strt), "FR"))
?addr gospl:Address_with_Postal_Code ?zipcode .
?addr gospl:Address_with_City ?cityname.
FILTER(langMatches(lang(?city), "FR")) }
GROUP BY ?institname ?geo1 ?street ?zipcode ?cityname
HAVING (count(*) > 9)
```

This query displayed institutions from Agenda.be and Bozar (in variable `?geo`), together with their names (`?institname`), address (`?street`, `zipcode` and `?cityname`) and the events taking place at them (through the concatenation of events in `?events`). There were two `FROM` clauses mentioning both data graphs. Given that the mapping of the Bozar and Agenda.be data was performed using the vocabulary of the GOSPL-produced ontology, it was not possible to distinguish the source of an institution. In the query, results were first grouped by institution. Then, the total number of events by institution (`count(*)`) was computed, and finally, results were filtered through the `HAVING` clause. Finally, note that the first filter used the `strdf:distance` function to include only places that were within Brussels (as mentioned above, a place was considered to be within Brussels if it is within a radius of 8 km around the center).

Query 6

"Pairs of events taking place at institutions less than 500 m from each other, and such that one of the events is sold out"

```
SELECT ?title ?nameInstit ?title1 ?nameInstit1 (strdf:distance(?geo, ?geo1,
<http://www.opengis.net/def/uom/OGC/1.0/meter>) as ?distance)
FROM <http://bozar.be>
FROM <http://agenda.be>
WHERE {
?event gospl:Event_with_Sold_Out_Status "1"^^xsd:boolean .
?event gospl:Event_taking_place_at_Institution ?instit .
?event gospl:Event_has_Title ?eventname .
FILTER(langMatches(lang(?title), "FR"))
?instit gospl:Institution_with_Name ?nameInstit .
FILTER(langMatches(lang(?nameInstit), "FR"))

?instit gospl:Institution_with_Address ?addr .
?addr geo:geometry ?geo~.

?event1 gospl:Event_with_Sold_Out_Status "0"^^xsd:boolean .
?event1 gospl:Event_taking_place_at_Institution ?instit1~.

?instit1 gospl:Institution_with_Address ?addr1 .
?addr1 geo:geometry ?geo1 .
filter (strdf:distance(?geo, ?geo1, <http://www.opengis.net/def/uom/OGC/1.0/meter>) < 500)

?event1 gospl:Event_has_Title ?eventname1 .
FILTER(langMatches(lang(?eventname1), "FR"))
?instit1 gospl:Institution_with_Name ?nameInstit1 .
FILTER(langMatches(lang(?nameInstit1), "FR"))}
ORDER BY ?distance
```

In this query, a join between all sold out events (`"gospl:Event_with_Sold_Out_Status "1"`) and the ones that were not was performed, such that the distance between them was less than 500 meters. As a result, the title of the events was displayed (`?eventname` and `?eventname1`), together with

the name of the institution where they took place (`?nameInstit` and `?nameInstit1`). Results were finally sorted by distance between the two locations.

Query 7

"For each commune in Brussels, give the number of institutions"

```
SELECT ?city (count(*) as ?total)
FROM <http://agenda.be>
FROM <http://bozar.be>
WHERE
{
?instit1 a gospl:Institution .
?instit1 gospl:Institution_with_Address ?addr .
?addr gospl:Address_with_City ?city .
FILTER(langMatches(lang(?city), "FR"))
?addr gospl:Address_with_Postal_Code ?zip .
BIND(xsd:integer(?zip) as ?zip)
FILTER(?zip >= 1000 && ?zip <= 1210)
}
GROUP BY ?city
ORDER BY desc (?total)
```

Here, the variable `?city` represents each commune, and the expression `(count(*) as ?total)`, together with the `GROUP BY` clause, computed the number of institutions per commune. A portion of the result is shown, in JSON format.

