C-GeoSPARQL: streaming GeoSPARQL support on C-SPARQL

Alexander Dejonghe, Femke Ongenae, Stijn Verstichel and Filip De Turck

IDLab - imec - Ghent University
Technologiepark 15, 9052 Zwijnaarde, Belgium
Alexander.Dejonghe@UGent.be

Abstract. The port of a city is a very dynamic environment that houses lots of companies. Ships come and go, and goods are always on the move. Information integration in geographic information systems is of great importance for tracking purposes, and for proactive and reactive incident handling by the port operators. Linked data and semantic web technologies can be of benefit for the integration of both streaming data such as location data about ships, trains and containers, with static data about companies, their activities and storage sites. With current RDF stream processors it is possible to execute SPARQL queries over RDF data streams taking into account static background data. However, none of them is capable of handling GeoSPARQL queries. GeoSPARQL is an extension to the SPARQL query language for processing geospatial data. To address this challenge we extended the RSP engine C-SPARQL with GeoSPARQL support, making it possible to query geospatial data streams.

Keywords: Stream processing, GeoSPARQL, C-SPARQL, RSP, Parliament, geospatial

1 Introduction

Geospatial information is of great importance in a large number of application domains. Everything that happens, happens at a certain location. With the growth of the Internet of Things (IoT), it becomes more easy to track things and monitor actions happening somewhere at a certain place in the world. IoT sensors and devices constantly generate streams of data that describe their location, state, and the state of the context or environment they are active in.

The port of a city for example, is a very dynamic environment where lots of activities take place in which many actors are involved. It houses lots of companies, and goods are always on the move on the water or on land. Information integration in geographic information systems is of great importance for tracking purposes, and for proactive and reactive incident handling by the port operators. Linked data and semantic web technologies can be of benefit for the integration of streaming data such as location data about ships, trains, containers and the weather, with static data about infrastructure, companies, their activities, storage sites and others.

With the current RDF stream processors (RSPs) it is possible to execute SPARQL queries over RDF data streams taking into account static semantic background data [10]. However, none of these RSPs are capable of handling GeoSPARQL queries. GeoSPARQL is an extension to the SPARQL query language for processing geospatial RDF data [2]. This paper explains how the RSP engine C-SPARQL [1] can be extended with GeoSPARQL support, making it possible to query geospatial RDF data streams. Moreover, it shows how streaming geographic RDF data, consisting of ship observations, and static geographic information about the port and its environment can be queried with this C-SPARQL extension.

The remainder of this paper is organized as follows. Section 2 represents the related work. It is followed by the section C-GeoSPARQL that explains the C-SPARQL extension to support GeoSPARQL. Section 4 presents a use case and test environment regarding the port of the city of Ghent.

2 Related Work

Semantic Web technologies like the data model RDF, the ontology language OWL, and the RDF query language SPARQL allow to represent, integrate, query and reason on heterogeneous data. However, these technologies were developed for static or slow changing data sources. On the other end of the spectrum Data Stream Management Systems (DSMS) and Complex Event Processing (CEP) systems allow to query homogeneous streaming data structured according to a fixed data model. They are not able to deal with heterogeneous data sources and lack support for the integration of domain knowledge. To bridge this gap, stream reasoning has emerged as a challenging research area that focuses on the adoption of Semantic Web technologies for streaming data [4]. As a result of the stream reasoning research conducted in the past years, different prototypes of RSP engines have been presented. Most of them extend SPARQL by using proven techniques from DSMSs and CEP systems, namely sliding windows and continuous queries. A continuous query is registered once and produces results continuously over time as the streaming data in the considered window changes. In the past years, multiple prototypes of RSP engines have been developed [10], e.g. C-SPARQL, CQELS, EP-SPARQL and SPARQLStream.

