Modern Ways of Spatial Data Publication

Wouter Beek (wouter@triply.cc) and Laurens Rietveld (laurens@triply.cc)

Triply (http://triply.cc)

v0.2

Almost every interesting dataset has some spatial component. Besides being prevalent, spatial relations – particularly geographical ones – tie the online to the offline world. As such, they provide a grounding of data stored in databases to the physical environment that is described in those databases.

Given the availability of Semantic Web services, Linked Datasets and Open Source web libraries we should be able to build a demonstration system that allows web programmers to build innovative applications on top of integrated Linked (Geo)datasets. Unfortunately, we found out that this is not (yet) the case.

1. Introduction

Earlier research by GeoNovum has resulted in a collection of lessons learned that describe in great detail the requirements of a modern spatial data publishing infrastructure. The purpose of the present report is document our research findings based on this prior work in an attempt to answer the following research question:

> How do the lessons learned meet the constraints (e.g. budgets) and capabilities (e.g. in-house know-how) of governmental organizations on the one hand, and of data users on the other?

While every governmental organization will be different, e.g., will be required to follow different rules and regulations depending on the domain or context in which it operates, it is possible to quantify the investment needed in order to build a Linked Geodata platform that implements the lessons learned.

2. First strategy: use a SotA triple store

When we started out building our geospatial Linked Data demonstrator we performed an industry-wide assessment of existing tools. For this we have consulted people from multiple domains. We first determined the leading query paradigm for geospatial Linked Data: this is GeoSPARQL [1]. We then determined the tool with the best GeoSPARQL support: this is Virtuoso[1]. We have loaded the data into an endpoint hosted at `http://sparql.geonovum.triply.cc/sparql`.

However, even Virtuoso does not support all geometries. Our data contains curves, resulting in the following error whenever a curve is encountered by the query engine:

```
Virtuoso 42000 Error GEO..: for after check of geo intersects, some
shape types (e.g., polygon rings and curves) are not yet supported
```

Here is an example of a query that gives the above warning ("Pairs of intersecting geometries that belong to the same resource."):

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
SELECT ?y1 ?y2
WHERE {
  ?x geo:asWKT ?y1 .
  ?x geo:asWKT ?y2
  FILTER (bif:st_intersects (?y1, ?y2, 0.1))
}
```

A second issue is that some queries do not terminate at all, as indicated by the following message:

```
Virtuoso S1T00 Error SR171: Transaction timed out
```

An example is an altered version of the above query ("Five pairs of dissimilar intersecting geometries that belong to the same resource."):

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
SELECT ?y1 ?y2 {
  ?x geo:asWKT ?y1 .
  ?x geo:asWKT ?y2
  FILTER (bif:st_intersects (?y1, ?y2, 0.1) && ?y1 != ?y2)
}
```

[1] See `https://github.com/openlink/virtuoso-opensource`

```
LIMIT 5
```

Thirdly, not all combinations of supported functions and supported shapes are supported. For instance, the following implements the geo query for our original use case:

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
SELECT ?y (MIN(bif:st_distance(?y, bif:st_point(0, 52))) AS ?z)
WHERE {
  ?x1 geo:asWKT ?y
}
LIMIT 5
```

In this particular case, the problem is that distance can only be calculated between points but not between a point and a surface, as communicated by the following message:

```
Virtuoso 22023 Error GEO..: Function st_distance() expects a geometry
of type 1 as argument 0, not geometry of type 10242
```

The fourth and biggest problem is that queries that combine geo functions and graph relations take *very* long to compute. For testing purposes we have set the limit for query execution to 10,000 (!) seconds and some queries still do not succeed within that time-frame:

```
Virtuoso 42000 Error The estimated execution time 12774 (sec)
exceeds the limit of 10000 (sec).
```

3. Second strategy: use a SotA Information Retrieval solution

We have corroborated our findings about the deficiency of existing triple stores with other developers. They are also deploying geospatial data for web users and they also discovered that existing triple stores are unable to support this out-of-the-box.

4. Users have different needs & capacities (lesson 1)

In this research project we have tried to address multiple groups of users by offering result set formats that they are most likely to be familiar with. Our demonstrator exposes the following result set formats:

- GeoJSON
- JSON-LD 1.0
- N-Quads 1.1, N-Triples 1.1

The N-Quads and N-Triples formats are, implementation-wise, the simplest RDF serialization formats available. They are supported by almost every RDF processor. Additional RDF serialization formats like Turtle 1.1, RDF/XML 1.1 and TRiG 1.1 are also available in ClioPatria. It would be a low-cost effort to expose them as well. We notice that the JavaScript RDF libraries that are currently being developed for client-side processing do not support, and will maybe never support, RDF/XML.

4.1. Header Dictionary Triples (HDT)

Another format that could be added is Header Dictionary Triples (HDT) [6] files. These are currently used to power the Graph API. They are also used to power Linked Data Fragments (LDF) [9] and the LOD Laundromat [3]. This format could be interesting for users who want to gather very large result sets.

Textual serialization formats work well for small results sets. For instance, if someone asks the ten nearest monuments then this can easily be returned in a text description. However, if someone has a larger-scale use case one may want to retrieve thousands of results. For instance someone may request all monuments in The Netherlands in order to generate a 'heat map' of areas with the most monuments. The result set of this will be hundreds of megabytes of textual data. The users will not be able to process this data off-hand, but would need to load this in her own database.

4.2. GeoJSON

The GeoJSON format is not yet fully standardized.[2] The GeoJSON RFC [4] is not a standard in the traditional sense of the word: a definition of all and only GeoJSON constructs and their meaning. For instance, the current RFC does not give a format grammar but only examples of GeoJSON in use.

It is not very difficult to implement GeoJSON, which is a very flexible format that can easily be combine with other JSON formats. Specifically, we were interested in mixing GeoJSON into JSON-LD constructs. Some people have worked on this in 2015 under

[2]GeoJSON is a proposed RFC standard as of August 2016.

the name 'geojson-ld'[3], but development in that direction has stalled.

The biggest hurdle towards integrating GeoJSON and JSON-LD into one format is that JSON arrays are used in JSON-LD for abbreviated object term notation. However, GeoJSON uses JSON arrays to represent geometries (nested lists of floating point coordinates). This point will be addressed in JSON-LD 1.1[4].

4.3. JSON-LD

JSON-LD 1.0 support was the most problematic to implement. It differs from other RDF serializations in that it does not translate a graph to a sequence of characters (i.e., a proper serialization), but it instead performs transformations between RDF graphs and JSON trees. As such it has to merry requirement from both paradigms. In this sense JSON-LD is similar to RDF/XML which is already relatively complex when compared to the N3-family of serializations (N3, Turtle, TRiG, N-Quads and N-Triples).

There are currently implementations for C#, Go, Java, JavaScript, PHP, Python, Ruby. Most notably support for C and C++, languages in which very many low-level database systems are written, is currently missing. Illustratively, one of the most widely used C++-based RDF processors, Raptor (`http://librdf.org/raptor/`) says on its web site:

> JSON-LD is not supported - too complex to implement.

Virtuoso does not support loading JSON-LD files[5] and ClioPatria does not support JSON-LD out-of-the-box either. We have written a partial implementation for generating JSON-LD and included it into library plRdf[6].

4.4. OGC standards (GML, WFS, WMS)

While GeoJSON addresses users from the web and geo domain, it may not address more advanced GIS users who may want to use more comprehensive formats standardized by the OGC, such as GML, WFS and WMS. The cost of integrating these OGC formats into a Linked Data platform are considerable, because they have their own vocabularies that need to be mapped to and from RDF. The OGC and W3C are currently working

[3] See `https://github.com/geojson/geojson-ld`
[4] See `https://github.com/json-ld/json-ld.org/issues/397`
[5] See `https://github.com/openlink/virtuoso-opensource/issues/478`
[6] See `https://github.com/wouterbeek/plRdf/blob/master/prolog/jsonld/jsonld_build.pl`

on this integration in the Spatial Data on the Web Working Group[7]. It is probably wise to wait until that WG has come up with a first (proposed) standard.

5. Findability (lesson 1a)

One of the most difficult things for web programmers, when confronted with the system, is to find out *what is in the data*. For existing web services this is usually not an issue: there is a limited number of entity types and relations that can be queries. For instance, a product review site has users who write reviews for products. Products may have companies producing them and users may themselves have ratings. That's about it.

Linked Data is very different: the monument dataset that we started out with contains over a hundred unique relationships between entities belonging to dozens of different types. And this is only one dataset. It is inherently impossible to provide a overview of what is in a large collection of Linked Data, in the same way in which it is impossible to provide an overview of what is in a large collection of the web. What we can do is come up with smart data visualization techniques. We found out that no such techniques are currently available.

6. Keep it simple (lesson 1B)

Simplicity is notoriously difficult to achieve in Linked Data. Simplicity is usually implemented by optimizing for a particular use case, but Linked Data promotes *arbitrary reuse*. However, this does not mean that exisitng approaches of Linked Geodata publishing cannot be simplified. We do observe that simplifications in Linked Geodata publishing are costly to implement because they must often be implemented at multiple layers.

6.1. Uniform representation of geodata

A concrete example of the high cost of simplifying Linked Geodata publishing arises when we want to make it easy to query all geodata in a uniforma anf simple way. Starting out with our source datasets this is not possible because geodata can be represented in different ways in RDF. The following pattern is commonly used:

[7] See `https://www.w3.org/2015/spatial/wiki/Main_Page`