```
"head": {
"vars": [ "city", "total" ]
},
"results": {
"bindings": [
{
"total": { "type": "typed-literal", "data type": "http:\/\/www.w3.org\/2001\/XMLSchema#integer", "value": "2444" },
"city": { "type": "literal", "xml:lang": "fr-be", "value": "Bruxelles" }
},
{
"total": { "type": "typed-literal", "data type": "http:\/\/www.w3.org\/2001\/XMLSchema#integer", "value": "913" },
"city": { "type": "literal", "xml:lang": "fr-be", "value": "Ixelles" }
},
{
"total": { "type": "typed-literal", "data type": "http:\/\/www.w3.org\/2001\/XMLSchema#integer", "value": "419" },
"city": { "type": "literal", "xml:lang": "fr-be", "value": "Saint-Gilles" }
},
{
"total": { "type": "typed-literal", "data type": "http:\/\/www.w3.org\/2001\/XMLSchema#integer", "value": "413" },
"city": { "type": "literal", "xml:lang": "fr-be", "value": "Schaerbeek" }
...
```

5.4. Queries Drawing Buffers or Lines on a Map

Many decision queries require drawing a buffer around a geometric object (e.g., a city, a river) or a line linking two points (e.g., two PoIs). These are the queries in this class, explained below.

Query 8

"Draw the places of interest within a radius of 200 m from a stop of line 94"

```
SELECT (strdf:buffer(?geo1, 200, <http://www.opengis.net/def/uom/OGC/1.0/meter>) as ?buf) ?geo ?label
WHERE {
?name stib:Route_with_Name "94" .
?route stib:Route_with_Route_Name ?name.
?route stib:Route_with_Stop ?stop~.

?stop stib:Stop_with_Description ?description.
?stop stib:Stop_with_Location ?loc .
?loc geo:geometry ?geo1.

?node a ?type .
?node geo:geometry ?geo .
```

```
FILTER(strdf:distance(?geo, ?geo1, <http://www.opengis.net/def/uom/OGC/1.0/meter>) < 200)
?node rdfs:label ?label .}
```

The query above used the function `buffer` to draw a circle with *?geo1* as the origin and a radius of 200 m. All places within a radius of 200 meters from a stop of Line 94 were displayed (using variable `?geo`) together with their names (in variable `?label`). The result is shown in Figure 8.

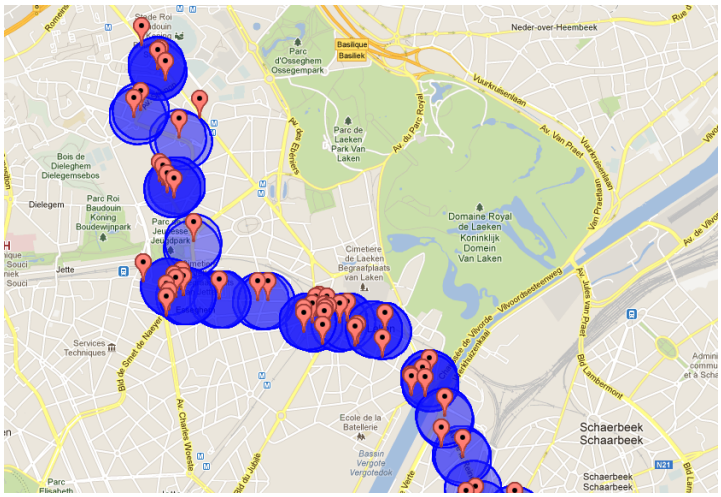

Figure 8. Places of interest less than 200 m away from a stop of Tram 94 (zoomed-in).

Query 9

Draw the places not closer than 400 meters and not farther than 900 meters from the Grand Place"

```
SELECT distinct (strdf:symDifference( strdf:buffer(?geo1, 400,
<http://www.opengis.net/def/uom/OGC/1.0/meter>),
strdf:buffer(?geo1, 900, <http://www.opengis.net/def/uom/OGC/1.0/meter>) )as ?sym) ?geo
FROM <http://linkedgeodata.org>
WHERE {
?node1 geo:geometry ?geo1.
?node1 rdfs:label "Grand Place"@fr.