GeoSPARQL is an Open Geospatial Consortium (OGC) standard published in 2012. It is a SPARQL extension that aims to address the issues of geospatial data representation and access. The GeoSPARQL specification provides a vocabulary, geometric class taxonomies and relations, for representing geospatial data in RDF. Moreover, it defines extensions to the SPARQL query language to query and filter on the relationships between geospatial objects. Like for standard SPARQL there is no support for streaming data.

To the best of our knowledge Parliament [2][9], a geospatial triple store platform, has the most complete GeoSPARQL implementation. It is a complete triple store and data management solution that is based on RDF, RDFS, OWL, SPARQL and GeoSPARQL standards. GeoSPARQL queries can be processed through an extension on the Jena ARQ engine.

The language stSPARQL [5], is another SPARQL extension for querying linked geospatial data. It can be used to query linked data represented in stRDF [5], an extension of RDF. Both, stSPAQRL and stRDF, have especially been designed for representing and querying geographic data that does not only changes in space but that also evolves over time. However, in analogy with GeoSPARQL there is no support to deal with streaming data.

Sextant [3] is an open-source web-based tool for visualizing time-evolving geospatial data sources. It is built on top of Strabon [6] a spatiotemporal store. Sextant has support for querying and visualizing different type of data source but has no support for querying streaming data sources.

3 C-GeoSPARQL

The C-SPARQL architecture (Fig 1)[1] is the result of an evolutionary approach in which existing solutions for streaming and semantic data are combined and treated as black-box subsystems. The two subsystems are a DSMS/CEP and a SPARQL engine. The first copes with the windowing on the RDF streams and forwards the window content to the SPARQL engine. The last one evaluates the SPARQL part of the query on a combination of both, the received window content and possible other static data sources that have to be taken into account during query evaluation. For its implementation, C-SPARQL is making use of Esper for the streaming subsystem, and Apache Jena ARQ for the SPARQL subsystem.

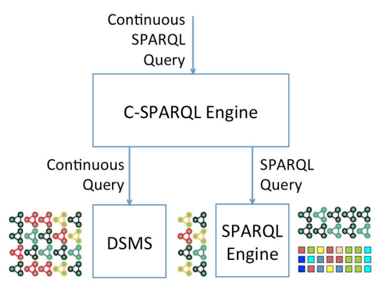

Fig. 1. C-SPARQL architecture

Listing 1.1 shows an example of a C-SPARQL query looking for the location of a ship which is named “ELM K”. The query is continuously executed with an interval of 5 seconds and takes into account all observations streamed in the last 60 seconds before execution. The output of the query will consist of the actual

[1] https://www.w3.org/community/rsp/wiki/RDF_Stream_Processors_Implementation

```
REGISTER STREAM pog AS
# prefixes have been omitted
SELECT ?o ?xWKT
FROM STREAM <http://.../sgraph> [RANGE 60s STEP 5s]
WHERE {
        ?o rdf:type caprads:Observation .
        ?o geo:hasGeometry ?g .
        ?g geo:asWKT ?xWKT .
        ?p caprads:properties ?p .
        ?p caprads:shipName “ELM K”.
```

Listing 1.1. C-SPARQL Query

```
REGISTER STREAM pog AS
# prefixes have been omitted
SELECT ?o ?sn ?xWKT
FROM STREAM <http://.../sgraph> [RANGE 60s STEP 5s]
FROM <https://datatank.stad.gent/4/infrastructuur/dokken.kml>
WHERE {
        ?o rdf:type caprads:Observation .
        ?o geo:hasGeometry ?x .
        ?x geo:asWKT ?xWKT .
        ?p caprads:properties ?p .
        ?p caprads:shipName ?sn .
        ?d rdf:type caprads:Dock .
        ?d rdfs:label “ALPHONSE SIFFERDOK” .
        ?d geo:hasGeometry ?y .
        ?y geo:asWKT ?yWKT .
        FILTER (geof:sfWithin(?xWKT, ?yWKT))
}
```

Listing 1.2. C-GeoSPARQL Query

observation identifier and the geometry of the observed location expressed in the Well-known Text[2] (WKT) format.

