Adding Spatial Support to R2RML Mappings

Kevin Chentout and Alejandro Vaisman

Université Libre de Bruxelles
{kchentou,avaisman}@ulb.ac.be

Abstract. The "Open Semantic Cloud for Brussels" (OSCB) project aims at building a platform for linked open data for the Brussels region in Belgium, such that participants can easily publish their data. In OSCB, data providers deliver their data in the form of relational tables or XML documents. These data are mapped to RDF triples using the R2RML mapping language. Since OSCB data are spatiotemporal in nature, we needed to adapt R2RML to be able to produce spatiotemporal linked open data in order to build to a spatial data-enabled SPARQL endpoint where the result of spatiotemporal SPARQL queries can be shown on a map. In this paper we show how we achieved this goal.

1 Introduction

"Open Data" aims at making public sector data easily available and encouraging researchers and application developers to analyze and build applications around that data in order to stimulate innovation, business and public interests. "Linked Data"[1] is a global initiative to interlink resources on the Web using Uniform Resource Identifiers (URI) for accessing the resources, and the Resource Description Framework (RDF) [5] for representing knowledge and annotating those resources. The conjunction of both concepts originated the notion of "Linked Open Data" (LOD). The goal of the "Open Semantic Cloud for Brussels" (OSCB) project[2] is to pave the way for building a platform for LOD for the region of Brussels. The objective is that participants benefit from the OSCB, linked data-based architecture to publish their data. To facilitate this, data will be published in RDF format, so final users and application developers would be able to query data in an ubiquitous, intuitive and simple way.

Background. RDF is a data model for expressing assertions over resources identified by an URI. Assertions are expressed as *subject-predicate-object* triples, where *subject* are always resources, and *predicate* and *object* could be resources or strings. *Blank nodes* are used to represent anonymous resources, typically with a structural function, e.g., to group a set of statements. Data values in RDF are called *literals*. Many formats for RDF serialization exist. In this paper we use Turtle [1], and assume that the reader is familiar with this notation.

[1] http://www.linkeddata.org
[2] http://www.oscb.be

Y.T. Demey and H. Panetto (Eds.): OTM 2013 Workshops, LNCS 8186, pp. 398–407, 2013.

SPARQL is the W3C standard query language for RDF [6]. 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 is expressed as a graph pattern in the `WHERE` clause, consisting basically in a set of triple patterns connected by the '.' operator.

Strabon[3] is an open-source semantic geospatial DBMS that can be used to store linked geospatial data expressed in the *stRDF format* (spatiotemporal RDF) and query them using stSPARQL (spatiotemporal SPARQL). Strabon extends the RDF store *Sesame* [2], allowing it to manage both thematic and spatial RDF data stored in the PostGIS spatial DBMS. In this way, Strabon provides features similar to those offered by geospatial DBMS that make it one of the richest RDF stores with geospatial support available today, and the main reason to adopt it as the spatial RDF data engine for the OSCB project. The data types `strdf:WKT` and `strdf:GML` allow representing geometries serialized using the OGC standards WKT and GML. The stSPARQL language extends SPARQL with functions that take as arguments spatial terms and can be used in the SELECT, FILTER, and HAVING clause of a query. Since in this paper we focus on the spatial mappings, we do not extend in the description of Strabon.

Paper Contributions. In OSCB, data providers deliver their data in relational tables or XML documents. To be openly accessed, these data are mapped to RDF triples using the R2RML mapping[4]. OSCB data are essentially spatiotemporal, therefore the mapping has to be adapted to produce spatiotemporal linked open data. Some of the datasets do not include spatial coordinates of the locations, so we had to: (a) compute the coordinates from addresses; and (b) account for this conversion in the mapping. Finally, we added data from external sources to provide context for query and analysis. In this paper we describe how we performed the above tasks in order to build a spatial data-enabled SPARQL endpoint, powered by the *Strabon* engine. In Section 2 we explain how we map spatial organizational data from OSCB contributors to stRDF triples. Section 3 describes how we obtain data and produce stRDF triples from public data providers. We also give an example of the queries that the spatial endpoint can support (Section 4). We conclude in Section 5.