?node rdfs:label ?name.
?node geo:geometry ?geo.

FILTER(strdf:within(?geo, strdf:symDifference(strdf:buffer(?geo1, 400,
<http://www.opengis.net/def/uom/OGC/1.0/meter>),
strdf:buffer(?geo1, 900, <http://www.opengis.net/def/uom/OGC/1.0/meter>))))}
```

The `strdf:symDifference()` function returned a new geometry, which was the symmetric difference between two geometries. The `strdf:within()` function was used in the filter in order to display places that were located in the new geometry constructed by the `symDifference` function. Figure 9 displays the result.

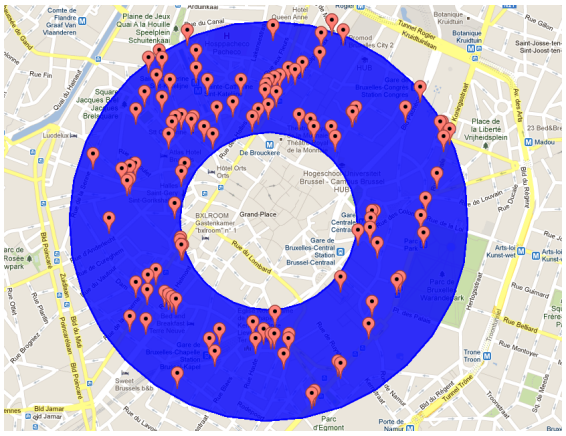

Figure 9. Places between 400 and 900 meters from Brussels' Grand Place.

Query 10

"Draw the lines between swimming pools and restaurants that are less than 1 km away from each other"

```
SELECT (postgis:ST_MakeLine(?geo, ?geo1) as ?line) ?label1 (strdf:distance(?geo, ?geo1,
<http://www.opengis.net/def/uom/OGC/1.0/meter>) as ?dist) ?geo1
FROM <http://linkedgeodata.org>
WHERE {
?node a lont:SwimmingPool .
?node geo:geometry ?geo .
?node rdfs:label ?label .
?node1 a lont:Restaurant .
?node1 geo:geometry ?geo1 .
?node1 rdfs:label ?label1 .
FILTER (strdf:distance(?geo, ?geo1,
<http://www.opengis.net/def/uom/OGC/1.0/meter>) < 1000)}
```

In this query, the `postgis:ST_MakeLine()` function was used to draw a line between two geometries (`?geo` and `?geo1`). The query also returned the location of the restaurants (`?geo1`) and their names (`?label1`). To see the distance, users must click on the nodes. Figure 10 shows the result.

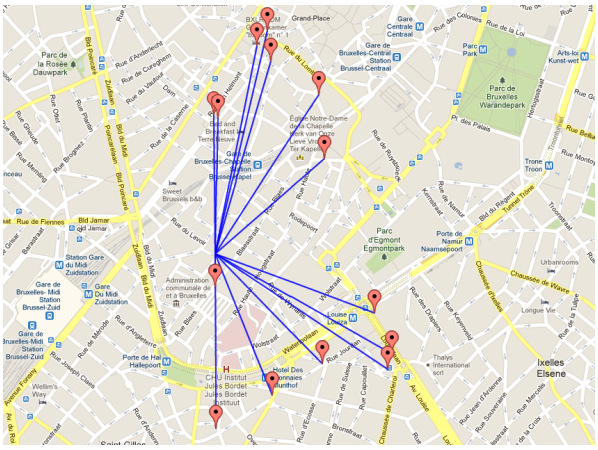

Figure 10. Swimming pools and restaurants less than 1 km away from each other.

5.5. Spatial Queries Including Temporal Conditions

Queries in this class, besides including one or more temporal condition(s), may also include the features in the classes already studied. For example, the next query includes aggregation, and it takes data from more than one graph.

Query 11

"Display all places where there is an event taking place on 3 May 22013, and that are in a radius of 100 m of a stop of lines 1, 3, 4, 5 or 6"