```
entitiy:x geosparql:defaultGeometry _:a                    .
_:a       geosparql:asWKT           "POINT(5.46 51.35)"    ;
          rdf:type                  geosparql:Geometry     .
```

We believe that the above three statements can be rewritten to a single statement conveying the same meaning:

```
entity:x def:geometry "POINT(5.46 51.35)"^^def:point .
```

This version has the following benefits:

- No introduction of unnecessary blank node terms.
- More explicit type information of the kind of geometry (a point in this case).

These properties result in direct usability improvements:

- One less level of nesting simplifies querying. For instance, we need one LDF request to retrieve the geometry of an entity instead of two.
- We can query for geometries of a specific type without having to parse geometry values. For instance, for one of our map views we wanted to show polygons but not points and lines. We achieved this with the query

```
?x def:geometry ?y FILTER (datatype(?y) = def:polygon)
%\label{lst:datatype}
```

7. Data transformation

Geodata, as it is currently published, does not contain explicit links. Links can be inferred from implicit cues by human domain experts, but not by automated and domain-independent means. This means that implementing lesson 2b (Foster to link everything with everything) requires a relatively large investment.

The same value sometimes denotes the same thing. E.g., 'Appingedam' denotes a municipality. However, Appingedam is sometimes denoted by 'GM0003' (gemeentecode) and occurrences of the same string can denote different things (e.g., the area of Appingedam changes over time, the municipality of Appingedam is more than just the area of Appingedam, and a tourist uses 'Appingedam' to denote the center of Appingedam).

7.1. Value unpacking

A single value in the source dataset may describe multiple entities and relations between them. For instance the small 'buurtcode' string `BU00030002` denotes three entities and two spatial containment relations:

```
buurt:00030002 rdf:type         def:Buurt       .
gemeente:0003  geof:sfContains  wijk:000300     ;
               rdf:type         def:Gemeente    .
wijk:000300    geof:sfContains  buurt:00030002  ;
               rdf:type         def:Wijk        .
```

We call the conversion of a simple value to multiple entities and relations 'value unpacking'. Domain experts have 'packed' meaning into encodings like `BU00030002`. Non-domain experts and machines cannot 'see' the meaning that is inside the package. In order to make the meaning widely available we must 'unpack' it using a grammar. For the above code we have to construct the following grammar.[8]