The black-box architecture of C-SPARQL makes it easy to replace the subsystems with other implementations. To support GeoSPARQL on the C-SPARQL engine, we can easily replace the Jena ARQ subsystem with the Parliament engine which is based on Jena ARQ as well. Due to the modularized implementation of C-SPARQL, integration of the Parliament engine is straightforward. As a result of this integration the C-SPARQL engine becomes capable of handling GeoSPARQL queries over streaming RDF data. We call this extension C-GeoSPARQL.

Listing 1.2 shows an extended version of the C-SPARQL query presented in listing 1.1. The query is now looking for all ships present in the dock labeled as “ALPHONSE SIFFERDOK”. Thanks to the parliament integration this can be done using the *geof:sfWithin(X, Y)* function. This GeoSPARQL function checks if a geometry X, describing a certain area, is located inside geometry Y, where X and Y can be expressed in WKT or in the Geography Markup Language[3] (GML). In this particular example query, *geof:sfWithin(?xWKT, ?yWKT)*, checks if the geometry related to the ship observation *(?xWKT)*, is laying within the boundaries of the geometry of the dock *(?yWKT)*.

[2] http://www.opengeospatial.org/standards/wkt-crs
[3] http://www.opengeospatial.org/standards/gml

4 The Port of Ghent: A Use Case

To practice with C-GeoSPARQL a test environment regarding the port of the city of Ghent (PoG) was implemented using TripleWave [7], RSP Services[4] and TripleGeo [8]. The test environment[5] is build with open-source libraries and makes use of open data sources.

4.1 Implementation

Figure 2 shows a high-level overview of the overall architecture. The general high-level data flow can be summarized in 5 steps:

1. Registration of a (geographic) RDF data stream in the system
2. Getting meta-data about the stream and establishing a websocket connection
3. Receiving the streaming RDF data
4. Registering additional static RDF or KML data sources
5. Registering a C-GeoSPARQL query and wait for results

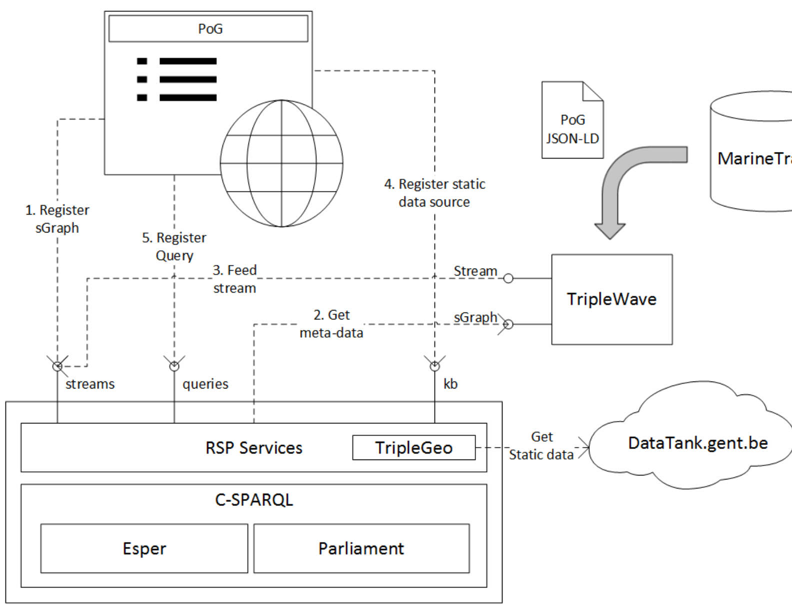

Fig. 2. High-level architecture

The geographic data sources, both streaming ship observations and static information about the port, are represented at the right side of figure 2.

