Transforming Geospatial RDF Data into GeoSPARQL-Compliant Data: A Case of Traffic Data

Milos Jovanovik
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, North Macedonia
milos.jovanovik@finki.ukim.mk

Mirko Spasić
Faculty of Mathematics
University of Belgrade
Belgrade, Serbia
mirko@matf.bg.ac.rs

***Abstract*—Geospatial RDF datasets have a tendency to use latitude and longitude properties to denote the geographic location of the entities described within them. On the other hand, geographic information systems prefer the use of WKT and GML geometries when working with geospatial data. In this paper, we present a process of RDF data transformation which produces a GeoSPARQL-compliant dataset, using an RDF geospatial dataset with traffic data as a starting point. The traffic is comprised of vehicle traces, which consist of numerous points with specific latitude and longitude values. With our transformations, we enable querying of the dataset with GeoSPARQL extensions, which can be used to feed a GIS solution.**

***Index Terms*—GeoSPARQL, Geospatial Data, Data Transformation, GIS, RDF, SPARQL, Linked Data**

I. Introduction

The exploration gain of semantic technology for spatial data management becomes more and more evident in many different domains. Specific taxonomies and ontologies have been used on thematic web portals for representing a variety of categories with geospatial features. With the increasing popularity of this technology within the spatial domain, the need for standardization is growing. Initial efforts include the Basic Geo Vocabulary[1] by the World Wide Web Consortium (W3C), which provides a namespace for representing latitude, longitude and other information about geospatial entities using WGS84 as a standard of the coordinate system. Later achievements comprise GeoRSS[2], GeoOWL[3], NeoGeo Geometry Ontology[4], GeoJSON[5], GeoRDF[6], etc. Finally, GeoSPARQL has emerged as a promising standard from W3C for geospatial RDF. It supports both representation and querying of geospatial data on the Semantic Web and defines a new vocabulary. It suggests a concrete ontology for representing features and geometries in RDF as Well Known Text (WKT) or Geography Markup Language (GML) literals. GeoSPARQL defines a core set of classes, properties and data types that can be used to construct query patterns in an extension of SPARQL. Its main aim is to ensure a consistent representation of geospatial semantic data across the Web, thus allowing both vendors and users to achieve uniform access to geospatial RDF data.

Although the standard is sufficiently mature, now being 7 years old, the data providers are not familiar enough with it, or more often they take the path of least resistance and publish their data in their own format or use custom ontologies. In these scenarios, an additional effort is needed in order to understand their data, and use it within an application. This way, cross-domain data mixing and fusion is very complicated and almost impossible. A better approach is to develop a means for transforming the dataset in question into a GeoSPARQL-compliant one, ready for use by a large number of stakeholders who already have knowledge of the standard. That was the main goal of our transformations described in this paper.

II. Related Work

Transforming data from one format to the other – whether related to a simple conversion between different file formats, or more complicated mappings from relational data to linked data, or even generation of datasets following different standards – is a fundamental aspect of most data integration and data management tasks. It can vary from very simple or very complex, depending on the required changes to the data and their model. This is an area that attracts the attention of many data scientists for very practical reasons. Integrating data from heterogeneous sources has led to the development of Extract-Transform-Load (ETL) systems and methodologies, as a means of addressing modern interoperability challenges.

One group of these are general tools for converting relational data from traditional DBMSs to the RDF data model, such as Triplify [1], D2R Server[7], or Virtuoso RDFizer Middleware (Sponger)[8]. On the other hand, there are libraries

[1] http://www.w3.org/2003/01/geo/
[2] http://www.georss.org/
[3] https://www.w3.org/2005/Incubator/geo/XGR-geo-20071023/W3C_XGR_Geo_files/geo_2007.owl
[4] http://geovocab.org/
[5] http://geojson.org/
[6] https://www.w3.org/wiki/GeoRDF
[7] http://d2rq.org/d2r-server
[8] http://docs.openlinksw.com/virtuoso/virtuososponger.html

and tools which deal with geospatial data, but without any RDF support: GDAL/OGR[9], which is a translator library for raster and vector geospatial data formats, GeoKettle[10], which is a spatial ETL tool dedicated to the integration of different spatial data sources for building and updating geospatial data warehouses, etc. It is worth to mention geometry2rdf[11], as a library for generating RDF from resources with geometrical information, but its RDF model is not compliant with the GeoSPARQL standard.

