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 to 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.

1.1. Primary use case

Primary use case: proximity queries

2. Storage

The first choice we have to make is how we want to store Linked Geospatial data. This choice has repercussions for almost all other aspects of the system, e.g., which queries can be performed and how long it takes to answer them.

2.1. First strategy: use a SotA triple store

Our first intuition was that SotA triple stores would be able to support geospatial data quite well. We were wrong in this. 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 running version 7.2 (March 2016) hosted at `http://sparql.geonovum.triply.cc/sparql`.

Unfortunately, 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

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

message:

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

An example is an altered version of the previous 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)
}
LIMIT 5
```

Thirdly, not all combinations of supported functions and supported shapes are supported. For instance, the following implements the query for our primary 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, but 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).
```

We must note that both Virtuoso and StarDog are very good triple stores overall and that they are working on improving geodata and GeoSPARQL support. It is difficult to assess when those improvements will be good enough to sufficiently support geospatial data publication.

2.2. Second strategy: use a SotA Information Retrieval solution

We have corroborated our findings about the deficiency of existing triple stores with other developers. One common approach is to perform graph queries in a triple store and geospatial queries in a document store such as Solr[2]. This approach results in a good performance and coverage for graph/SPARQL queries as well as good performance and coverage for geospatial queries. However, because the results come form two disconnected backends, it is generally not possible to efficiently perform a GeoSPARQL query in which graph and geospatial components are intertwined. Unfortunately, it is precisely in these integrated queries that the benefits of using Linked Data will become apparent (Section 1.1).

2.3. Third strategy: use SotA GIS libraries

Concluding that the previous two strategies will not result in a satisfactory spatial data publication platform, we decided to go for an out-of-the-box approach by looking beyond common Linked Data practices. We asked GIS experts what they use. PostGIS[3], based on the PostgreSQL[4] database, was unanimously considered the most performant tool for handling geospatial data. Other tool that was mentioned was qGIS[5], an Open Source GIS. Both tools use the backend library GEOS[6].

Our reasoning was that if PostgreSQL could be successfully combined with GEOS that we should be possible to combine a triple store with GEOS as well. In fact, we found a 6 year old research paper [11] that had attempted to do the same thing. The paper had combined GEOS as a plugin of ClioPatria [13], a triple store that we happened to be familiar with.

3. Communities have different needs & capacities (Lesson 1)

One of the ways in which multiple groups of users can be addressed is by offering result set formats that they are familiar with. Our demonstrator exposes the following result set formats:

[2]See `http://lucene.apache.org/solr/`
[3]See `http://postgis.net/`
[4]See `https://www.postgresql.org/`
[5]See `http://qgis.org/en/site/`
[6]See `https://trac.osgeo.org/geos`

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

N-Quads and N-Triples 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 widely available and are easy to add. We notice that the JavaScript RDF libraries[7] that are currently developed for client-side processing do not support, and will maybe never support, RDF/XML.

3.1. Header Dictionary Triples (HDT)

Another format that could be added is Header Dictionary Triples (HDT) [6]. These are currently used to power the Graph API to allow Basic Graph Pattern queries to be performed (similar to Linked Data Fragments (LDF) [12]). 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-based reply. However, some users have a large-scale use case, such as requesting *all* monuments in the Netherlands (e..g, with the purpose of data visualization) A textual result set of this size is difficult to process. With HDT the data is not only much smaller in size (as with regular compression) but can also be easily processed by the user.

3.2. JSON-LD

JSON-LD 1.0 support was the most problematic result format to implement. It differs from Turtle-based serialization formats in that it does not translate a graph to a sequence of characters, but it instead performs transformations between RDF graphs and JSON trees. As such it has to merry requirements from both paradigms. In this sense JSON-LD is similar to RDF/XML which undertakes a graph to/from tree mapping as well.

JSON-LD is valid JSON, the primary data interchange format for web programmers. As such, JSON-LD is a good way of lowering the entry level for this user group. The JSON-LD format is relatively costly to generate and process, because not all aspects of RDF can be directly encoded in JSON. For this a *context* that steers the data transformation step needs to be defined (while generating) and applied (while processing). However, as

[7]For more information, see the RDF JavaScript Libraries Group (`https://www.w3.org/community/rdfjs/`).

long as JSON-LD is used for interchanging smaller chunks of data, the extra processing time is not an issue.