[4] http://streamreasoning.org/resources/rsp-services
[5] https://github.com/adejonghe/pog-demo

An Automatic Identification System (AIS) is an automatic tracking system used for collision avoidance on ships and by Vessel Traffic Services (VTS). It produces a continuous stream of detected ships in the area under surveillance. Today the port of Ghent and Zeeland is monitored by a system called Enigma+ (Electronic Network for Information in the Ghent-Zeeland Maritime Area). Unfortunately this platform is not publicly accessible. As an alternative we make use of MarineTraffic[6] data. MarineTraffic is an internet service offering payed and free services regarding ship tracking. To test the application an off-line dataset with ship movements from MarineTraffic is used. The ship observation from MarineTraffic are stored on disk and can be replayed with TripleWave to create a stream of data consisting of ship observations.

TripleWave[7] is an open-source tool for making RDF streams available to the Web. The aim of the framework is to help creating and publishing streaming RDF data through the Web, in such a way the data can be used in a continuous execution model. The framework is a generic and flexible solution that can be used for different purposes: (1) transforming streaming data on the web to RDF streams using R2RML mappings; (2) replaying RDF dumps with temporal information; or (3) Replaying RDF data with temporal information exported through a SPARQL endpoint. For the implementation of the use case the second application is used. The stored ship observations from MarineTraffic are replayed to create the data streams.

The ship observations from MarineTraffic are encoded as GeoJSON[8]. GeoJSON is a format for encoding a variety of geographic data structures using JavaScript Object Notation (JSON). To be able to use GeoSPARQL to query over the observed geometries, these have to be expressed in WKT or GML format. Listing 1.3 shows how an observation expressed in JSON looks like after serializing the geometry into the WKT format. To annotate the observations semantically and express them in JSON-LD, a context, of which a snapshot is displayed in listing 1.4, is added.

Next to the ship observations, the system also needs information about the location of the docks, the company sites and the quays. This static data about the port is publicly available in the datatank[9] of the city of Ghent[10]. It is a data platform offering open data about the city of Ghent. The data is publicly available in Keyhole Markup Language (KML) format which is an XML based GIS format. These KML files can easily be displayed on maps. However, conversion is needed to perform semantic integration with the streaming data and other semantic data sources.

TripleGeo[11] is an open-source tool for extracting features from files with geospatial data into RDF triples. It has been integrated in the RSP services to

[6] http://www.marinetraffic.com/
[7] http://streamreasoning.github.io/TripleWave/
[8] https://tools.ietf.org/html/rfc7946
[9] http://http://thedatatank.com/
[10] http://datatank.gent.be
[11] https://web.imis.athena-innovation.gr/redmine/projects/geoknow_public/wiki/TripleGeo

```
{
  "@id":"http://idlab.ugent.be/observation/Ob1473441121001",
  "type":"Feature",
  "geometry": {
    "@id":"http://idlab.ugent.be/geometry/Ob1473441121001",
    "type":"Point",
    "asWKT":{
      "@type":"http://www.opengis.net/ont/geosparql#wktLiteral",
      "@value":"POINT(3.75623 51.11656)"
    }
  },
  "properties": {
    "@id":"http://idlab.ugent.be/properties/Ob1473441121001",
    "flag":"DK",
    "gt_shiptype":"12",
    "heading":"335",
    "length":"230",
    "destination":"BREVIK",
    "lon":"3.75623",
    "shiptype":"Cargo",
    "speed":"3",
    ...
    "ship_id":"157073",
    "shipname":"BEGONIA SEAWAYS",
    "lat":"51.11656",
    "timemillis":1473441121001,
    "timestamp":"2016-09-09T17:12:01.001Z"
  }
}
```

Listing 1.3. Ship Observation

```
{
  "@context": {
    "sf": "http://www.opengis.net/ont/sf#",
    "geo": "http://www.opengis.net/ont/geosparql#",
    "caprads": "http://idlab.ugent.be/caprads/vocab#" ,
    "xsd": "http://www.w3.org/2001/XMLSchema#",
    "Feature": "geo:Feature",
    "Point": "sf:Point",
    "properties": "caprads:properties",
    "geometry": "geo:hasGeometry",
    "asWKT": "geo:asWKT",
    "type":"@type",
    ...
  }
}
```

Listing 1.4. JSON-LD context

make it possible to register KML files as static data sources. When this type of data sources are registered at the RSP Service, the RSP Service collects the data and uses TripleGeo to convert it to RDF data that is loaded as static background knowledge into the RSP engine.