The largest open geospatial dataset is Linked Geo Data, a spatial knowledge base which has been derived from Open Street Map[12]. The authors of [2] elaborate on how the collaboratively collected OSM data can be interactively transformed and represented adhering to the RDF data model, building a rich integrated and interlinked geographic dataset for the Semantic Web. Their approach is to store the mapping information in a relational database (PostGIS) together with the OSM data, and use the SPARQL-to-SQL rewriter Sparqlify[13] to generate RDF. The mapping uses the GeoSPARQL vocabulary to represent the data.

TripleGeo [3] is an open-source tool that can produce geospatial datasets in compliance with GeoSPAQL standard, based on different inputs. It is an ETL utility that can extract geospatial features from various sources and transform them into triples for subsequent loading into RDF stores. TripleGeo can directly access both geometric representations and thematic attributes either from standard geographic formats or widely used DBMSs. It can also reproject input geometries on-the-fly into a different coordinate reference system, before exporting the resulting triples into a variety of notations.

III. Traffic Dataset

As part of our activities in the project "SAGE: Semantic Geospatial Analytics"[14] [4], we use traffic datasets generated by the synthetic data generator from TomTom [5]. The synthetic data generator can generate RDF datasets containing synthetic traces of vehicles on public roads, based on a specialized algorithm which mimics real-world traffic.

The datasets contain `Trace` entities, representing a one-day trace of a vehicle on a map, where each `Trace` consists of multiple `Point` entities. Each `Point` has a `latitude`, a `longitude`, a `timestamp` and a `Speed` entity, depicting a specific position in time and space of the vehicle. The `Speed` entity has a velocity `value` and `metric`.

The size of the generated dataset can be specified in advance, by specifying the number of traces we want it to contain. For the research presented in this manuscript, we generated a dataset which contains 1,000,000 traces of vehicles in and around the city of Leipzig, spanning from the beginning of year 2016 to the end of 2017. These `Trace` contain a total of over 126,000,000 `Point` entities. In total, the synthetic RDF dataset contains over 889,000,000 RDF triples.

[9]http://www.gdal.org/
[10]http://www.spatialytics.org/projects/geokettle/
[11]https://github.com/boricles/geometry2rdf
[12]https://www.openstreetmap.org/about
[13]http://aksw.org/Projects/Sparqlify.html
[14]http://sageproject.eu/

IV. GeoSPARQL Enhancements of the Dataset

The synthetic dataset uses a very basic ontology, depicting the classes and properties mentioned above. The produced dataset is in RDF, but it is not GeoSPARQL compliant, i.e. it does not contain `Feature` and `Geometry` instances, which have explicit well-known-text (WKT) geometries [6]. This means that while the dataset itself is a geospatial dataset, it cannot be directly used with GeoSPARQL and in applications which require explicit WKT values for the geospatial entities.

To improve this, and enable application interactions with the dataset with GeoSPARQL queries, we created general enhancements for the traffic datasets. With them, we produce GeoSPARQL-enabled datasets, suitable for the project, its use-cases and general compliance testing.

The GeoSPARQL enhancements are defined as SPARQL queries which can be executed over a TomTom synthetic traffic dataset which is loaded into an RDF triplestore [7]. The enhancements include `Trace` transformations, `Point` transformations, and some additional transformation necessary for improved use-cases.

A. Trace Transformations

The trace transformations have the purpose of creating a LineString geometry for each trace in the dataset. Since the dataset model is specific for the data generator we use to generate the datasets, we need a custom transformation which will select the latitude and longitude values of all points which comprise a given trace, and construct the corresponding WKT LineString geometry [7]. To achieve this, we use the following transformations for each trace:

- Each `Trace` is enhanced to represent a `geo:Feature` entity, which has a `geo:hasGeometry` relation with a geometry entity specified as a `sf:LineString` entity;
- The `sf:LineString` entity has a `geo:asWKT` relation to a `geo:wktLiteral` value, which represents the entire `Trace` as a GeoSPARQL `LINESTRING`.