```
SELECT ?geo ?node ?eventname(GROUP_CONCAT(?description ; separator=", ") AS ?description)
?start ?end (strdf:buffer(?geo1, 100, <http://www.opengis.net/def/uom/OGC/1.0/meter>) as ?buf)
FROM <http://agenda.be>
FROM <http://bozar.be>
FROM <http://stib.be>
WHERE {
?node gospl:DateTimeSpecification_valid_from_Date ?start .
?node gospl:DateTimeSpecification_valid_until_Date ?end .
FILTER(?start <= "2013-05-23"^^xsd:date && "2013-05-23"^^xsd:date <= ?end)
?node gospl:Event_has_Title ?eventname .
FILTER(langMatches(lang(?eventname), "FR"))
?node gospl:Event_with_Description ?desc .
FILTER(langMatches(lang(?desc), "FR"))
?node gospl:Event_taking_place_at_Institution ?instit .
?instit gospl:Institution_with_Address ?address .
?address geo:geometry ?geo~.

?name stib:Route_with_Name ?line .
FILTER (?line = "1" || ?line = "3" || ?line = "4" || ?line = "5" || ?line = "6")
?route stib:Route_with_Route_Name ?name.
?route stib:Route_with_Stop ?stop .
?stop stib:Stop_with_Location ?loc .
?loc geo:geometry ?geo1.

FILTER (strdf:distance(?geo, ?geo1, <http://www.opengis.net/def/uom/OGC/1.0/meter>) < 100)
}
GROUP BY ?geo ?node ?eventname ?start ?end ?buf
```

The `FILTER` clause holds the temporal condition, which restricts the event date. The `buffer` function, again, draws a buffer around the tram stops. There are nine events in the result. A sample is shown below, as an XML output. Note that this query gets data from the three datasets.

```
<Point>
<coordinates>4.349437,50.848164</coordinates>
</Point>
...
<Data name = 'end'> <value>2013-06-02</value></Data>
<Data name = 'node'> <value>http://www.agenda.be/db/Event/264844</value></Data>
<Data name = 'eventname'> <value>Kiddy wax, la discotheque des enfants</value></Data>
<Data name='description'><value> </value>nouveau concept reserve aux enfants de 9 ans a
14 ans. Un dimanche par mois, le wax club reserve sa piste de dans aux enfants horaire 14h30 A
18h30... </Data>
.....
```

5.6. Miscellaneous

Query 12

"List the complete itinerary of all lines intersecting at the Buyl stop (with a tolerance of 100 meters)"

```
SELECT ?geo2 ?description (GROUP_CONCAT(?route; separator = ' - ') as ?allRoute)
FROM <http://stib.be>
WHERE
{
?stop1 stib:Stop_of_Route ?route .
?stop1 stib:Stop_with_Description ?description .
FILTER(langMatches(lang(?description), "FR"))
?stop1 stib:Stop_with_Location ?loc2 .
?loc2 geo:geometry ?geo2 .
{
SELECT distinct ?route
FROM <http://stib.be>
WHERE {
?stop stib:Stop_with_Description
"BUYL"@fr-BE .
```

```
?stop stib:Stop_with_Location ?loc .
?loc geo:geometry ?geo .
?loc1 geo:geometry ?geo1 .
FILTER(strdf:distance(?geo, ?geo1, <http://www.opengis.net/def/uom/OGC/1.0/meter>) < 100)
?loc1 stib:Location_of_Stop ?stop1 .
?stop1 stib:Stop_of_Route ?route .
} } }
GROUP BY ?geo2 ?description
```

This query included a subquery to find out all the routes passing within a radius of 100 m from the "BUYL" stop. This tolerance was required since several lines in Brussels intersect here. Then, all stops of each route were searched and displayed. The `GROUP BY` clause concatenated the routes in the `SELECT` clause. Figure 11 shows the lines intersecting at the stop: Tram Lines 7, 94, and 25, and Bus 71.

Figure 11. Lines intersecting at the "BUYL" stop.

Query 13

"Give me the next five times when tram 7 (direction Heysel) will be at the "Bascule" stop"