At the top left of figure 2, the PoG web interface is visualized. The user interface, a simple web application (Fig. 3), is composed of a map and some I/O text fields. The text field components are used to load static data sources, register streams and register queries. The map is used to display both, the static geographic data sources and the streaming query results. On the map in the figure 3 for example, we can distinguish docks (blue), company sites (green) and quays (red dots).

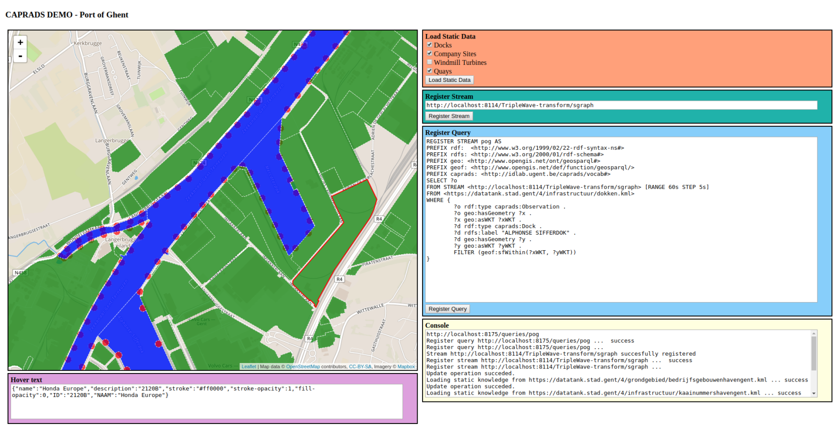

Fig. 3. User interface

The main components of the semantic stream processing unit are displayed at the bottom left of figure 2. The RSP service API offers simple REST interfaces on top of the underlying RSP engine to help exposing RSP engine capabilities to the Web. It provides different interfaces to manage streams, queries and static data sources in the underlying RSP engine. The actual RSP engine is the adapted C-SPARQL engine where the Jena ARQ subsystem has been replaced by the Parliament engine.

4.2 Scenarios

Some possible scenarios the application can deal with are:

1. Get all ships:
   In this scenario only the streaming ship observations are queried. The query is limited to C-SPARQL features and does not make use of any GeoSPARQL features.

```
# prefixes have been omitted
CONSTRUCT {
  ?s rdf:type geo:Feature .
  ?s geo:hasGeometry ?x .
  ?x geo:asWKT ?xWKT .
  ?s caprads:properties ?p .
  ?p ?pred ?prop .
}
FROM STREAM <http://.../sgraph> [RANGE 180s STEP 3s]
FROM <http://.../incidents.rdf>
FROM <http://.../bedrijfspercelenhavengent.kml>
WHERE {
  ?s rdf:type geo:Feature .
  ?s rdf:type ?t .
  ?s geo:hasGeometry ?x .
  ?x geo:asWKT ?xWKT .
  ?s caprads:properties ?p .
  ?p ?pred ?prop .
  ?f rdf:type caprads:CompanySite .
  ?f caprads:impactedBy ?incident .
  ?f geo:hasGeometry ?y .
  ?y geo:asWKT ?yWKT .
  ?incident rdf:type caprads:Incident .
  ?incident caprads:evacuationRadius ?radius .
  ?incident caprads:radiusUnit ?unit .
  {
    SELECT ?id (MAX (f:timestamp(?z,caprads:shipId,?id)) AS ?ts)
    WHERE {
      ?z caprads:shipId ?id
    }
    GROUP BY ?id
  }
  FILTER(f:timestamp(?s,rdf:type,?t) = ?ts)
  FILTER(geof:sfWithin(?xWKT, geof:buffer(?yWKT,?radius,?unit)))
}
```

Listing 1.5. Scenario 3: Get ships in evacuation zone

2. Get all ships of a certain type present in a certain dock:
   This scenario makes use of both streaming ship observations and static information about the docks. Ship observations can be selected based on the gt_shiptype property, and filtered making use of the GeoSPARQL function *geof:sfWithin(X, Y)* where X is the geometry of the observed ship and Y is the geometry of the dock.