The SPARQL INSERT query for the trace transformations adds new triples into the dataset graph, via the snippet below.

Trace Enhancements Snippet

```
INSERT {
    ?trace rdf:type geo:Feature .
    ?trace geo:hasGeometry ?traceGeomID .
    ?traceGeomID a sf:LineString .
    ?traceGeomID geo:asWKT ?wkt .
}
```

Here, `geo:Feature` and `sf:LineString` are classes defined by the GeoSPARQL standard [6]. On the other hand, `?traceGeomID` is constructed by a simple string concatenation between the original URI of the trace and a fixed suffix. The `?wkt` value is constructed by reading all points from the trace in a subquery, ordering them by their timestamp,

constructing a combined string of the latitude and longitude of each point, and then combining all concatenated latitudes and longitudes of the ordered points in a format which defines the complete trace as a LineString. An example WKT LineString value for a trace is shown below.

WKT LineString Example

```
LINESTRING(
    12.127818 51.284445,
    12.124343 51.282212,
    12.120798 51.280137,
    12.117422 51.277686,
    12.114321 51.275172,
    12.111059 51.272767,
    ...
    12.174391 51.328762)
```

B. Point Transformations

Similarly as with traces, we aimed to transform each point of each trace into a Point geometry [7]. Therefore, for each point we use the following transformations:

- Each `Point` is enhanced to represent a `geo:Feature` entity, which has a `geo:hasGeometry` relation with a geometry entity specified as a `sf:Point` entity;
- The `sf:Point` entity has a `geo:asWKT` relation to a `geo:wktLiteral` value, which represents the `Point` as a GeoSPARQL `POINT`.

Similarly as with the traces, the SPARQL INSERT query for the point transformations also adds new triples into the dataset graph, via the snippet below.

Point Enhancements Snippet

```
INSERT {
    ?point rdf:type geo:Feature .
    ?point geo:hasGeometry ?pointGeomID .
    ?pointGeomID a sf:Point .
    ?pointGeomID geo:asWKT ?wkt .
}
```

As with the traces, the `geo:Feature` and `sf:Point` classes used in the INSERT clause are classes defined by the GeoSPARQL standard, while `?pointGeomID` is constructed by a simple string concatenation between the original URI of the point and a fixed suffix. The `?wkt` value is constructed by reading and concatenating the latitude and longitude values of the point, in a format which defines a correct Point. An example WKT Point value is shown below.

WKT Point Example

```
POINT(12.127818 51.284445)
```

C. Additional Transformations

Aside from the trace and point transformations, we made additional transformations which are not directly related to GeoSPARQL, but enhance the use-cases [7]:

- Adding a numerical ID to each `Trace` entity, via a new property: `traces:numID`;
- Adding an explicit relation between a `Trace` and its start and end points, via new properties: `traces:hasStartPoint` and `traces:hasEndPoint`, respectively;
- Adding an explicit relation between a `Trace` and its calculated duration, in seconds, via a new property: `traces:hasDuration`.

After running all enhancements over our Leipzig traffic dataset, we ended up with over 1,532,000,000 RDF triples in total in the dataset. This dataset is available within a Virtuoso instance [8], made available as part of the SAGE project.

V. Using the GeoSPARQL-Compliant Dataset

In order to demonstrate the usability of the resulting GeoSPARQL-compliant dataset, we present several use-cases in the form of SPARQL queries, query results and their actual usage within a geographic information system. For brevity, all prefix definitions and FROM clauses of the SPARQL queries have been omitted from the examples, but can be seen in full detail on the GitHub page of the project [7]. All examples below use our enhanced synthetic traffic dataset for the city of Leipzig, described previously.

1) Query 1: Find all vehicle traces started within a specified time period and select their respective WKT `LINESTRING` values, to be drawn on the map. Additionally, select the duration of each such trace, and calculate the distance between the starting point and the ending point of the trace.

