

# Modern Ways of Spatial Data Publication

Wouter Beek ([wouter@triply.cc](mailto:wouter@triply.cc)) and Laurens Rietveld ([laurens@triply.cc](mailto:laurens@triply.cc))

Triply (<http://triply.cc>)

October 2, 2016

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.

Dataset	Format	Statements
Monumenten	Turtle	1,356,641
Beeldbank	XML	1,108,307
BGT Valkenswaard	JSON-LD	946,715
CBS 2015	CSV	613,717
Gemeentegeschiedenis	NDJSON	25,073

In addition to describing and quantifying the effort required to meet the constraints laid down in the existing lessons learned, we also describe the problems of publishing Linked Geodata in general. According to our assessment there are no off-the-shelf tools that allow Linked + geospatial data to be queried with acceptable performance.

## 1.1 Source datasets

The source datasets in Table 1.1 were used.

# 2 Storage

Do not use existing WoD products to implement a Web-based geospatial stack.

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<sup>1</sup>. 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:

<sup>1</sup>See <https://github.com/openlink/virtuoso-opensource>

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 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<sup>2</sup>. 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 from 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 become apparent.

## 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<sup>3</sup>, based on the PostgreSQL<sup>4</sup> database, was unanimously considered the most performant tool for handling geospatial data. Other tool that was mentioned was qGIS<sup>5</sup>, an Open Source GIS. Both tools use the backend library GEOS<sup>6</sup>.

---

<sup>2</sup>See <http://lucene.apache.org/solr/>

<sup>3</sup>See <http://postgis.net/>

<sup>4</sup>See <https://www.postgresql.org/>

<sup>5</sup>See <http://qgis.org/en/site/>

<sup>6</sup>See <https://trac.osgeo.org/geos>

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. This is the triple store that we used in this project.

### 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:

- 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<sup>7</sup> 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.

---

<sup>7</sup>For more information, see the RDF JavaScript Libraries Group (<https://www.w3.org/community/rdfjs/>).

## 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 marry 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 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<sup>8</sup>, states on its web site that “JSON-LD is not supported - too complex to implement”.

Virtuoso does not support loading JSON-LD files<sup>9</sup> 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<sup>10</sup>.

## 3.3 GeoJSON

Another format we expose is GeoJSON. The GeoJSON format is not yet fully standardized<sup>11</sup> 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

---

<sup>8</sup>See <http://librdf.org/raptor/>

<sup>9</sup>See <https://github.com/openlink/virtuoso-opensource/issues/478>

<sup>10</sup>See [https://github.com/wouterbeek/plRdf/blob/master/prolog/jsonld/jsonld\\_build.pl](https://github.com/wouterbeek/plRdf/blob/master/prolog/jsonld/jsonld_build.pl)

<sup>11</sup>GeoJSON is a proposed RFC standard as of August 2016.

Leaflet<sup>12</sup>. 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’<sup>13</sup>, 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). This point will be addressed in JSON-LD 1.1<sup>14</sup>.

### 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<sup>15</sup>. 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.

---

<sup>12</sup>See <http://leafletjs.com>

<sup>13</sup>See <https://github.com/geojson/geojson-ld>

<sup>14</sup>See <https://github.com/json-ld/json-ld.org/issues/397>

<sup>15</sup>See [https://www.w3.org/2015/spatial/wiki/Main\\_Page](https://www.w3.org/2015/spatial/wiki/Main_Page)

## 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 RCE monument dataset 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.

**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. 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 (Section 1.1) 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.