2 Mapping Spatial Data Using R2RML

The W3C standard for mapping relational to RDF data, denoted R2RML, 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, composed of a "logical table", a "subject map", and zero or more "predicate object maps". A logical table is either a table name (using the predicate

[3] `http://www.strabon.di.uoa.gr/`
[4] `http://www.w3.org/TR/r2rml/`

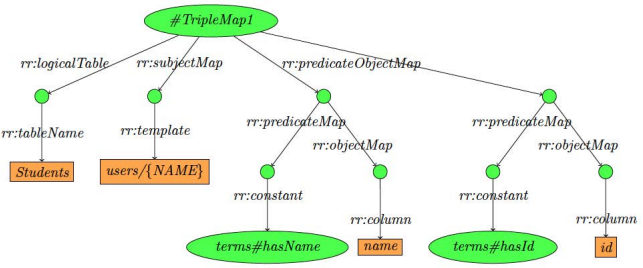

Fig. 1. R2RML mapping example

`rr:tableName`) or an SQL query (using the 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 are constants (`rr:constant`), column-based maps (`rr:column`) or template-based maps (`rr:template`). As an example, Figure 1 shows how a table Students, with attributes id and name, is mapped to RDF using the R2RML specification. This mapping can be then applied to any instance of the table, to produce the triples.

In summary, given a relational or an XML database, we produce a mapping file containing the specification for translating relational data into RDF triples (using R2RML). The mappings are generated using a common vocabulary defined in the *Gospel* ontology[5]. Finally, a mapping engine takes the database instance and the mapping file to produce the RDF document.

Adding Spatial Support to R2RML. In what follows we use three datasets provided as input by three Belgian companies: *Agenda.be, Bozar,* and *STIB*. To geolocate places we need their longitude and latitude. However, sometimes datasets contain just the address of a place (street, number, city, and postal code), thus we need to convert a full address into coordinates. This is called geocoding. From the many geocoding providers, we used Yahoo! PlaceFinder[6], a RESTful Web service which can be accessed through a web browser to get the result in JSON format. We then parse the result and retrieve the longitude and latitude, which we use in our spatial mapping.

The **Agenda.be** data reports cultural events in Brussels and the institutions where they take place. It is delivered in two XML documents: events.xml, containing events, and institutions.xml. The size of the XML files is approximately 6MB. The place and organizer in events.xml correspond to an institution in institutions.xml. For example, the (non-spatial) mapping of the event #190989 in the XML file events.xml will produce the RDF triples in Listing 1.

```
<Event_Instance>
        <EventID>190989</EventID>
```

[5] http://starpc18.vub.ac.be:8080/gospl/ontology/60
[6] http://developer.yahoo.com/boss/geo/docs/index.html

```
        ...
        <Event_Place>
            <Event_Place_InstitutionID>246340</
                Event_Place_InstitutionID>
            ...
        </Event_Place>
        <Event_Organisator>
            <Event_Organisator_InstitutionID>246340</
                Event_Organisator_InstitutionID>
            ...
        </Event_Organisator>
        ...
</Event_Instance>
```

```
...
<http://www.agenda.be/db/Institution/246340>
      <http://starpc18.vub.ac.be:8080/gospl/ontology/2#
          Institution_organizer_of_Event>
              <http://www.agenda.be/db/Event/190989> ;
      <http://starpc18.vub.ac.be:8080/gospl/ontology/2#
          Institution_place_of_Event>
              <http://www.agenda.be/db/Event/190989> .
```

Listing 1. Result of non-spatial mapping of events and institutions

Once we have linked events and institutions to each other, we must transform the institutions' addresses into points. The *PlaceFinder* API must receive a string with the full address, thus we need to concatenate information to produce these data. Although we can browse the XML file with XPath, this language does not allow us to create new node containing the full address. Thus, we use XSLT which allows this. The code is shown in Listing 2. When the program reaches an "Institution_Street_FR" node (we use French to find the coordinates), it looks for the zip code and city and concatenates them in a new node "Institution_Full_Address". The temporary file is then used as the original file on which the mapping is performed. Below we show how addresses are represented.

```
<Institution_Street_FR>pl. Sainte-Croix 1</
    Institution_Street_FR>
<Institution_Street_NL>Heilig-Kruisplein 1</
    Institution_Street_NL>
<Institution_HouseNumber>1</Institution_HouseNumber>
<Institution_ZipCode>1050</Institution_ZipCode>
<Institution_City_FR>Ixelles</Institution_City_FR>
<Institution_City_NL>Elsene</Institution_City_NL>
```

```
<?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>
```

Listing 2. XSL file

Then, we apply the mapping in Listing 3 over the new file (we show the part relevant to the spatial mapping). Listing 4 shows the result of a concrete mapping. Note that the geographic triples are already generated in stRDF format.

```
...
    rr:logicalTable [
    rr:xpathQuery """//Institution_Instance~
      ./InstitutionID~
      ./Institution_Street_FR~
      ...
      ./Institution_City_FR~
      ./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:datatype virtrdf:Geometry;  ]; ];.
```