```
buurt          := "BU" gemeente-code wijk-code buurt-code
buurt-code     := DIGIT DIGIT
gemeente       := "GM" gemeente-code
gemeente-code  := DIGIT DIGIT DIGIT DIGIT
wijk           := "WK" gemeente-code wijk-code(Wijk).
wijk-code      := DIGIT DIGIT
```

When loading the data we have to tell the machine to *parse* values within a given column, tag or key using the above grammar. The machine is then able to automatically construct the encoded entities and relations for all such values.

Notice that the above has to be done for every column, tag or key. Moreover, not all values can be parsed in this way. For instance, human language can generally not be parsed by a machine processor.

7.2. Value enrichment

Besides combining and splitting values, we sometimes find that a value is incomplete: it does not contain all the information we need in order to interpret it.

For instance, very many values are encoded in human languages. In order to do anything

[8]The grammar is written in Augmented Backus-Naur Form (ABNF) [5].

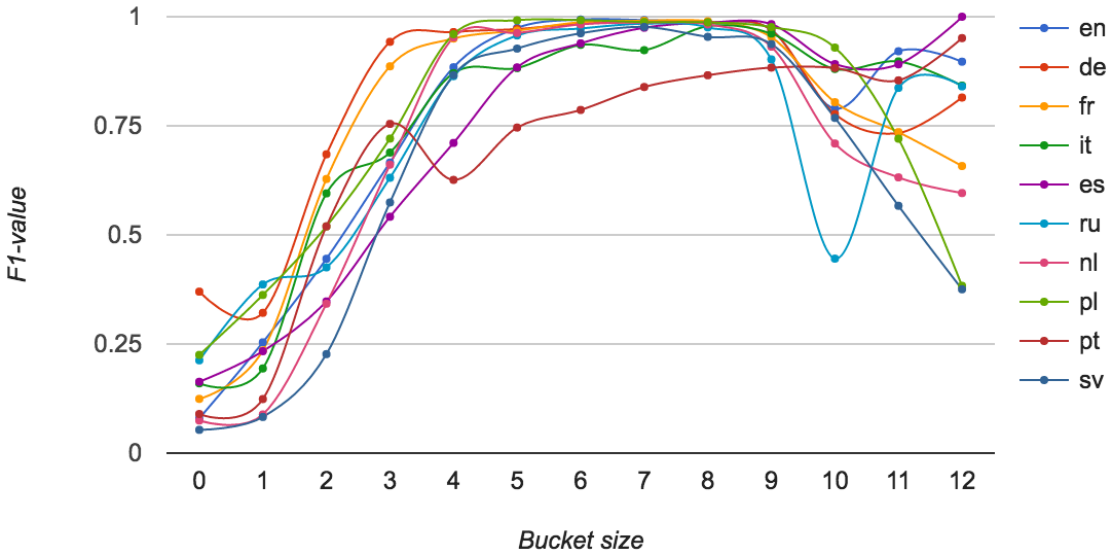

Figure 1: Accuracy of natural language detection in RDF strings for the 10 most frequent languages. Strings of similar length are combined in buckets. The size of buckets increases logarithmically. Figure taken from [2].

useful with these values we must at least know to which language they belong. For this we run automatic language detection algorithms on the values. The accuracy for medium-length strings is known to be very high (Figure 1

The main benefit of running natural language detection on the current data collection is to distinguish between Dutch and English strings.

7.3. Value combining

For example events are often described with a day, month and/or year column, tag or key:

```
<EVENT-ID> | 1997 | Aug | 7
```

In order for dates to be comparable with one another they have to be converted to values of XML Schema Part 2: Datatypes 1.1 [8]. Month names can be converted using a simple, one-to-one grammar (Section 7.1). After that values can be converted manually to datatypes.