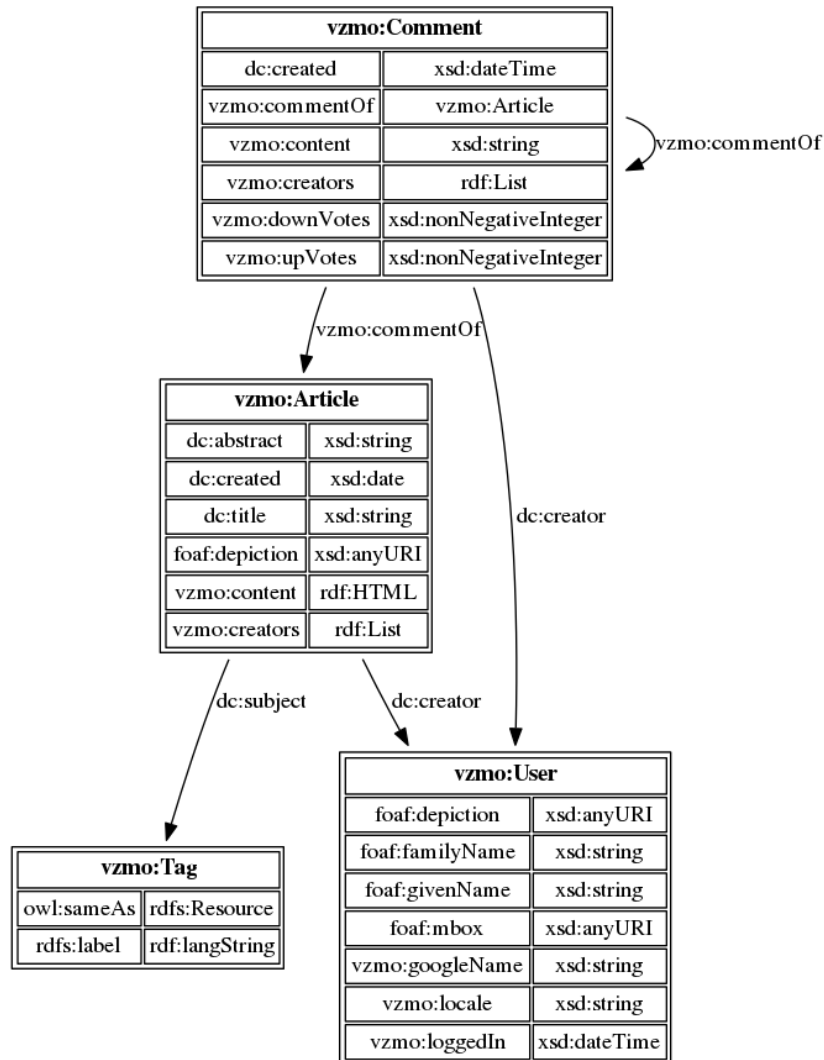


Figure 1: Example of a vocabulary visualization with 4 classes, 5 object properties, and 20 value properties.

### 3 search results for search string “Hofnar”:







Subject	Predicate	Object	Graph
<a href="#">nsid:strike/14092</a> 	<a href="#">nsdef:company</a> 	Hofnar	<a href="#">Stakingen/Data</a> 
<a href="#">http://data.culture...</a> 	<a href="#">dct:description</a> 	Inleiding WINKELHUIS van de N.V. Toebackslijterij "d'Hollandse Damper", 1919-1920, in rationalistische stijl. Gesitueerd op de hoek van Munstraat, Kleine Staat en Grote Staat. Gebouwd naar een ontwerp van de Maastrichtse architect V. Marres, in opdracht van bovengenoemde N.V. Omschrijving Winkelhuis op rechthoekige plattegrond van slechts 20 vierkante meter, beeldbepalend gesitueerd in de binnenstad. Op deze plattegrond staan eensouterrain, vier bouwlagen plus een zolderverdieping onder een zadeldak met twee insteekkappen, gedekt door leien. De eerste bouwlaag wordt vrijwel	<a href="#">Monumenten/Data</a> 

Figure 2: Results for the search “Hofnar”.

## 4.2 Instance findability

Datasets can be very large, so good search functions are required to let users find the needle in the haystack. 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 (IR) 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 2 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.<sup>16</sup>

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

<sup>16</sup>An example of an ElasticSearch deployment over Linked Data is our LOD Search () endpoint that indexes 4.3 billion literals.