Listing 3. (Spatial) Mapping file for an institution

```
....
<http://www.agenda.be/db/Address_of_Institution/3>
      a    <http://starpc18.vub.ac.be:8080/gospl/ontology/2#
          Address> ;
```

```
      <http://starpc18.vub.ac.be:8080/gospl/ontology/2#
          Address_of_Institution>
            <http://www.agenda.be/db/Institution/3> ;
      <http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
            "POINT(4.35752750.854287);http://www.opengis.net/
                def/crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr
                /ontology#WKT> .
```

Listing 4. Portion of the result of the mapping

The **BOZAR database** stores approximately 130MB of data. The logical table in the Bozar mapping file is an aggregation of several tables dealing with activities. Addresses are stored in a table denoted location_lng, which contains information about name, address, zip and city. Its schema has the form (id, lng, field, content). The address information is represented as values of the attribute field. The concatenation is performed with a simple SQL query. For example, from the set: $\{< 110, fr, zip, 1070 >, < 110, fr, name, ABC\ Factory >, < 110, fr, address, 175\ rue\ Bara >, < 110, fr, country, Belgique >, < 110, fr, city, Bruxelles >\}$, we obtain the tuple with schema (fullAddress,id): $< ABC\ Factory\ 175\ rue\ Bara\ 1070\ Bruxelles\ Belgique >$. The mapping file and resulting triple are depicted in Listings 5 and 6.

```
    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:datatype virtrdf:Geometry;];];
```

Listing 5. Mapping file for locations from Bozar

```
<http://bozar.be/db/Locations/110>
      a       <http://starpc18.vub.ac.be:8080/gospl/ontology
          /2#Location> ;
      <http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
              "POINT(4.292185 50.82917);http://www.opengis.
                  net/def/crs/EPSG/0/4326"^^<http://strdf.di.
                  uoa.gr/ontology#WKT> .
```

Listing 6. Triples for location id

The third component of our use case adds the *spatiotemporal* dimension. The **STIB**, *Société des Transports Intercommunaux de Bruxelles*, is the main company of public transport in Brussels. Data from STIB are stored in an SQL database that consists of four tables: Block, Stop, Trip and Tripstop. A Block can be seen as the time elapsed between the moment in which a vehicle leaves the warehouse and returns to it. The Stop table corresponds to a stop (bus, tram or

Table 1. Sample of Stop table

stp_identifier	stp_description	ud_stp_desc_flam	stp_longitude	stp_latitude
8042	ARTS-LOI	KUNST-WET	4.36963	50.8453
8032	PARC	PARK	4.36293	50.8458

metro) with its description, in French and Dutch, and its location, in x and y coordinates as well as in GPS coordinates (longitude / latitude). A Trip is a part of a block and is defined as the path between the starting point and the ending point carried on a particular route. Finally, Tripstop describes the stop and the time during a trip when the vehicle reaches the stop. The spatial mapping here is straightforward because we already have the longitude and latitude of a stop, so we can compute the point as an SQL query. Listing 7 shows the mapping file for stop locations. Listing 8 shows the resulting triple for location of stop 8042 in Table 1. Note that although the mapping produces the latitude and longitude coordinates, as well as the POINT geometry in the WKT reference system, for the sake of space we only show the latter which is what we use in the endpoint.

```
<#TM_location>
    a rr:TriplesMap ;
    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:datatype xsd:double;   ];];

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

Listing 7. Mapping for a STIB location

```
<http://www.stib.be/location/8042>
   a    <http://starpc18.vub.ac.be:8080/gospl/ontology/60#
       Location> ;
   <http://starpc18.vub.ac.be:8080/gospl/ontology/60#
       Location_of_Stop>
         <http://www.stib.be/stop/8042> ;
  ...
   <http://www.w3.org/2003/01/geo/wgs84_pos#geometry>
        "POINT(4.36963 50.8453);http://www.opengis.net/def/
           crs/EPSG/0/4326"^^<http://strdf.di.uoa.gr/
           ontology#WKT> .
 ...
```

Listing 8. An RDF triple for location of stop 8042

3 Adding External Datasets

To give context to the queries over organizational data, we added places of interest in Brussels, e.g., restaurants, banks, etc. To obtain these data we queried three datasets from the linked open data web: DBPedia, GeoNames and LinkedGeoData. *DBPedia* contains structured information extracted from Wikipedia, which can be queried via a SPARQL endpoint[7]. For data extraction we built a tool in Java which queries a DBPedia service and returns RDF triples. To find places in Brussels, we used the property "`is dbpedia-owl:location of`". We obtained 50 different places in Brussels for which we know the location.

Our second source, *GeoNames*[8], is a geographical database covering all countries and containing information of over 8 million places. A geonames Java library allows us to access the geonames web services. We obtained 114 different places for Brussels. For example, for the Brussels central square (note that after obtaining the data we must convert spatial data into the stRDF format):