There are currently JSON-LD implementations for C#, Go, Java, JavaScript, PHP, Python, Ruby. Most notably support for C and C++, languages, in which many low-level database systems and libraries are written, is missing. Sadly, the top C++-based RDF processor, Raptor[8], states on its web site that "JSON-LD is not supported - too complex to implement".

Virtuoso does not support loading JSON-LD files[9] 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[10].

3.3. GeoJSON

Another format we expose is GeoJSON. The GeoJSON format is not yet fully standardized[11] and the current RFC [4] is not a standard in the traditional sense of the word, i.e., a definition of all and only GeoJSON constructs and their meaning. For instance, the current RFC does not give a formal grammar but relies on examples to convey GeoJSON syntax and semantics.

An important difference with JSON-LD is that GeoJSON cannot be read back as Linked Data. As a consequence, GeoJSON should only be used in very specific circumstances. E.g., as an export format for applications that do not understand Linked Data, like Leaflet[12]. If a Linked Data-capable application accesses the data in GeoJSON it is unnecessarily throwing most of the RDF semantics away.

It is not difficult to implement GeoJSON, which is a very flexible format that can easily be combined with other JSON formats. Specifically, we were interested in mixing GeoJSON into JSON-LD constructs. This would make it a viable format for Linked Data and non-Linked Data applications alike. Some people have worked on this in 2015 under the name 'geojson-ld'[13], but development in that direction has now 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).

[8] See `http://librdf.org/raptor/`
[9] See `https://github.com/openlink/virtuoso-opensource/issues/478`
[10] See `https://github.com/wouterbeek/plRdf/blob/master/prolog/jsonld/jsonld_build.pl`
[11] GeoJSON is a proposed RFC standard as of August 2016.
[12] See `http://leafletjs.com`
[13] See `https://github.com/geojson/geojson-ld`

This point will be addressed in JSON-LD 1.1[14].

3.4. 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. These formats include 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. Luckily, the OGC and W3C are currently working on integrating their respective standards within the Spatial Data on the Web Working Group[15]. It is probably a good idea to wait until that Working Group has come up with a first (proposed) standard.

4. Findability (Lesson 1A)

One of the most difficult things for users of a Linked Geodata service, is to find out *what is in the data*. We distinguish between the problem of finding out what the vocabulary is (Section 4.2) and finding out which instances are described in the data (Section 4.2). A user often needs to know what is in the vocabulary, in order to be able to write a query to find particular instances.

4.1. Vocabulary overview

It is difficult to gain an overview of a Linked Data vocabulary. For non-Linked Data services this is usually not a problem: there is a limited number of entity types and relations that can be queried. For instance, a product review site may have users who write reviews for products. Products may have companies producing them and users may themselves have ratings to denote their trust level. 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 difficult to provide an overview of what is inside 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.

[14]See `https://github.com/json-ld/json-ld.org/issues/397`
[15]See `https://www.w3.org/2015/spatial/wiki/Main_Page`

Hierarchy visualization can be used to display the class and/or property hierarchy in a tree view. E.g., StarDog successfully uses this in their default data browser.

Domain/range visualization can be used to display the classes and properties in relation to each other. Value properties are displayed as part of the class nodes. Object properties are displayed as directed arcs between class nodes. This requires domain (`rdfs:domain`) and range (`rdfs:range`) information for the properties. Figure 1 shows an example for the Semantic blogging platform SemBlog.

If OWL cardinality restrictions are also present then this can also be used to annotate the arcs with. We are unaware of a tool that does this currently, but in theory it should be possible to generate visualizations that are somewhat akin to class diagrams in object-oriented software engineering (e.g., UML class diagrams).

Unfortunately, our source datasets contain very little hierarchical information and no domain/range information. In order for existing visualization techniques to be applicable we have to first perform a data enrichment operation. Data that is automatically converted from non-RDF formats will often lack this information.