SPARQL Query 1

```
SELECT ?wkt ?date ?duration ?distance ?traceID
WHERE {
    ?trace a traces:Trace ;
        geo:hasGeometry ?traceGeom ;
        traces:numID ?traceID ;
        traces:hasDuration ?duration ;
        traces:hasStartPoint ?start ;
        traces:hasEndPoint ?end .
    ?traceGeom geo:asWKT ?wkt .
    ?start traces:hasTimestamp ?date .

    FILTER (?date >=
        "2017-05-03T06:00:00Z"^^xsd:dateTime
            && ?date <=
        "2017-05-03T23:45:00Z"^^xsd:dateTime)

    ?start geo:hasGeometry ?startGeom .
    ?startGeom geo:asWKT ?startWKT .
    ?end geo:hasGeometry ?endGeom .
    ?endGeom geo:asWKT ?endWKT .

    BIND(geof:distance(?startWKT,
                       ?endWKT,
                       units:meter) as ?distance)
} ORDER BY ?date
```

This query returns results for all traces of vehicles which started their route in the specified time period. A partial result

TABLE I
PARTIAL RESULTS FROM QUERY 1.

Trace as WKT	Date and Time	Duration (s)	Distance (m)	Trace ID
LINESTRING(12.353488 51.343931,12.352586 51.343744, ...	2017-05-03T08:37:59Z	1,070	16,581.50	422335
LINESTRING(12.610297 51.441528,12.612627 51.441695, ...	2017-05-03T08:38:11Z	360	1,459.58	952143
LINESTRING(12.491782 51.381629,12.491827 51.381999, ...	2017-05-03T08:38:19Z	1,740	19,425.20	675117
LINESTRING(12.207745 51.407249,12.207753 51.406779, ...	2017-05-03T08:38:21Z	590	7,281.98	350236
LINESTRING(12.489270 51.272567,12.491189 51.275444, ...	2017-05-03T08:38:41Z	1,650	12,941.70	941890
LINESTRING(12.315006 51.328223,12.315075 51.328165, ...	2017-05-03T08:38:43Z	546	4,021.36	955692
LINESTRING(12.409652 51.335220,12.408825 51.335428, ...	2017-05-03T08:38:52Z	1,155	14,875.00	430094
LINESTRING(12.528184 51.267206,12.528220 51.267318, ...	2017-05-03T08:38:57Z	505	1,786.52	771733
LINESTRING(12.362608 51.358667,12.362919 51.359149, ...	2017-05-03T08:38:58Z	480	3,876.72	517663
LINESTRING(12.127818 51.284445,12.124343 51.282212, ...	2017-05-03T08:38:58Z	1,000	5,892.86	957376

