Mobile Endpoints: Accessing Dynamic Information from Mobile Devices

Roberto Yus and Eduardo Mena

University of Zaragoza, Spain
{ryus,emena}@unizar.es

Abstract. Mobile devices are ubiquitous and, due to their increasing number of sensors and their powerful features, enable users to consume and produce huge amounts of highly-dynamic data (such as location data, other devices in range, and another local context information recollected by sensors). Semantic techniques can be applied to offer *smart data* to their users and remote client requests.
In this paper we present the use of the SHERLOCK system as endpoint that processes *GeoSPARQL-DL* queries over mobile devices. It uses the GeoSPARQL extension to support location-based queries and the SPARQL-DL extension to support queries over OWL ontologies. To process the queries the system obtains dynamic data directly from mobile devices by communicating with them in a P2P manner.

Keywords: Semantic Web, Mobile Computing, SPARQL

1 Introduction

Current mobile devices (e.g., smartphones and tablets) are equipped with a high number of sensors and communication mechanisms which make them not only consumers but producers of interesting dynamic information. In this scenario it is possible to develop useful information systems that, for example, use the location of users obtained from their mobile devices for taxi searching, detecting nearby friends, or restaurant suggestion, among many others.

The Semantic Web has been proven useful to publish information from sensors using OWL and RDF. Also, the query language SPARQL has become the standard to retrieve and manipulate data stored in RDF format. However, although users with mobile devices are sometimes consider sensors, having a centralized Knowledge Base (KB) with updated information from them is not feasible: There are millions of mobile devices generating information continuously and also there could be other privacy and security implications.

Therefore, due to the features of this scenario it would be more interesting to view each device as a possible resource that can be queried independently, i.e., as *mobile endpoints*. There are many challenges to take into account in this scenario such as the collaboration among devices and the handling of ad hoc communications. We are developing a general system (SHERLOCK [5]) which

provides users with interesting Location-Based Services (LBSs) and tackles many of these challenges. SHERLOCK executes on mobile devices and leverages their communication mechanisms to exchange information among them in an ad hoc manner. The system uses: 1) OWL ontologies and semantic reasoners based on Description Logics (DL) to handle knowledge about LBSs and interesting objects, and 2) mobile agents to bring computation wherever needed and balance CPU consumption and communication load.

Here we introduce the processing of *GeoSPARQL-DL* queries, which makes SHERLOCK-enabled devices behave as mobile GeoSPARQL-DL endpoints that obtain information directly from mobile devices around to answer user requests. We have adapted an existing Android prototype of the system to process these queries and performed some preliminary experiments.

2 The Query Language

SHERLOCK provides users with LBSs and manages OWL ontologies and a semantic DL reasoner to infer implicit knowledge exchanged among the devices. Therefore, we define *GeoSPARQL-DL*, the query language used by the system, as the combination of two preexisting extensions to SPARQL:

- GeoSPARQL [1]: a standard for representation and querying of geospatial linked data from the Open Geospatial Consortium (OGC).
- SPARQL-DL [3]: a subset of SPARQL fully aligned with OWL 2 and which can be used for the specific questions typically associated with OWL.

As an example of GeoSPARQL-DL query see Figure 1 where a user has shown her interest in transports, different from taxis, within a radius of 1 km. Notice that the restriction on the type of the objects has been modeled using SPARQL-DL which would include subclasses of the `sherlock:Transport` class which are not subclasses of `sherlock:Taxi` (e.g., `sherlock:Shuttle`); also, the restriction on the location has been modeled using the `geof:within` and `geof:buffer` operands from GeoSPARQL.

3 Query Processing

The high-level algorithm used to process a query follows these steps:

1. *Execute the query against the KB on the user device* that could contain the information requested from previous interactions.
2. *Evaluate the need of querying external KBs*: The results obtained in the previous step are analyzed to evaluate if they are good enough for the user:
   (a) *Check the timestamp associated to each fact*: In this scenario, information changes dynamically so, for example, the location obtained for a taxi 10 minutes ago might not be interesting for the user.
   (b) *Check the number of results*: If the answer obtained locally is empty or below a certain limit, it should be upgraded.

```
PREFIX sherlock: <http://sid.cps.unizar.es/ontology/sherlock/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX geof: <http://www.opengis.net/def/geosparql/function/>

SELECT ?name, ?type, ?timestamp, ?lat, ?lon
WHERE {
        Type(?thing, sherlock:Transport),
        DirectType(?thing, C),
        DisjointWith(?C, sherlock:Taxi),
        FILTER(geof:within(?thing,
          geof:buffer(51.139725, -0.895386, 1, 'km'))),

        PropertyValue(?thing, sherlock:name, ?name),
        PropertyValue(?thing, geo:lat, ?lat),
        PropertyValue(?thing, geo:lon, ?lon),
        PropertyValue(?thing, sherlock:timestamp, ?timestamp)
}
```

Fig. 1. GeoSPARQL-DL sample query to return the list of transports nearby.