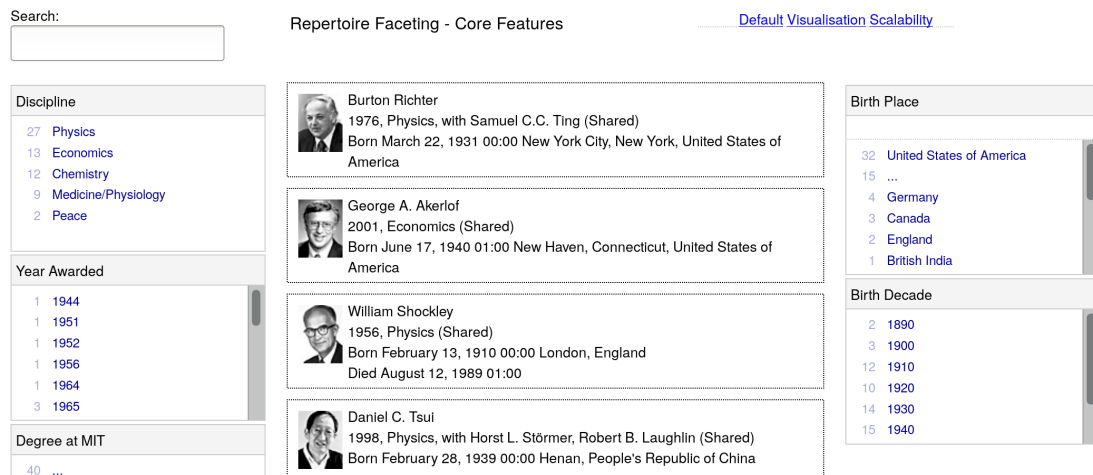


Figure 3: A faceted browser demo indexing MIT scientists (<http://hyperstudio.mit.edu/facets>).

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 3.

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

There are various data visualization techniques that have been proposed for generating overviews of Linked Data instances, but many of them only work with very small data collections. E.g., spring embedding visualizations result in unwieldy visuals when they include more than a few thousand nodes.

## 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. According to Tim Berners-Lee 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 particular 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 allow data to be queried in a simple way. The following pattern is commonly used in our source datasets (e.g., BGT):

```
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 Linked Data Fragment 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 display polygon regions, 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 shape types (17) with a simple grammar<sup>17</sup> 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

<sup>17</sup>See <http://svn.osgeo.org/postgis/trunk/doc/bnf-wkt.txt>

```
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 allows authorized users to perform SPARQL Update requests, but we have not advertised or used this functionality in this project.

## 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. The source datasets that we converted were serialized in XML, CSV and JSON.

Because existing tools do not stream data but load everything into memory we wrote our own data transformation scripts from XML/CSV/JSON to RDF. Only with a streamed approach is it possible to transform large datasets. Other source formats that were given to us were too complicated to transform within reasonable time. For instance dBase and Microsoft Access datasets are binary formats that depend on non-free software that is not commonly available in a web server setting.

## 6 Link everything with everything (Lesson 2B)

In our source data (Section 1.1), the municipality of Appingedam is denoted by multiple values such as ‘Appingedam’ and ‘GM0003’ (gemeentecode). At the same time, 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). The same is true for other value types, e.g., ‘1903’ may denote a year in the Gregorian calendar or the length of a road in meters. The challenge of instance term linking is to link

all and only terms that denote the same resource. We distinguish between the following two strategies towards linking:

**Heuristic** This form of linking relies on structural and/or string similarities between instance terms. Heuristic linking is risky because the optimal edit distance (both structure- and string-based) has to be determined in an ad-hoc way and because of ambiguity (where structurally and/or string-wise identical instance terms denote different resources). To mitigate the risk of heuristic linking, human curation of the results is necessary. This requires a per-instance term amount of effort.

**Declarative** This form of linking relies on domain-specific codes that are interpreted according to a known grammar. By using this grammar, equivalent instance terms can be identified across datasets. Equivalence is defined in terms of the result of parsing strings with the aforementioned grammar. While the grammar must be made on a per-domain basis, linking is accurate and can be performed automatically.

In this project we did not use heuristic linking because it required too much effort to filter out false positives given our source data (Section 1.1). We applied the declarative linking approach on the domain-specific grammars that appear in the source data. When we apply declarative grammars, we find out that some atomic terms in the source data are actually compound terms whose components and links are implicit. This means that in order to add automatic links we first have to make more of the structure explicit through value unpacking (Section 8.1).

After value unpacking, we automatically link the source datasets at different levels of abstraction. E.g., the “monumenten” and “gemeentegeschiedenis” datasets are linked to each other and to the “CBS” dataset at the municipality level.

## 7 Persistent IRIs (Lesson 2C)

For the source datasets (Section 1.1) that were already Linked Data we did not change the IRIs. Since most source datasets were not Linked Data we had to mint new IRIs. For this we follow the Dutch National URI Strategy [?].

An IRI consists of the following components

**Scheme** Traditionally, the de facto scheme for almost all Linked Data IRIs has been `http`. With the uptake in `https` adoption it may be wise to use HTTPS IRIs in the near future.

**Authority** This consists of four components, three of which (‘user’, ‘password’ and ‘port’) are not allowed to occur in resource-denoting IRIs. The fourth component, ‘host’,

identifies the server from which the denoted Linked Data resources are hosted. Since our source datasets are published and maintained by different organizations, we use the temporary host name `geonovum.triply.cc` from which we can host the converted resources. In the end, the Linked Data versions of the respective source datasets should be published by their original owners, and the host names of all IRIs will have to be updated accordingly.