Cost The cost of vocabulary visualization are low when the data contains sufficient hierarchical and/or domain/range information. Otherwise, the cost of adding such schema information to the dataset is relatively high.

4.2. Instance findability

There are two main approaches towards finding instances in Linked Datasets. The former is based on **text-based search**, which uses techniques from Information Retrieval to perform string matching in combination with relevance ranking. We implemented a simple text-based search feature that indexes all RDF literals and matches substrings against them.

Figure 3 shows the results for searching for the 'Hofnar' building. One result is the building as described by the BGT. Another result is a strike event that took place at the Hofnar building in 1979, where 24 workers demanded higher wages. The remaining result is unrelated, it is a monument with a depiction of a jester ('hofnar' in Dutch).

What is currently lacking is a good ranking over the matched results. This would put resources called 'Hofnar' higher in the search results than resources that only contain the word somewhere in a lengthy description. Better search results are obtained by using a dedicated IR backend like ElasticSearch/Lucene.

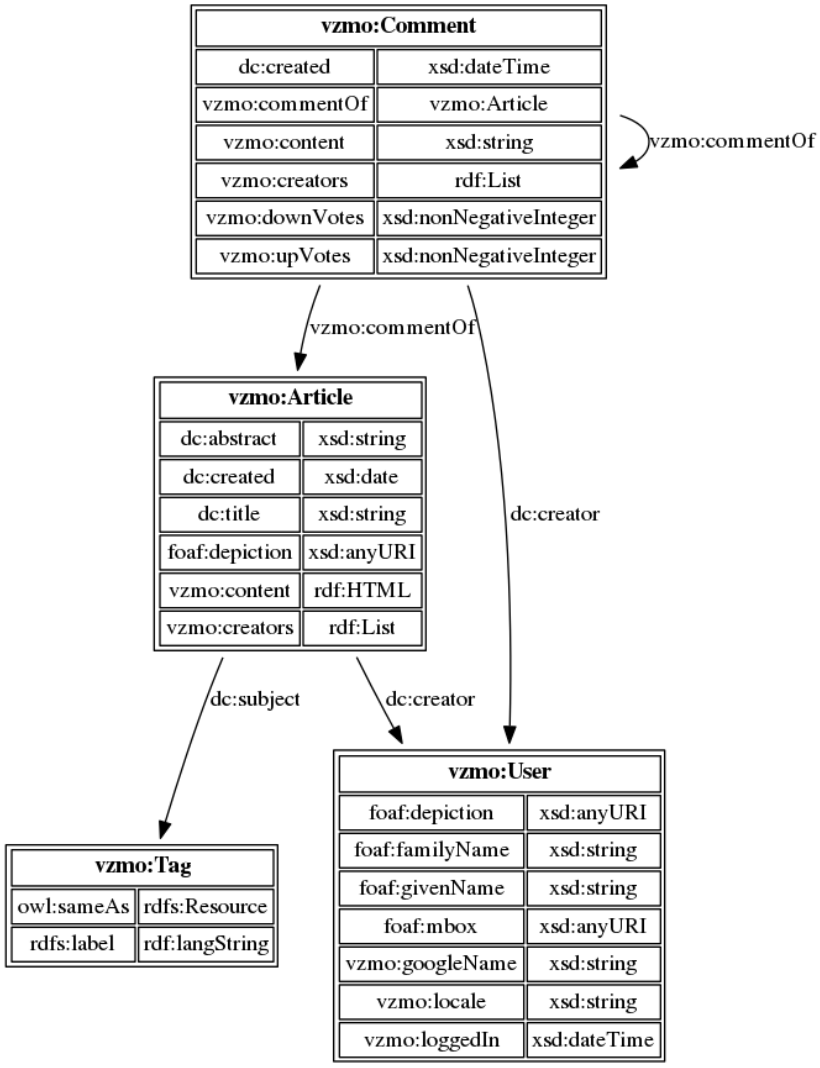

Figure 1: Example of a vocabulary visualization with 4 classes, 5 object properties, and 20 value properties.