```
<http://geonames.org/6930484>
      <http://www.w3.org/2000/01/rdf-schema#label>
              "Grand Place Brussels" ;
      ...
      <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> ;
```

Listing 9. A triple from GeoNames

The third data source was *LinkedGeoData*[9], a spatial database derived from OpenStreetMaps, a project that aims at building a free world map database. It consists of more than 1 billion nodes and 100 million ways (geometric constructs in which the data are based). Data can be accessed and queried via a SPARQL

[7] `http://dbpedia.org/sparql`
[8] `http://geonames.org`
[9] `http://live.linkedgeodata.org/sparql`

endpoint. As it was done for DBPedia, we queried a service and constructed RDF triples transforming the results. Since LinkedGeoData does not provide information about the city, we checked if the places are really in Brussels, adding a FILTER clause which verifies if the place is in a radius of 8 km from the center of Brussels. We obtained 2191 new places. Thus, in total we collected 2355 places from the three sources external sources, and added these data to our use case dataset, to build the spatial-enabled SPARQL endpoint.

4 Querying the SPARQL Endpoint

With all data in place we can now pose queries over the Strabon-powered endpoint[10]. Queries can return textual data or spatial data on a map. We give full details of this in [3]. As an example of a spatiotemporal query, we can ask for the "*places of events occurring on the 23 May 2013 in a radius of 200m of a stop of tram line 3*". The following stSPARQL query computes the answer.

```
SELECT ?geo ?node ?title (GROUP_CONCAT(?description ;
    separator=",") AS ?someDescription) ?start ?end (strdf:
    buffer(?geo, 200, <http://www.opengis.net/def/uom/OGC
    /1.0/metre>) 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 ?title .
  FILTER(langMatches(lang(?title), "FR"))
  ?node gospl:Event_with_Description ?description .
  FILTER(langMatches(lang(?description), "FR"))
  ?node gospl:Event_taking_place_at_Institution ?inst .
  ?inst gospl:Institution_with_Address ?addr .
  ?addr geo:geometry ?geo .

  ?name stib:Route_with_Name ?line .
  FILTER (?line = "3")
  ?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/metre>) < 200)}
group by ?geo ?node ?title ?start ?end ?buf
```

[10] The endpoint can be accessed at `http://eao4.ulb.ac.be:8080/strabonendpoint/`

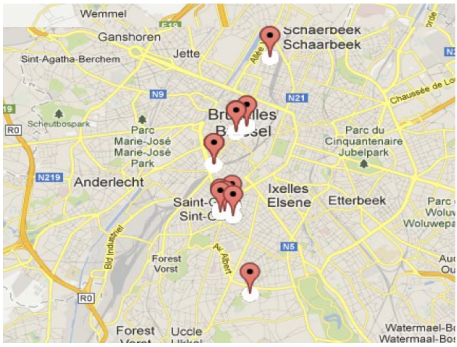

Fig. 2. Result of a query submitted to the endpoint

Figure 2 shows the result over a map. Note that in spite of the complexity of the query, it is expressed relatively easily. The query operates over the graphs in the `FROM` clause, then performs typical joins though the variables, and finally filters by route and distance. Other kinds of queries can be found at the endpoint.

5 Conclusion

In this paper we have described how relational to RDF mapping using the R2RML standard can be extended to support spatial data. We explain the method we used, and show the use cases employed in the OSCB project, which expose three different kinds of data: relational (Bozar), XML (Agenda.be), and spatiotemporal relational data (STIB). We show how we imported Brussels data from external sources in order to build a spatially-enabled SPARQL endpoint.

Acknowledgement. Alejandro Vaisman has been partially funded by the "Open Semantic Cloud for Brussels (OSCB)" project, funded by Innoviris.

References

1. Beckett, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language (2011), `http://www.w3.org/TeamSubmission/turtle/`
2. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In: Spinning the Semantic Web Dagstuhl Seminar, pp. 197–222 (2003)
3. Chentout, K., Vaisman, A.: Mapping Spatial Data using R2RML (submitted, 2013)
4. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (2010), `http://www.w3.org/TR/sparql11-query/`
5. Klyne, G., Carroll, J.J., McBride, B.: Resource Description Framework (RDF): Concepts and Abstract Syntax (2004), `http://www.w3.org/TR/rdf-concepts/`
6. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (2008), `http://www.w3.org/TR/rdf-sparql-query/`