**Path** According to the Dutch National URI Strategy the structure of this component depends on whether the IRI denotes a schema term (i.e., a class or property, sometimes called ‘TBox’) or an instance term (sometimes called ‘ABox’):

**Schema path** This is always `/def`.

**Instance path** This is `/id/<TYPE>/<NAME>`, where `<TYPE>` is the name of the primary type to which the instance belongs, and `<NAME>` is the instance name.

**Query** Is not allowed to appear in a resource-denoting IRI.

**Fragment** For schema IRIs (see the path component) this consists of the name of the class or property.

An example of a schema IRI we minted is `http://geonovum.triply.cc/def#validUntil` or `nsdef:validUntil` in abbreviated prefix notation. An example of an instance IRI we minted is `http://geonovum.triply.cc/id/gemeente/0003` or `nsid:gemeente/0003` in abbreviated prefix notation.

The Dutch National URI Strategy gives clear guidance towards minting new IRIs. In our experience, the most costly aspects of minting new IRIs are the `<TYPE>` and `<NAME>` parts of the path component of instance IRIs:

**<TYPE>** This is ideally the `rdfs:label` value of the primary class of the instance. If there are multiple classes, then one has to be manually chosen. If there are no classes, very common for our source datasets most of which were not Linked Data, these first have to be added to the data.

**<NAME>** If the source data contains a key relation, then the values of this key relation can be used as names. If there is no such key relation in the data then names have to be generated arbitrarily. For this we use UUIDs.

## 8 Make use of structure (Lesson 2D)

The use of structure is important to expose more aspects of meaning, as present in the source data, to external services. E.g., the string `BU00030002` denotes a specific neighborhood, but the fact that this neighborhood is contained in a municipality is opaque. There are many ways in which implicit structure can be made explicit and new structure can be added. Since of our source data is not Linked Data, we focused on repeatable/automated data enrichment steps that can be applied to source data that is

newly transformed to RDF: value unpacking (Section 8.1), value combining (Section 8.2) and the removal of erroneous, empty or null values (Section 8.3). There are many other data (re)structuring operations that we did not perform. The transformations we did perform provide the necessary scaffolding for future data transformations.