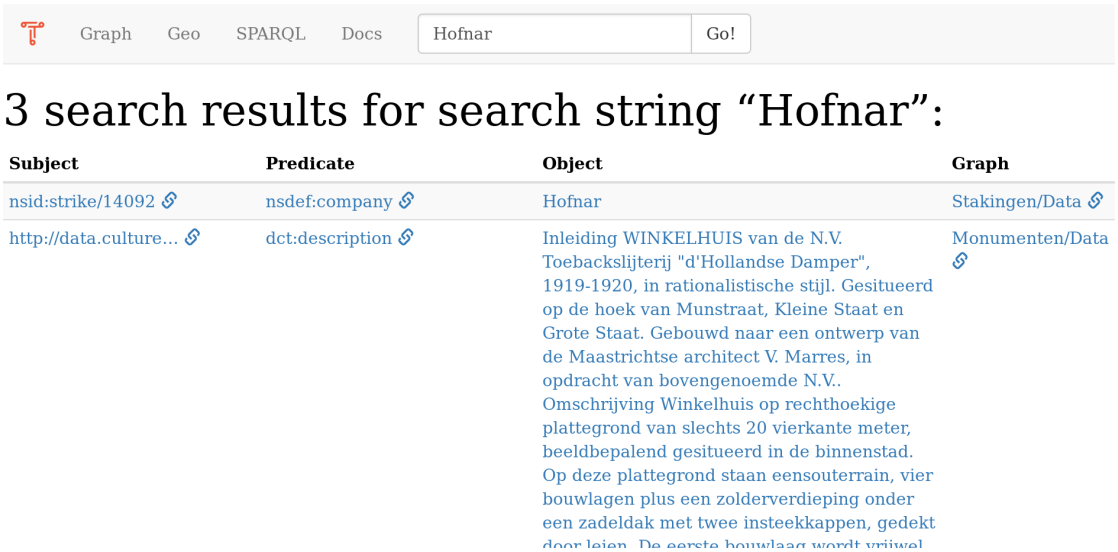

Figure 2: Results for the search “Hofnar”.

The second approach for finding instances in Linked Data is not based on string matching but on the structure of the vocabulary. It is implemented in **faceted browsers** that allow classes and properties to be selected from automatically generated option lists. By selecting options from these lists the user applies filters to the dataset, resulting in an increasingly smaller results set of entities that adhere to the specified filters. A screenshot of a faceted browser is shown in Figure **??**.

We did not add a faceted browser because this requires a structured data model, as with the vocabulary overview solutions in Section .

4.3. Data overview

- Various data visualization techniques have been proposed for Linked Data, but many of them only work with small data collections. E.g., spring embedding visualizations result in unwieldy visuals when they include more than a few thousand nodes.

4.4. Free text search

Datasets can be very large, so good search functions are required to let users find the needle in the haystack.

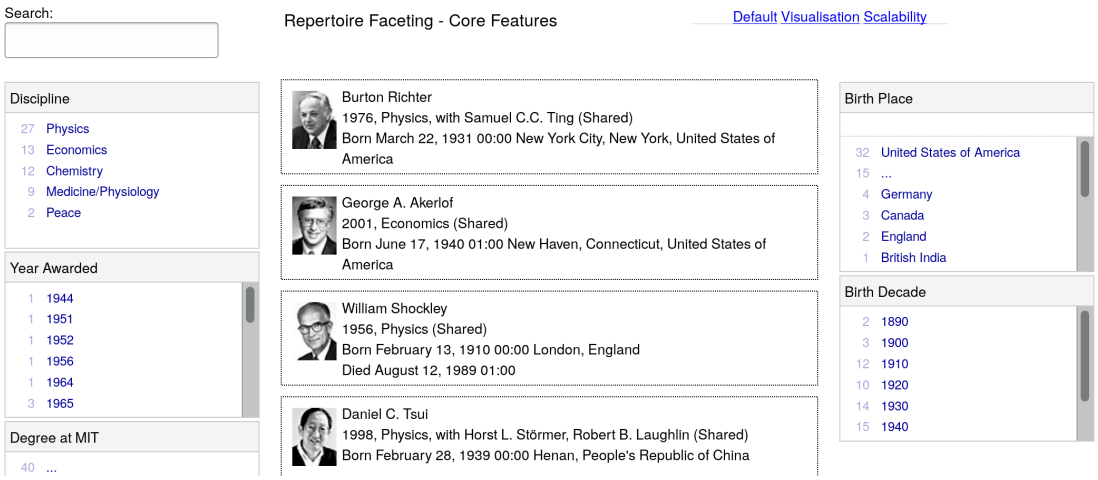

Figure 3: A faceted browser demo indexing MIT scientists (`http://hyperstudio.mit.edu/facets`).