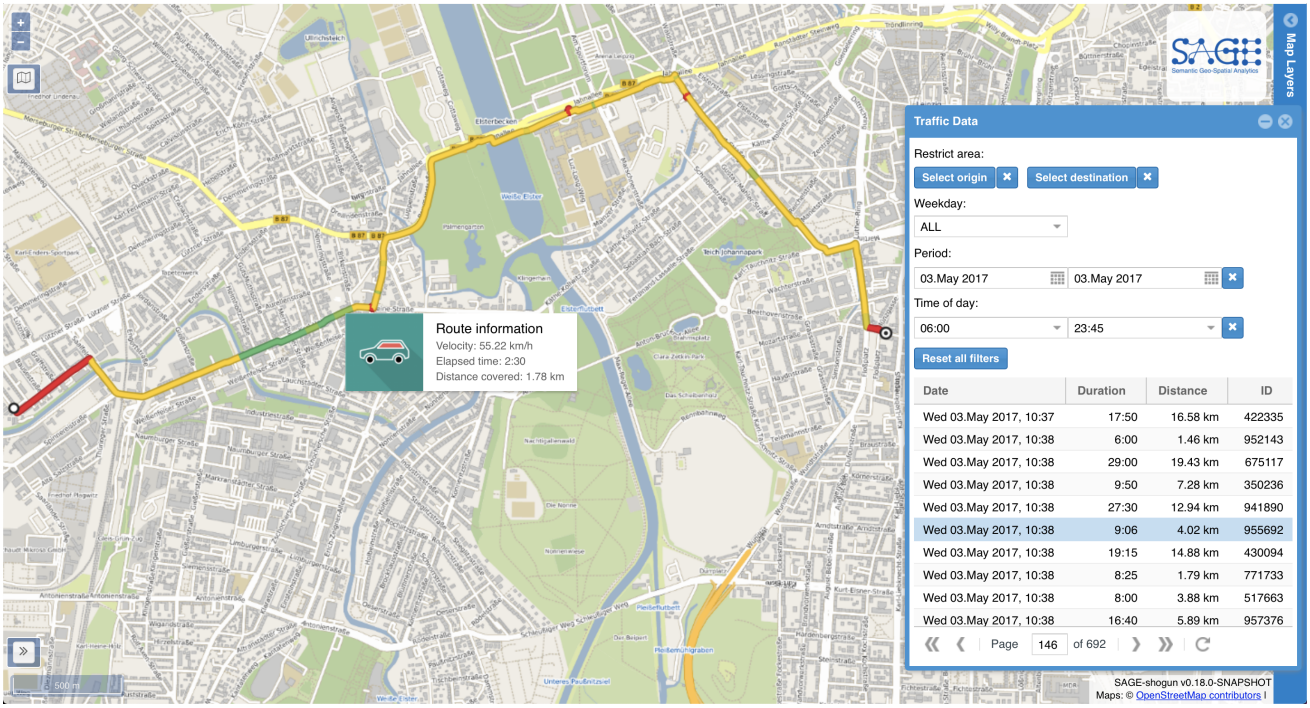

Fig. 1. Visual representation of the results of Query 1: mapping them within a GIS solution.

is shown in Table I, and the same traces are depicted on Figure 1 as part of a GIS solution which visualizes one of the selected WKT traces. The difference in the date-time values between the table and the figure is due to different time-zones: the dataset holds the date-time values in GMT time, as they are originally generated, while the GIS application transforms and displays them into CET time, for user-experience purposes. Note that the GIS application sends out an additional SPARQL query to the dataset to get the speed values of each point of the selected trace, the result of which is visible in Figure 1 via the trace coloring. For brevity, this additional query is omitted from the example.

2) Query 2: Find all vehicle traces which have a given map region as a destination. The query selects all traces which satisfy the constraints, gets their WKT `LINESTRING` values, their duration and calculates the distance between the starting point and the ending point of the trace.

SPARQL Query 2

```
SELECT ?wkt ?date ?duration ?distance ?traceID
WHERE {
    ?trace a traces:Trace ;
        geo:hasGeometry ?traceGeom ;
        traces:numID ?traceID ;
        traces:hasDuration ?duration ;
        traces:hasStartPoint ?start ;
        traces:hasEndPoint ?end .
    ?traceGeom geo:asWKT ?wkt .
    ?start traces:hasTimestamp ?date ;
        geo:hasGeometry ?startGeom .
    ?startGeom geo:asWKT ?startWKT .
    ?end geo:hasGeometry ?endGeom .
    ?endGeom geo:asWKT ?endWKT .

    FILTER(geof:sfContains(
      bif:ST_GeometryFromText("POLYGON((
        12.346821967457 51.34679532759,
        12.348366919850 51.34703657322,
```

TABLE II
PARTIAL RESULTS FROM QUERY 2.