3. If the query has to be posed to external KBs, SHERLOCK will try to query devices in the geographic area relevant for the query, to maximize the chances of obtaining information interesting for the user, as follows:
   (a) *Split the query for each non overlapping geographic constraint*: There will be a tracker agent in charge of monitoring the geographic area associated with each geographic constraint.
   (b) *Send agents to devices in the relevant area*: Each tracker agent creates mobile agents that will move to devices inside the relevant area that such a tracker agent has to monitor. These agents will pose the query to the devices.
   (c) *Each device executes the query against its local KB* and returns as answer those objects fulfilling the query constraints (including geospatial and DL constraints, these last ones evaluated by the DL reasoner on each device).
   (d) *Each tracker correlates the results obtained* by its mobile agents and returns it to SHERLOCK.

Finally, the results obtained from each tracker are correlated and presented on the user device (location data are shown on a map).

4 Prototype

Up to the authors' knowledge, there is not implementation of GeoSPARQL-DL yet so, we developed a simple processor combining existing approaches. For SPARQL-DL, we used a combination of the OWLAPI[1], the SPARQL-DL API[2], and the HermiT reasoner [2], after porting it to Android [4]. For GeoSPARQL, there exist some partial implementations but designed for large triple stores therefore, we have implemented a simple processor for the `geof:within` and `geof:buffer` functions.

[1] `http://owlapi.sourceforge.net`
[2] `http://www.derivo.de/en/resources/sparql-dl-api`

For a preliminary experiment[3] we used four real devices: three smartphones (simulating a taxi and two shuttles) and a tablet (simulating a user looking for transportation). SHERLOCK on the tablet processed the query in Figure 1 on the local KB and, to improve the answer, a mobile agent moved to one of the smartphones to query the rest of devices in the vicinity about objects classified as transports (SPARQL-DL Type constraint) which are not taxis (SPARQL-DL DisjointWith constraint) and within a radius of 1 km from the user (GeoSPARQL geof:within constraint). The location of the two shuttles (simulated by the smartphones), which fulfill all the constraints, were returned as an answer which were shown on a map on the user device.

5 Next Steps

In this paper we introduced the *GeoSPARQL-DL* processing capabilities of SHERLOCK-enabled mobile devices to provide their users with interesting LBSs. We have introduced this query language (a combination of the SPARQL-DL and GeoSPARQL extensions) and its processing, as well as the prototype developed. In this way, a SHERLOCK-enabled device behaves as a mobile SPARQL endpoint for users and transparently obtains the results deploying agents to query the appropriate devices in a P2P manner. Also, the use of a query language based on SPARQL enables the system to obtain results from external endpoints such as DBpedia or GeoNames.

In the future we plan to complete our implementation of GeoSPARQL-DL focusing on supporting more GeoSPARQL functions and to incorporate a web interface for users to pose queries to SHERLOCK-enabled devices. Also, we plan to study the use of existing approaches to specify privacy policies regarding the information from a user KB that agents from other devices can access.

Acknowledgments. This research was supported by the CICYT project TIN-2013-46238-C4-4-R and DGA FSE.

References

1. Battle, R., Kolas, D.: GeoSPARQL: enabling a geospatial Semantic Web. Semantic Web Journal 3(4), 355–370 (2011)
2. Shearer, R., Motik, B., Horrocks, I.: HermiT: A highly-efficient OWL reasoner. In: OWLED. vol. 432, p. 91 (2008)
3. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: OWLED. vol. 258 (2007)
4. Yus, R., Bobed, C., Esteban, G., Bobillo, F., Mena, E.: Android goes semantic: DL reasoners on smartphones. In: 2nd International Workshop on OWL Reasoner Evaluation (ORE 2013). pp. 46–52 (2013)
5. Yus, R., Mena, E., Ilarri, S., Illarramendi, A.: SHERLOCK: Semantic management of location-based services in wireless environments. Pervasive and Mobile Computing 15, 87–99 (2014)

[3] `http://sid.cps.unizar.es/SHERLOCK` for videos and more information.