5. Keep it simple (Lesson 1B)

Simplicity is notoriously difficult to achieve in Linked Data. The main reason for this is that simplicity is usually implemented by optimizing for a particular use case. Linked Data is about "the re-use of information in ways that are unforeseen by the publisher" [3]. However, this does not mean that existing approaches of Linked Geodata publishing cannot be simplified. We do observe that simplifications in Linked Geodata publishing are costly to implement. Simply changing something inside the User Interface is not enough: there are often conceptual reasons why certain things are difficult. We now give an example of a simplification we were able to implement.

5.1. Uniform & simple geodata representation

One of the ways in which geospatial data handling can be simplified is by enforcing a uniform representation format. The representation format must be generic enough to allow all representations that occur in the source datasets to be converted to it. At the same time, the uniform representation must allows data to be queried in a simple way. The following pattern is commonly used in our source datasets:

```
entity:x geosparql:defaultGeometry _:1                .
_:1      geosparql:asWKT           "POLYGON((...))"   ;
         rdf:type                  geosparql:Geometry .
```

The above three statements can be rewritten to a single statement conveying the same meaning:

```
entity:x def:geometry "POLYGON((...))"^^def:polygon .
```

This version has the following benefits:

- No introduction of an unnecessary blank node term (`_:1`).
- More explicit type information about the kind of geometry (`def:point` i.o. `geosparql:Geometry`).

These changes bring about the following 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 want to show polygons but not points and lines. We achieve this with the following query:

```
?x def:geometry ?y FILTER (datatype(?y) = def:polygon)
```

We use Well-Known Text (WKT) as a uniform representation format, since it supports many different shapes (17) with a simple grammar[16] that is easy to implement. All other representations that appear in our source datasets can be converted to our single-triple representation. For instance, the following WGS84 representations

```
entity:y wgs84:lat  "51.35"^^xsd:float ;
         wgs84:long "5.46"^^xsd:float  .
```

is converted to

```
entity:y def:geometry "POINT(5.46 51.35)"^^def:polygon .
```

5.2. Who is allowed to do what (Lesson 1C)

All our data was published as Linked Open Data with no authentication. In general we notice that Linked Data is far more often *read* than *written*. Our demonstration system does allow authorized users to send SPARQL Update requests, but we have not advertised or used this functionality in the current research.

[16] See `http://svn.osgeo.org/postgis/trunk/doc/bnf-wkt.txt`

5.3. Communities speak different languages (Lesson 1D)

Because communities speak in different languages, geodata is currently published in various different dialects. Many of these dialects are not Linked data and do 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. We can distinguish the following issues with linking:

- Atomic terms are actually compound terms whose components and links are implicit. We solve this with *value unpacking* (Section 5.4).

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).

5.4. Value unpacking

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

```
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*. The idea behind the term is that domain experts have 'packed' meaning into encodings like `BU00030002`. In order to 'open up' this information to non-domain experts and machine processors, we have to 'unpack' the meaning again by using a grammar. For the above 'buurtcode' we have to construct the following grammar (written in Augmented Backus-Naur Form (ABNF) [5]):