```
SELECT ?time
FROM <http://stib.be>
WHERE {
?name stib:Route_with_Name "7" .
?route stib:Route_with_Route_Name ?name.
?stop stib:Stop_with_Description "BASCULE"@fr-BE .
?stop stib:Stop_with_Direction "HEYSEL"@fr-BE .
?trip_stop stib:Trip_Stop_with_Stop ?stop .
?trip_stop stib:Trip_with_passing_time ?time .
filter (?time > "08:00:00"^^xsd:time)
}
ORDER BY ?time
LIMIT 5
```

The answer returned by this query is listed below, as plain text.

```
time
"08:02:53"^^<http://www.w3.org/2001/XMLSchema#time>
"08:08:53"^^<http://www.w3.org/2001/XMLSchema#time>
"08:14:53"^^<http://www.w3.org/2001/XMLSchema#time>
"08:20:53"^^<http://www.w3.org/2001/XMLSchema#time>
"08:26:53"^^<http://www.w3.org/2001/XMLSchema#time>
```

6. Conclusions

The paper studied the problem of capturing spatiotemporal data from different data sources, integrating these data, storing them in a geospatial RDF data store, and exposing the integrated data in a spatially-enabled SPARQL endpoint. The source data used for the research reported in this

paper were obtained from relational databases, XML documents, and different external Semantic Web repositories (accessed through APIs and SPARQL queries). The case study used in this paper is part of a larger project aimed at developing a prototype for integrating data from cultural events in Brussels with the public city transport schedule. Therefore, the data mentioned above were provided by the project partners, namely Agenda.be and Bozar (for the cultural data), and STIB (for the transportation data). These (relational and XML) data were enriched with external data coming from the Semantic Web, specifically from LinkedGeoData, GeoNames, and DBpedia.

The first research problem addressed consisted of mapping the spatial component of the source data, into spatial RDF. This was done by extending the standard R2RML mapping language in order to produce data represented in the stRDF data model supported by Strabon, the spatial data store used in this project. Since R2RML does not directly support XML data sources, it was also extended with such a capability. Over the deployed endpoint, the second problem tackled aimed at showing how these integrated data could be exploited. For this, a comprehensive set of stSPARQL queries was devised, divided into five query classes: dataset exploration queries, aggregate queries, queries involving several datasets, spatial analytical queries (e.g., queries that draw buffers), and queries involving temporal conditions. This variety illustrates the power of the solution. Last but not least, the paper also discussed the design alternatives considered to solve the problem at hand and the rationale for the decisions that were finally made.

Although for this project, only spatial geometries of type `point`were needed and addressed, the proposed solutions and the endpoint can be extended to include other kinds of spatial data, like for instance polygons. Moreover, since Strabon has support for column-store databases, the endpoint can be moved to platforms of this kind, to support larger databases efficiently. Finally, the work presented here can be used as a reference for new case studies that can tackle many other different domains.

Author Contributions: Both authors contributed evenly in the development of the material described in this work. The writing was mainly done by the first author.

Funding: This research received no external funding.