## 8.1 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
wijk            := "WK" gemeente-code wijk-code(Wijk).
wijk-code       := DIGIT DIGIT

```

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



## 8.2 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.

## 8.3 Removal of erroneous, empty or null values

Null values are often domain-dependent. For instance, in CBS data the value ‘-99999999’ is used to denote unknown values. Errors in the data can be found by performing a statistical outlier test. Not all outliers are errors, so the results of an outlier test have to be manually checked as well. For the CBS dataset we found the value ‘-99999999.0’ as an outlier, which is probably a typo of the domain-specific null value.

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.

## 9 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 9.1). In practice, some clients prefer to set the preferred Media Type in the request URL (Section 9.2).

### 9.1 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.

## 9.2 URL-based content-negotiation

URL-based content-negotiation 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.

## 10 Improve performance, reduce payload (Lesson 3B)

**HTTP compression** Compression of HTTP reply bodies is easy to implement by 'upgrading' the output stream to which the reply is written to Gzip/Deflate. Every HTTP server supports this.

**Reduce precision** The required accuracy of geometry shapes is application dependent: some applications require maximum detail (e.g., centimeters) while others only need to draw a rough map (e.g., tens of meters). For this reason we implemented an IRI query

component called `frac` that sets the maximum length of the fractional part of longitudes and latitude values.

A problem with this approach is that we do not know the precision of longitude and latitude values. Since many systems use floating-point numbers to represent longitude and latitude values, it is generally unclear which part of the fractional part is truthful and which part is a technical detail. This problem can easily be solved by using rational numbers to represent longitude and latitude values. Rational numbers with decimal denominator allow the precision to be expressed by the numerator. Unfortunately, since our source datasets all use floating point numbers and therefore remove precision information, we were unable to use rational number notation to express precision.

A second approach to reducing the precision of geospatial shapes is to reduce the number of points that are used to represent shape outlines. This requires relatively complicated algorithms like Ramer–Douglas–Peucker or Visvalingam–Whyatt to be implemented. We did not implement this form of precision reduction because these algorithms are currently not available in the language in which our triple store is written. A library that does support these algorithms is the JavaScript library TopoJSON<sup>18</sup>.

**Filtering** Geospatial support in SotA triple stores returns results in a bounding box. This is similar to the way in which triples stores return SPARQL result sets in chunks. For many real-world applications this way of returning results is impractical: there may be too many results within a given bounding box. At the same time, results are often more/less valuable to the user based on an application-specific criterion such as their proximity to the user.

Our triple store does not suffer from these deficiencies. It returns results in an anytime stream, i.e., on a one-by-one basis. This means that the client application needs to perform far less filtering efforts because the server returns results in a sensible order.

## 11 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

---

<sup>18</sup>See <https://github.com/mbstock/topojson>

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.

As Table 2 shows, the costs of setting up a web service for Linked Geodata are still prohibitive. In practice, some of the costs can be reduced by taking the following ‘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 by existing triple stores and are faster to compute. The downside to this is that much of the geospatial information is lost and very many geospatial queries (‘contains’, ‘intersects’) cannot be performed at all.

**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).

Lesson	Topic	Task	Cost	Section
1	Serve different result set flavors	Serve JSON-LD to web programmers	JSON-LD support costs hours to implement when used for small result sets and when programming in one of the supported languages. It takes more effort in an unsupported language: 12 hours. Do not use this approach at all for large data collections.	3.2
		Serve GeoJSON to web programmers	Low cost due to a simple format: 2 hours. This format cannot be read back as LOD.	
		GIS experts know&like OGC standards	Currently no LOD-compatible solutions. Wait until OGC and W3C come with an integrated standard.	
1A	Vocabulary overview	Hierarchy visualization	Low cost when the data contains hierarchical information; high cost otherwise.	4.2
		Domain/range visualization	Low cost when the data contains domain/range information; high cost otherwise.	4.2
		Faceted browser	Low cost with property and class information; high cost otherwise.	4.2
1B	Keep it simple	Text-based search	Simple sub-string search based on a SPARQL endpoint: 10 hours. Complex text-based search with approximate matching using ElasticSearch: 40 hours.	4.2
		Uniform geo representation	Convert all shapes to non-nested WKT: 28 hours	5.1
1D	Different languages	Source data is not Linked Data	Converting data from open text-based formats is cheap: 5-10 hours per format. Closed binary formats are costly to convert and require specific tools or programming languages to convert.	5.3
2B	Link everything	Heuristic linking	This form of linking is expensive, because links have to be checked for false positives. We did not perform heuristic linking in this project.	8.1

		Declarative linking	We used our implementation for lesson 2D ('value unpacking') to perform this form of linking: 8 hours	8.1
		Remove errors	Outlier detection is easy to perform with a statistical framework like R. The effort needed to fix errors is linear in the number of outliers that are discovered. We have spend 4 hours on outlier detection and error removal.	8.2
2C	Persistent IRIs	Following Dutch National URI Strategy	Easy when the data contains (1) a key relation and (2) a primary class with an <code>rdfs:label</code> property. When these are not available they have to be added to the data which can be costly: 16 hours.	7
2D	Use structure	Value unpacking	Grammar construction on a per-domain basis. For CBS buurtcode grammar construction and applying it to value unpacking: 10 hours.	8.1
		Value combining	Applied to all date/time values: 12 hours.	8.2
		Removal of erroneous, empty or null values	Effort needed includes running outlier detection software, validating each outlier by hand, and extending the data loading script with removal operations for outliers that are considered erroneous: 14 hours.	8.3
3A	Request different flavors	Use of HTTP Accept header	Not very well supported by most web servers, but easy to implement by REST libraries or by hand: 10 hours.	9.1
		URL-based content negotiation	Trivial: 1 hour	9.2
3B	Reduce payload	HTTP compression	Supported by every HTTP server: 1 hour	10
		Reduce precision	Allowing the maximum length of the fractional part to be set requires low cost: 3 hours. Implementing reduction in the number of outline points is low cost if a support library is available. Not implemented in this project.	10

Table 2: Table enumerating the tasks involved in implementing the GeoNovum lessons learned, quantifying the effort where possible.

## 12 Conclusion

We conclude that it is not yet possible to go from a loose collection of (geo)data sources to a usable web service for web programmers within two months. There are too many missing pieces and the cost of data conversion and data transformation is simply too high. In addition, there are no off-the-self tools that allow Linked Geodata to be queried with acceptable performance. Our exploration shows that it is possible to perform query optimization over Linked Data and geospatial objects, but within 2 month we were unable to implement full GeoSPARQL support. A full implementation would require further development and investment.

Even though the problems in terms of data storage, conversion and transformation are severe, we do believe that the GeoNovum lessons learned give the direction that is needed in order to improve existing tools so that it will become easier to publish Linked Geodata in the future.

## 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.



- [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.