3. Get ships in evacuation zone:
   By means of example the query for this scenario is shown in listing 1.5. In this query three data sources are used: the stream with ship observations, information about company sites and a file with incidents. An incident can be a fire or a kind of leak like an oil or gas leak. Thanks to the RDFS reasoning support of the C-SPARQL engine we can make abstraction of the actual incident type. The incident is related to a company site with the *impactedBy* property. To determine if ships are in a certain evacuation zone filtering can be performed using the GeoSPARQL functions *geof:sfWithin(X, Y)* and *goef:buffer(X, radius, unit)*.

5 Conclusions & Future Work

In this paper we presented how the C-SPARQL engine, designed for querying RDF data streams, can easily be extended with GeoSPARQL support to allow querying geospatial RDF data streams. To test the solution a use case regarding the port of the city of Ghent was implemented.

Future work includes looking at benchmarking the extended C-SPARQL engine to quantify the impact of the integration with the Parliament library. For this a set of well chosen test queries, that allow us to measure the execution time spent at evaluating the GeoSPARQL functions, has to be defined. Another thing to look at is integration of geographic stream processing in more advanced semantic visualization tools for geographic data.

Acknowledgement This research was partly funded by the strategic research project DiSSeCt funded by the AIO and FWO, and the CAPRADS imec.ICON Project co-funded by the AIO, imec, Luciad, Televic and JForce.

References

1. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-sparql: Sparql for continuous querying. In: Proceedings of the 18th international conference on World wide web. pp. 1061–1062. ACM (2009)
2. Battle, R., Kolas, D.: Enabling the geospatial semantic web with parliament and geosparql. Semantic Web 3(4), 355–370 (2012)
3. Bereta, K., Nikolaou, C., Karpathiotakis, M., Kyzirakos, K., Koubarakis, M.: Sextant: Visualizing time-evolving linked geospatial data. In: Proceedings of the 2013th International Conference on Posters & Demonstrations Track-Volume 1035. pp. 177–180. CEUR-WS. org (2013)
4. Della Valle, E., Ceri, S., Harmelen, F., Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems 24(6), 83–89 (2009)
5. Koubarakis, M., Kyzirakos, K.: Modeling and querying metadata in the semantic sensor web: The model strdf and the query language stsparql. In: Extended Semantic Web Conference. pp. 425–439. Springer (2010)
6. Kyzirakos, K., Karpathiotakis, M., Koubarakis, M.: Strabon: a semantic geospatial dbms. In: International Semantic Web Conference. pp. 295–311. Springer (2012)
7. Mauri, A., Calbimonte, J.P., DellAglio, D., Balduini, M., Brambilla, M., Della Valle, E., Aberer, K.: Triplewave: Spreading rdf streams on the web. In: International Semantic Web Conference. pp. 140–149. Springer (2016)
8. Patroumpas, K., Alexakis, M., Giannopoulos, G., Athanasiou, S.: Triplegeo: an etl tool for transforming geospatial data into rdf triples. In: EDBT/ICDT Workshops. pp. 275–278 (2014)
9. Patroumpas, K., Giannopoulos, G., Athanasiou, S.: Towards geospatial semantic data management: strengths, weaknesses, and challenges ahead. In: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 301–310. ACM (2014)
10. Su, X., Gilman, E., Wetz, P., Riekki, J., Zuo, Y., Leppänen, T.: Stream reasoning for the internet of things: Challenges and gap analysis. In: Proc. of the 6th International Conference on Web Intelligence, Mining and Semantics (WIMS). ACM (June 2016)