Acknowledgments: Alejandro Vaisman was partially funded by the OSCB (Open Semantic Cloud for Brussels) project, financed by Innoviris. He was also partially supported by PICT-2014 Project 0787 and PICT-2017 Project 1054, from the Argentina Scientific Agency.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Horrocks, I.; Parsia, B.; Patel-Schneider, P.; Hendler, J. Semantic Web Architecture: Stack or Two Towers? In *Principles and Practice of Semantic Web Reasoning (PPSWR)*; Springer: Berlin/Heidelberg, Germany, 2005; LNCS Volume 3703, pp. 37–41.
2. Klyne, G.; Carroll, J.J.; McBride, B. Resource Description Framework (RDF): Concepts and Abstract Syntax. 2004. Available online: http://www.w3.org/TR/rdf-concepts/ (accessed on 28 July 2019).
3. Beckett, D.; Berners-Lee, T.; Prud'hommeaux, E.; Carothers, G. Turtle-Terse RDF Triple Language. 2014. Available online: https://www.w3.org/TR/turtle/ (accessed on 28 July 2019).
4. Prud'hommeaux, E.; Seaborne, A. SPARQL 1.1 Query Language for RDF. 2011. Available online: http://www.w3.org/TR/sparql11-query/ (accessed on 28 July 2019).
5. Bereta, K.; Koubarakis, M. Ontop of Geospatial Databases. In Proceedings of the International Semantic Web Conference, Kobe, Japan, 17–21 October 2016; pp. 37–52.
6. Koubarakis, M.; Bereta, K.; Papadakis, G.; Savva, D.; Stamoulis, G. Big, Linked Geospatial Data and Its Applications in Earth Observation. *IEEE Internet Comput.* **2017**, *21*, 87–91. [CrossRef]
7. Burgstaller, S.; Angermair, W.; Migdall, S.; Bach, H.; Vlachopoulos, I.; Savva, D.; Smeros, P.; Stamoulis, G.; Bereta, K.; Koubarakis, M. LEOpatra: A Mobile Application for Smart Fertilization Based on Linked Data. In Proceedings of the 8th International Conference on Information and Communication Technologies in Agriculture, Food and Environment (HAICTA 2017), Chania, Greece, 21–24 September 2017; pp. 160–171.

8. Bereta, K.; Caumont, H.; Daniels, U.; Goor, E.; Koubarakis, M.; Pantazi, D.; Stamoulis, G.; Ubels, S.; Venus, V.; Wahyudi, F. The Copernicus App Lab project: Easy Access to Copernicus Data. In Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), Lisbon, Portugal, 26–29 March 2019; pp. 501–511.
9. Stadler, C.; Lehmann, J.; Höffner, K.; Auer, S. LinkedGeoData: A core for a web of spatial open data. *Semant. Web* **2012**, *3*, 333–354.
10. Janowicz, K.; Scheider, S.; Pehle, T.; Hart, G. Geospatial semantics and linked spatiotemporal data-Past, present, and future. *Semant. Web* **2012**, *3*, 321–332.
11. Koubarakis, M.; Karpathiotakis, M.; Kyzirakos, K.; Nikolaou, C.; Sioutis, M. Data Models and Query Languages for Linked Geospatial Data. In Proceedings of the Reasoning Web. Semantic Technologies for Advanced Query Answering, Vienna, Austria, 3–8 September 2012; Volume 7487, pp. 290–328.
12. Kyzirakos, K.; Vlachopoulos, I.; Savva, D.; Manegold, S.; Koubarakis, M. GeoTriples: A Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings. In Proceedings of the International Semantic Web Conference, Trentino, Italy, 19–23 October 2014; pp. 393–396.
13. Kyzirakos, K.; Savva, D.; Vlachopoulos, I.; Vasileiou, A.; Karalis, N.; Koubarakis, M.; Manegold, S. GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. *J. Web Semant.* **2018**, *52–53*, 16–32. [CrossRef]
14. Chentout, K.; Vaisman, A.A. Querying Brussels Spatiotemporal Linked Open Data. In Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Graz, Austria, 9–13 September 2013; pp. 378–387.