```
event:x def:day   "07"^^xsd:gDay    ;
        def:jaar  "1997"^^xsd:gYear ;
        def:maand "8"^^xsd:gMonth   .
```

Values can now be automatically combined into the following more complex datatype:

```
event:x def:date "1997-08-07"^^xsd:date .
```

Because support for XML Schema Datatypes is very good overall, dates, times, durations, lengths, weights, etc. can now be compared through built-in functions. For instance the function `op:dateTime-less-than` is used by SPARQL 1.1 implementations to directly compare an event to all events that happened before it [7].

8. Persistent IRIs (lesson 2c)

It is very difficult to ensure persistent IRIs.

Most datasets have no IRIs at all, so we have to mint IRIs for each entity. While enriching the data (Section 7) the number of entities increases. These requires IRIs as well.

We have found the Dutch Linked Data URI strategy to be very beneficial during this process. It distinguishes between IRIs for instance (ABOX) and schema (TBOX) entities.

9. Quantifying the effort

There were also some 'hidden costs', i.e., either things that we expected to be very simple but that turned out to be very difficult, or things that we did not expect in the first place.

Firstly, we had to spend an enormous amount of time converting the various data formats (Section 7). After the conversion, we had to go through several iterations of transforming the data to improve its quality. While our transformed data has a much higher quality than the source data we started out with, we believe that there are still several more iterations needed to get the data to a quality level that allows generic (SPARQL) queries to be performed painlessly (i.e., without ad-hoc string manipulation).

Secondly, we expected to find a plethora of tools that allow Linked Data to be exposed to web programmers. After all, Linked Data is web data. We were very surprised to find

that this is not the case at all. In fact, there were many perfectly valid requests by the web programmers we worked with that could not be easily implemented by integrating some existing library.

9.1. Shortcuts

We believe that it is possible to speed up and reduce the cost of the process of developing a geospatial Linked Data web service by taking 'shortcuts'.

Points only Many of the deficiencies of SotA triple stores can be worked around by reducing all geometries to 2D points. GeoSPARQL functions between points are mostly supported. The downside to this is that much of the geospatial information is lost and very many geospatial query (contains, intersects) cannot be performed at all. This also do not solve the problem that some queries never terminate or that many queries take too long.

Start with Linked Data In this research project we had to split development costs between implementing a web service and converting/transforming data. By starting out with Linked Data a big chunk of the costs can be saved (approx. 40% in our case).

References

[1] Robert Battle and Dave Kolas. GeoSPARQL: Enabling a geospatial Semantic Web. *Semantic Web Journal*, 3(4):355–370, 2011.

[2] Wouter Beek, Filip Ilievski, Jeremy Debattista, Stefan Schlobach, and Jan Wielemaker. *Literally* better: Analyzing and improving the quality of literals. *Under submission*, 2016.

[3] Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A uniform way of publishing other people's dirty data. In *The Semantic Web–ISWC 2014*, pages 213–228. Springer, 2014.

[4] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. The GeoJSON format. *RFC 7946*, August 2016.

[5] D. Crocker. Augmented BNF for syntax specifications: ABNF. *RFC 5234*, 2008.

Alias	IRI prefix
geo	http://www.opengis.net/ont/geosparql#
bif	http://www.openlinksw.com/schemas/bif#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#

Table 1: Aliases for commonly occurring IRI prefixes.

[6] Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary RDF representation for publication and exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.

[7] Ashok Malhotra, Jim Melton, Norman Walsh, and Michael Kay. XQuery 1.0 and XPath 2.0 functions and operators (second edition). *W3C Recommendation*, April 2015.

[8] David Peterson, Shudi (Sandy) Gao, Ashok Malhotra, C.M. Sperberg-McQueen, and Henry S. Thompson. XML Schema Definition Language (XSD) 1.1 part 2: Datatypes, April 2012.

[9] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Querying datasets on the web with high availability. In *The Semantic Web–ISWC 2014*, pages 180–196. Springer, 2014.

A. Used aliases