On Demand SPARQL Extension: A Case Study of Extending Geo-SPARQL for Sensor Data Exploration in Semantic Cities

Snehasis Banerjee and **Debnath Mukherjee**
TCS Innovation Labs, Tata Consultancy Services, Ecospace 1B Building, Kolkata, 700156, India.
{snehasis.banerjee, debnath.mukherjee}@tcs.com

Abstract

SPARQL is the standard language for querying on RDF data, however it lacks support for many useful functions that a semantic city service may require. In this paper, a system is discussed where functions written in commonly known programming language like Java can be instantly added to the SPARQL function library, thereby decreasing development time as well as keeping the underlying implementation transparent. As a case study for instant SPARQL extension, we have chosen to extend the Geo-SPARQL library. We propose various functionality missing in Geo-SPARQL specification and discuss how their instant addition makes sensor data exploration in semantic cities more rich.

Introduction

With a wide variety of RDF data available, the number of rich SPARQL queries possible is humongous. However, expressing complex logic in form of SPARQL queries that require computations apart from graph pattern matching is not handled well. As an example, 'co-prime' is not supported by default, and the logic for the same has to be written in SPARQL syntax each time a query needs a calculation of mutual prime. One way is to extend the existing SPARQL grammar, but that needs implementation expertise that a normal SPARQL query user need not have. Hence in this paper, an approach is proposed where a user can paste from some source or write code in native language of a functionality (say co-prime), and the same will be instantly added as a function for use across multiple queries. In this way the query writer need not know the implementation details; and can use a set of functions that some other user have created, which will lead to a collaborative ecosystem. To showcase the proposed functionality, we have taken help of a use case where users can formulate rich queries over city sensor data, whose interface is named as Sensor Explorer. Sensors are attached with a location, so Geo-SPARQL, a SPARQL extension in the geo-spatial domain plays a major role in supporting spatial queries. However, the Geo-SPARQL specification by OGC[1] lacks some features which are required to satisfy many useful Sensor Explorer queries, thereby making extension a necessity for this as well as similar use cases.

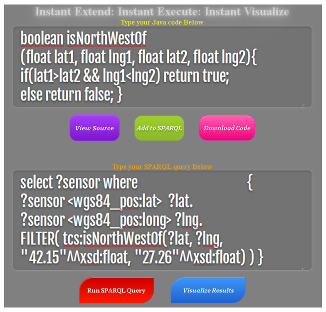
[1]http://www.opengeospatial.org/standards/geosparql

Section 2 discusses possible SPARQL enhancements for Sensor Explorer use case while section 3 describes the proposed system. Finally section 4 concludes the paper.

Extending SPARQL for Sensor Explorer

Sensor data queries in a semantic city can be of 2 types:
a)Query on sensor meta-data (like age of a sensor)
b)Query on sensor data or readings (like humidity level)
Geo-SPARQL is a good choice as a standard for sensor data exploration. However, in the spatial domain, Geo-SPARQL specification does not handle Orientation aspect at all, partially handles Proximity and Containment and misses on some other aspects as shown below.
1. Orientation:- The basic features in Geocentric reference system missing from the specification are: a) North b) East c) West d) South, and by combining the aspect of angle between two entities we can have a set of function such as 'isNorthOf', 'isNorthWestOf', 'isParallellToEquator', 'isNorthWestOfAtAngle(x)' and so on. For egocentric and external reference system, Oriented Point Relation Algebra is one of the ways to represent qualitative relations. The list of functions include 'isFrontOf', 'isBackOf', 'isLeftOf', 'isRightOf', relative angles and their combination. Some other functions include 'isAbove' and 'isBelow' in case of 3-D. A simple query is to find the average temperature to the north of a specified zone.
2. Proximity:- It is the calculation of distance between two points and can be quantitative like within 10 km as well as qualitative like 'near' and 'far' (fuzzy functions), the latter being missing from the Geo-SPARQL specification. Often distance between two points is determined by the path, as a straight line journey is infeasible. This leads to functions like 'distance-on-road', 'distance-by-flight' and the like. A simple function can be 'quickest-distance' that returns the distance in terms of best suitable travel mode, similar to the Google Maps API[2]. Another aspect in this regard is that sometimes distance is denoted in terms of time to reach a destination instead of path length, so functions in that category can be of the form 'reachableWithIn(x, y time units)'.
3. Containment:- One interesting function missing in Geo-SPARQL specification is 'between' (whether an object lies between two objects). This can be determined if an object