Trace as WKT	Date and Time	Duration (s)	Distance (m)	Trace ID
LINESTRING(12.354888 51.349741,12.353951 51.349536, ...	2016-03-06T06:52:34Z	130	651.96	507008
LINESTRING(12.351120 51.345119,12.350997 51.344888, ...	2016-03-06T07:42:50Z	225	8.93	935397
LINESTRING(12.292396 51.413958,12.293035 51.413740, ...	2016-03-06T08:38:42Z	940	8,899.60	100679
LINESTRING(12.354429 51.348359,12.354407 51.348249, ...	2016-03-06T09:09:35Z	278	429.51	970485
LINESTRING(12.358288 51.360596,12.358529 51.360280, ...	2016-03-06T09:30:12Z	240	1,670.04	853118
LINESTRING(12.269029 51.305859,12.268899 51.306153, ...	2016-03-06T12:02:25Z	925	7,440.02	899824
LINESTRING(12.347528 51.370391,12.348132 51.371072, ...	2016-03-06T12:48:52Z	450	2.714.82	85904
LINESTRING(12.338960 51.331886,12.338704 51.331418, ...	2016-03-06T12:49:42Z	240	1,429.53	972873
LINESTRING(12.462456 51.370047,12.462360 51.370006, ...	2016-03-06T13:22:17Z	807	8,120.75	605680
LINESTRING(12.210354 51.399915,12.210412 51.399675, ...	2016-03-06T14:08:47Z	1,065	11,607.40	791482

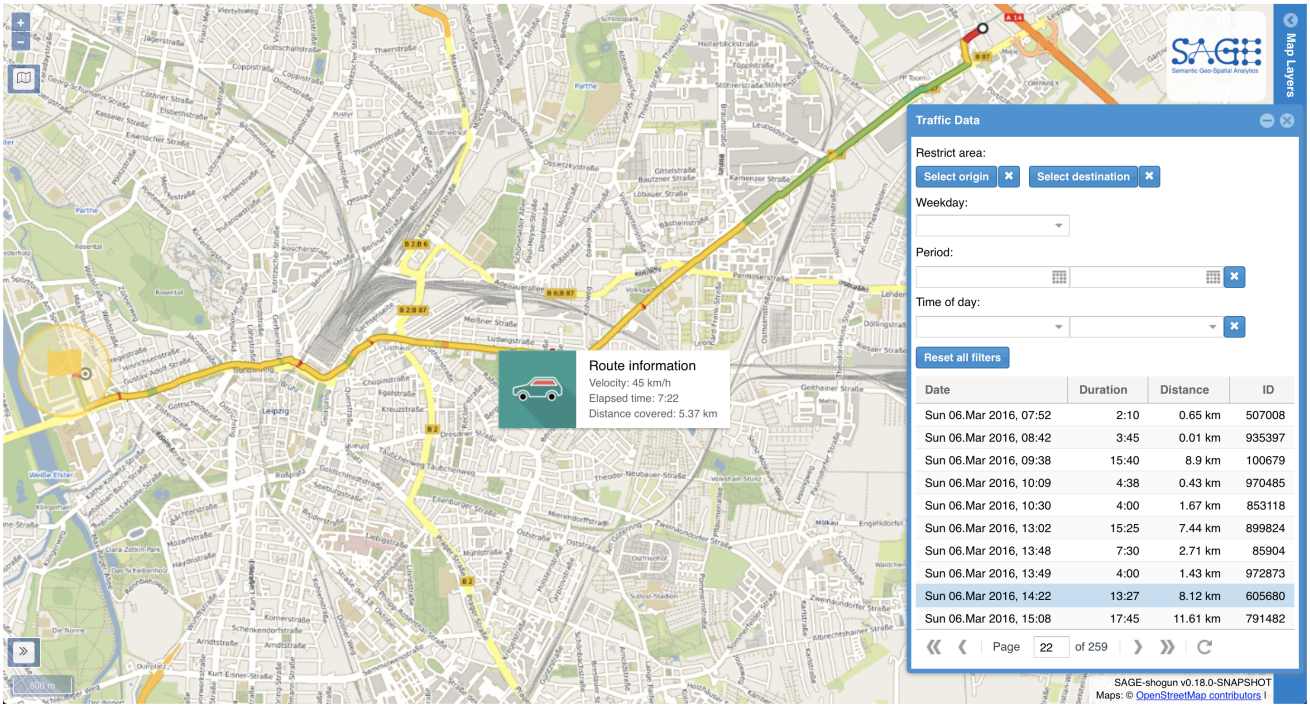

Fig. 2. Visual representation of the results of Query 2: mapping them within a GIS solution.

```
      12.350898925160 51.34706337821,
      12.352486792897 51.34655408069,
      12.353817168568 51.34574991518,
      12.354503814076 51.34449002752,
      ...
      12.346821967457 51.34679532759))"),
    ?endWKT))

  BIND(geof:distance(?startWKT,
                     ?endWKT,
                     units:meter) as ?distance)
} ORDER BY ?date
```