15. Chentout, K.; Vaisman, A.A. Adding Spatial Support to R2RML Mappings. In Proceedings of the OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Graz, Austria, 9–13 September 2013; pp. 398–407.
16. Bereta, K.; Smeros, P.; Koubarakis, M. Representation and Querying of Valid Time of Triples in Linked Geospatial Data. In Proceedings of the European Semantic Web Conference, Montpellier, France, 26–30 May 2013; pp. 259–274.
17. Kyzirakos, K.; Karpathiotakis, M.; Koubarakis, M. Strabon: A Semantic Geospatial DBMS. In Proceedings of the International Semantic Web Conference ISWC, Boston, MA, USA, 11–15 November 2012; pp. 295–311.
18. Nikolaou, C.; Koubarakis, M. Querying incomplete information in RDF with SPARQL. *Artif. Intell.* **2016**, *237*, 138–171. [CrossRef]
19. Garbis, G.; Kyzirakos, K.; Koubarakis, M. Geographica: A Benchmark for Geospatial RDF Stores (Long Version). In Proceedings of the ISWC 2013—12th International Semantic Web Conference, Part II, Sydney, Australia, 21–25 October 2013; pp. 343–359.
20. Lott, R. Geographic Information-Well-Known Text Representation of Coordinate Reference Systems. 2015. Available online: http://docs.opengeospatial.org/is/12-063r5/12-063r5.html (accessed on 1 July 2019).
21. Portele, C. OGC Geographic Markup Language (GML)—Extended Schemas and Encoding Rules. 2012. Available online: https://www.opengeospatial.org/standards/gml (accessed on 2 July 2019).
22. PostGIS. Spatial and Geographic objects for PostgreSQL. Available online: http://postgis.net/ (accessed on 2 July 2019).
23. Herring, J.R. OPENGIS-Implementation Standard for Geographic Information—Simple Feature Access—Part 1: Common Architecture. 2011. Available online: https://www.opengeospatial.org/standards/sfa (accessed on 27 July 2019).
24. Battle, R.; Kolas, D. Enabling the geospatial Semantic Web with Parliament and GeoSPARQL. *Semant. Web* **2012**, *3*, 355–370.
25. USeekM. USeekM: GeoSparql, Full Text Search, and Search Engine Integration Add-on for Triplestores. Available online: https://www.w3.org/2001/sw/wiki/USeekM (accessed on 25 July 2019).
26. Das, S.; Sundara, S.; Cyganiak, R. R2RML: RDB to RDF Mapping Language. 2012. Available online: http://www.w3.org/TR/r2rml/ (accessed on 30 July 2019).
27. Debruyne, C.; Meersman, R. GOSPL: A Method and Tool for Fact-Oriented Hybrid Ontology Engineering. In Proceedings of the European Conference on Advances in Databases and Information Systems, Poznan, Poland, 18–21 September 2012; pp. 153–166.

28. Dimou, A.; Sande, M.V.; Colpaert, P.; Verborgh, R.; Mannens, E.; de Walle, R.V. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In Proceedings of the Workshop on Linked Data on the Web Co-Located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, 8 April 2014.
29. Heyvaert, P.; Chaves-Fraga, D.; Priyatna, F.; Corcho, Ó.; Mannens, E.; Verborgh, R.; Dimou, A. Conformance Test Cases for the RDF Mapping Language (RML). In Proceedings of the Iberoamerican Knowledge Graphs and Semantic Web Conference, Villa Clara, Cuba, 23–30 June 2019; pp. 162–173.
30. Prud'hommeaux, E.; Arenas, M.; Bertails, A.; Sequeda, J. A Direct Mapping of Relational Data to RDF. W3C, Februrary 2012. 2012. Available online: http://www.w3.org/TR/rdb-direct-mapping/ (accessed on 30 July 2019).
31. Ioannidis, T.; Garbis, G.; Kyzirakos, K.; Bereta, K.; Koubarakis, M. Evaluating Geospatial RDF stores Using the Benchmark Geographica 2. *arXiv* **2019**, arXiv:1906.01933.