[2]https://developers.google.com/maps/documentation


Instant Extend: Instant Execute: Instant Visualize
Type your Java code Below
View Source
Add to SPARQL
Download Code
Type your SPARQL query Below
Run SPARQL Query
Visualize Results

```
boolean isNorthWestOf
(float lat1, float lng1, float lat2, float lng2){
if(lat1>lat2 && lng1<lng2) return true;
else return false; }
```

```
select ?sensor where                    {
?sensor <wgs84_pos:lat>  ?lat.
?sensor <wgs84_pos:long> ?lng.
FILTER( tcs:isNorthWestOf(?lat, ?lng,
"42.15"^^xsd:float, "27.26"^^xsd:float) ) }
```

Figure 1: Frontend screen to run query

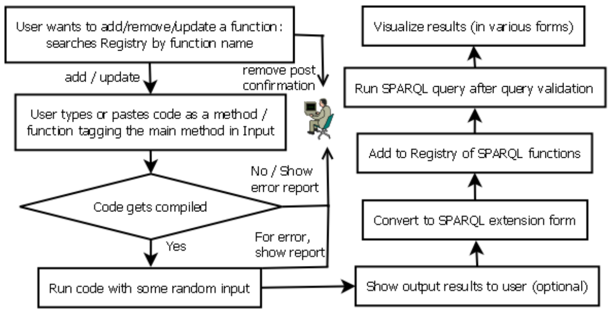

User wants to add/remove/update a function : searches Registry by function name
add / update
remove post confirmation
User types or pastes code as a method / function tagging the main method in Input
Code gets compiled
No / Show error report
Yes
For error, show report
Run code with some random input
Show output results to user (optional)
Convert to SPARQL extension form
Add to Registry of SPARQL functions
Run SPARQL query after query validation
Visualize results (in various forms)

Figure 2: System Workflow

lies in the area formed by connecting the ends of 'n' objects. An example query is to list the active camera sensors between two specified buildings.
4. Pattern-based:- In 2-D and 3-D space, spatial objects can form patterns of geometric functions and shapes such as if four objects form a square or not. An example query can be to find collinear sensors in a zone.
5. Altitude:- It is a neglected feature in the spatial extensions of SPARQL. Usually sensor locations can be expressed as a tuple (latitude, longitude, altitude) which makes way for a rich function set like 'higher' and 'lower'. 'Visibility' between two points (considering possible obstructions) is another interesting function.
5. External API calls:- As a large number of computational tasks are provided as a web service, using API calls instead of whole function implementation is the way to go. As an example, instead of using the standard Haversine[3] formula to calculate distance between two points on Earth's surface, the distance computation API of Google Maps can be used for better results.
6. External Tools:- Offloading computation to external tools, containg well tested function implementations is another approach. For example, statistical calculations can be offloaded to 'R' tool and the results fetched back.

System Description

A J2EE based system was developed where Apache Jena ARQ[4] was used as the SPARQL engine. If a user wants to add a function say 'isNorthWestOf', then the user searches a Registry of existing functions. The Registry is a code base of functions that a user can use for querying. The code base contains a) Standard functions: some frequently used functions b) Defined functions: functions added by the user c) Shared functions: functions added by other users and shared for re-usability. The user needs to turn on the functions that he or she wish to use. The user can remove and update self-created functions. If a desired function does not exist, the user can add that to Registry. If a desired function does exist, but some logic needs to be changed for application specific requirements, it can be done by overriding the function for that user. Fig. 1 showcases the front-end screen where a Java code is either typed or pasted and the same is added to the SPARQL library (post validation) which is used by the query below, keeping the users transparent from implementation. The system response time (time to add a function in Registry for usage) lies within a few seconds. In Fig. 2, the proposed system is illustrated. If a desired function do not exist in Registry, the user can add that function by tagging the 'main' method in case of multiple modules. Then the code is added to SPARQL library for immediate use, and the same can be shared with others for use. Finally the user visualizes the result in an appropriate form. There are some intermediate steps like error handling and error reporting.

Conclusion and Future Work

In this paper, a system and method for instant addition of SPARQL functions is showcased by extending SPARQL, especially Geo-SPARQL for sensor discovery. In future, support for more programming languages apart from Java to express logical functionality are planned. Future area of work include stabilizing and testing the system, and managing external function and tool integration. Some of the Geo-SPARQL extensions discussed here will be proposed to the OGC steering committee for possible inclusion into the specification.

References

Battle, R., and Kolas, D. 2011. Geosparql: Enabling a geospatial semantic web. *Semantic Web Journal* 3(4):355–370.

Dey, S.; Dasgupta, R.; Pal, A.; and Misra, P. 2013. Sensxplore: A tool for sensor discovery using semantics with focus on smart metering. Semantic Web Challenge.

Mossakowski, T., and Moratz, R. 2012. Qualitative reasoning about relative direction of oriented points. *Artificial Intelligence* 180:34–45.

[3]http://www.movable-type.co.uk/scripts/latlong.html
[4]https://jena.apache.org/documentation/query/