In this example we select all vehicle traces which end on the parking grounds around the main football stadium in Leipzig. Part of the results are presented in Table II and on Figure 2, along with one selected trace which is displayed in full detail. The map region selected as the destination filter for traces can be seen in the left part of the figure, as a yellow circle with a yellow flag in the middle.

It is important to note that both of these examples, and their SPARQL queries, would not be possible without the introduced transformations of the original dataset. Namely, the SPARQL functions `geof:distance` and `geof:sfContains`, just like all other GeoSPARQL functions, cannot work with arguments which are not valid WKT values, such as our LineStrings and Points.

Many other examples and use-cases are made available with the transformation of the latitude and longitude coordinates into comprehensive WKT literals, and with the additional enhancements we did as part of the work presented here. For instance, some of the other GeoSPARQL function can be used to find intersections between regions on a map, e.g. a street, and the entire trace, not just its start and end point. This would provide a way to select vehicle traces which traverse a specific location on a map, on a particular day, in a particular time.

The examples shown here have been integrated as features of the GIS solution presented in Figures 1 and 2, where it generates and issues GeoSPARQL-extended SPARQL queries to the dataset. The GeoSPARQL-compliant RDF results are then used by the application in a convenient manner.

VI. Conclusion

The transformation of a simple geospatial RDF dataset into a GeoSPARQL-compliant dataset brings the data quality to a significantly high level, where it can be used by GIS software which does not speak RDF. This is a very important step for both providers and users of geospatial data as it allows interoperability between the already existing toolset for dealing with geospatial data on one hand, and the RDF dataset providers on the other hand. We demonstrate this by transforming a synthetic RDF dataset in the traffic domain into a dataset which can be queried using the GeoSPARQL extensions of SPARQL, and can be visualized in a GIS application which draws WKT geometries on a map.

Acknowledgment

This work has been supported by Eurostars Project SAGE (GA no. E!10882)

References

[1] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, D. Aumueller. “Triplify: Light-weight Linked Data Publication from Relational Databases”, 18th International Conference on World Wide Web, pp. 621–630, April 2009.

[2] C. Stadler, J. Lehmann, K. Hffner, S. Auer, “LinkedGeoData: A Core for a Web of Spatial Open Data”, Semantic Web Journal, vol. 3, no. 4, pp. 333–354, 2012.

[3] K. Patroumpas, M. Alexakis, G. Giannopoulos, S. Athanasiou, “TripleGeo: An ETL Tool for Transforming Geospatial Data into RDF Triples”. EDBT/ICDT Workshops, pp. 275–278, 2014.

[4] M. Jovanovik, H. Williams, M. Spasić, “D4.1: Prototype Verifying Virtuoso GeoSPARQL and DE-9IM Compliance”, Project deliverable from Project SAGE, January 2019.

[5] K. Bösche, T. Sellam, H. Pirk, R. Beier, P. Mieth, S. Manegold, “Scalable Generation of Synthetic GPS Traces with Real-Life Data Characteristics.”, Technology Conference on Performance Evaluation and Benchmarking, pp. 140–155, August 2012.

[6] Open Geospatial Consortium, “OGC GeoSPARQL - A Geographic Query Language for RDF Data”. Open Geospatial Consortium, September 2012. https://www.opengeospatial.org/standards/geosparql, Document 11-052r4.

[7] M. Jovanovik, “Traffic Data to GeoSPARQL” GitHub Project Page: https://github.com/mjovanovik/TrafficData-to-GeoSPARQL, Retrieved on 12 April 2019.

[8] O. Erling, “Virtuoso, a Hybrid RDBMS/Graph Column Store”, IEEE Data Eng. Bull., vol. 35, no. 1, pp. 3–8, 2012.