```
buurt          := "BU" gemeente-code wijk-code buurt-code
buurt-code     := DIGIT DIGIT
gemeente       := "GM" gemeente-code
gemeente-code  := DIGIT DIGIT DIGIT DIGIT
```

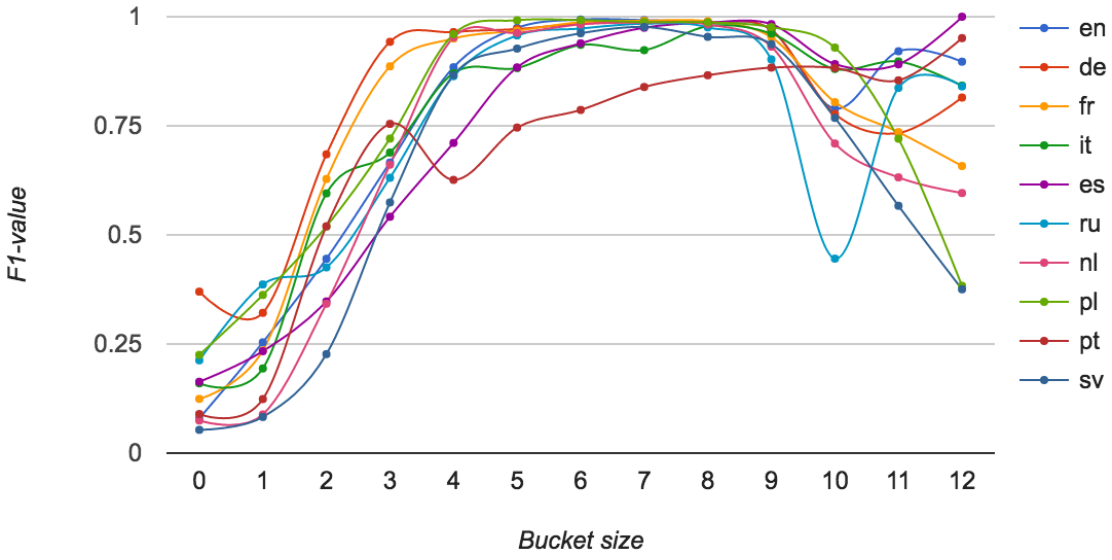

Figure 4: Accuracy of natural language detection in RDF strings for the 10 most frequent languages. Strings of similar length are combined in buckets. The size of buckets increases logarithmically. Figure taken from [2].

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

When loading the data we have to tell the machine to *parse* values that appear in certain locations (either columns, tags or keys) by using the above grammar. The machine is able to automatically construct the encoded entities and relations for all conforming values. The high cost of value unpacking is caused by the fact that grammars are domain-dependent and by the fact that some grammars are more complex.

5.5. 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 fully interpret it.

For instance, very many values are encoded in human languages. In order to do anything useful with these values we must at least know the language to which 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 4

The main benefit of running natural language detection on the current data collection is

to distinguish between Dutch and English strings.

5.6. Value combining

Sometimes multiple values can be combined into one aggregated value. This is specifically the case for datatypes. For example events are often described with a day, month and/or year column in the source day:

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

In order for dates to be comparable with one another in RDF they have to be converted to values of XML Schema Datatypes 1.1 [10]. Month names can be converted using a simple, one-to-one mapping. After that values can be converted to datatypes. The following

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

would be combined into

```
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 all be compared with 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 [9].

Value combining is only a good idea if the act of combining is lossless, i.e., no information is lost in the process. For instance, replacing the `foaf:givenName` and `foaf:familyName` properties with the combined `foaf:name` property is not an example of value combining, because it loses the cutoff point between the given and the family name.

5.7. Remove erroneous, empty and null values

TODO

6. Help search engines discover you (Lesson 2)

TODO

7. Show search engines the way with XML sitemap (Lesson 2A)

TODO

8. Link everything with everything (Lesson 2B)

TODO

9. 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. This is easy when the source data has a key. If this is not the case then

While enriching the data (Section 5.3) 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.

10. Make use of structure (Lesson 2D)

TODO

Format	Media Type
HTML 5	`text/html`
JSON	`application/json`
JSON-LD 1.0	`application/ld+json`
N-Quads 1.1	`application/n-quads`
N-Triples 1.1	`application/n-triples`
SPARQL 1.1 JSON	`application/sparql-results+json`
SPARQL 1.1 TSV	`text/tab-separated-values`
SPARQL 1.1 XML	`application/sparql-results+xml`

Table 1: Overview of the Media Types supported by the demonstrator.

11. Deal with multiple developers & devices (Lesson 3)

TODO

12. Serve results in different flavors (Lesson 3A)

Section 3 explains how exposing multiple result set formats can increase the number of users that can interact with a web service. In this section we explain how a user or client can request particular representation formats from a server.

Representation formats have been standardized as Media Types [8]. Table 1 shows the Media Types for the formats that are currently supported by the demonstrator.

According to the HTTP standard and HTTP best practices, different Media Types of the same resource have to be requested by a client using the `Accept` HTTP header (Section 12.1). In practice, some clients prefer to set the preferred Media Type in the request URL (Section 12.2).

12.1. Accept header

The `Accept` header [7] is the preferred way of implementing content negotiation. It allows multiple Media Types to be specified, including a precedence order over those Media Types by using weight parameters. It also allows Media Types to be matched hierarchically, due to distinguishing between a type, a subtype and parameters. For instance, the following HTTP header states that the client prefers JSON-LD over plain JSON and would accept any other format if neither JSON-LD nor plain JSON were

available.

```
Accept: application/json; q=0.5, application/ld+json, */*; q=0.1
```

If `*/*; q=0.1` is not included then our demonstrator returns a 406 status code ('Not Acceptable') reply to let the client know that none of the requested Media Types is supported.

12.2. URL-based content-negotiation

URL-based content-negotation allow clients to include their preferred Media Type in the query component of the request URL. For example, the following is intended to return Media Type `application/json`:

```
http://definities.geostandaarden.nl/concepten/imgeo/doc/begrip/Bak?format=json
```

While we are not opposed to implementing URL-based content-negotiation in our demonstrator, we do believe that the use of URL-based content negotiation should be discouraged because of the following reasons:

- `Accept` header-based content negotiation is more expressive than URL-based content negotiation. The former allows an ordered sequence of acceptable formats to be specified. For URL-based content negotiation to properly work, the `format` HTTP parameter should be processed in the same way as values of `Accept` headers.
- Because URL-based content negotiation is not standards-compliant, client and servers are tempted to use non-standardized Media Type names. For example, clients use the HTTP parameter `format=json`, but `json` is not a registered Media Type. Suppose we want to receive results in JSON-LD. Should we use the HTTP-parameter `format=jsonld`, `format=ld+json`, or something else?
- What happens when a client specifies the preferred Media Type in the `Accept` header and in the URL's query component? Because the latter approach is not standardized, the interaction between the two approaches is unknown. If the URL's query component overrules the `Accept` header value, the behavior is even violating HTTP standards.

13. Improve performance, reduce payload (Lesson 3B)

TODO

14. Do not copy data but use a proxy (Lesson 4)

TODO

15. 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 5.3). 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.

Lesson	Problem	Solution	Cost	Section
1	Web programmers know&like JSON	Return results in JSON-LD	Hours when used for smaller result sets and when programming in a supported language. Days in an unsupported language. Do not use at all for large data collections.	3.2
	Map programmers know&like GeoJSON	Return results in GeoJSON	Low cost due to a simple format. Cannot be read as LOD.	3.3
	GIS experts know&like OGC standards	Currently no LOD-compatible solution	Wait until OGC and W3C come with an integrated standard	3.4
1a	Vocabulary overview	Hierarchy visualization	Low cost when the data contains hierarchical information. High cost if the data model first has to be extended with hierarchical information.	4.2
		Domain/range visualization		
	Instance findability	Faceted browser		
		Text-based search		

15.1. 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] Tim Berners-lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006.

[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.

[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.

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 2: Aliases for commonly occurring IRI prefixes.

[7] R. Fielding and J. Reschke. Hypertext transfer protocol (HTTP/1.1): Semantics and content, June 2014.

[8] J. Klensin and T. Hansen. Media Type specifications and registration procedures, January 2013.

[9] 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.

[10] 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.

[11] Willem Robert Van Hage, Jan Wielemaker, and Guus Schreiber. The space package: Tight integration between space and semantics. *Transactions in GIS*, 14(2):131–146, 2010.

[12] 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.

[13] Jan Wielemaker, Wouter Beek, Michiel Hildebrand, and Jacco van Ossenbruggen. ClioPatria: A SWI-Prolog infrastructure for the Semantic Web. *Semantic Web Journal*, 2015.

A. Used aliases

Table 2 defines the RDF aliases that are used in this document.