Integrating and Querying OpenStreetMap and Linked Geo Open Data

Jesús M. Almendros-Jiménez, Antonio Becerra-Terón and Manuel Torres

Dept. of Informatics. University of Almería. SPAIN.
Email: {jalmen,abecerra,mtorres}@ual.es

In recent years, Open Street Map (OSM) has evolved into a highly popular geospatial system. The key of success of OSM is that OSM is open to absolutely everyone. OSM is not dependent on any one government, company, university, or international organization. OSM is based on crowdsourcing, in which users collaborate to collect spatial data of urban and rural areas on the earth. With the arising of Linked Open Data (LOD) initiative, and more concretely with Linked Geo Open Data (LGOD), many Web resources have been made available to everyone, providing geo-located datasets. In this paper a framework, called *XOSM (XQuery for OpenStreetMap)*, for integrating and querying OSM and LGOD resources is presented. The framework is equipped with a Web tool and a rich XQuery-based library, enabling the definition of queries combining OSM layers and layers created from LGOD resources (KML, GeoJSON, CSV and RDF (and also XML)). The framework also provides an API to execute XQuery queries using the library.

Keywords: OpenStreetMap, Linked Open Data, XML, XQuery

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

Volunteered Geographic Information (VGI) is a term introduced by Goodchild [1, 2, 3] to describe geographic information systems based on crowdsourcing, and *OpenStreetMap* (OSM) [4, 5, 6, 7] is one of the most relevant VGI systems, with more than three millions of registered users. OSM data can be visualized from the OSM Web site[1], and many applications[2] have also been built for the handling of maps.

With the arising of *Linked Open Data (LOD)*[3][8, 9] and, more concretely, *Linked Geo Open Data (LGOD)*[4][10], many Web resources are available providing geo-located datasets. OSM takes part of the LGOD initiative. However, many other LGOD Web resources are provided by several institutions, organizations and public administrations. The *European Union*[5], *ArcGIS Open Data*[6], and *Open Data Inception* [7] are the most relevant sites collecting LGOD data.

The integration of OSM data and other geo-located datasets can be useful from many points of view. While users contribute to OSM maps, these maps can be also *enriched by information provided by some other Web resources*, which can be also used to improve and refine queries on OSM maps. The information provided by other Web resources can complement (and also correct) information provided by users. However, most of Web resources provide information which is not in a suitable format. *CSV* format is used extensively by institutions to provide information, since the table-like format is commonly used to store datasets. Some other cases provide data in *JSON* format, mostly generated by Web APIs, and particularly in *GeoJSON* when geo-spatial information is included. In some other cases, the influence of the well-stablished geo-spatial tool *Google maps* has caused the use of *KML* format for storing geo-spatial information. Finally, *RDF*, which is promoted by *Semantic Web* initiatives, has been also adopted as knowledge representation format, while this is still far from the common use.

XQuery [11, 12] is a programming language proposed by the W3C as standard for the handing of XML documents. It is a functional language in which *for-let-*

[1] http://www.openstreetmap.org
[2] http://wiki.openstreetmap.org/wiki/Software
[3] http://linkeddata.org/
[4] http://linkedgeodata.org
[5] https://data.europa.eu
[6] http://opendata.arcgis.com/
[7] https://opendatainception.io/

orderby-where-return (FLOWR) expressions are able to traverse XML documents. It can express Boolean conditions and provides format to output documents. XQuery has a sublanguage, called *XPath* [13], whose role is to address nodes on the XML tree. XPath is properly a query language equipped with Boolean conditions and many path-based operators. XQuery adds expressivity to XPath by providing mechanisms to join several XML documents.

XQuery is not just a query language, but a fully-fledged programming language, equipped with a very large library, able to define complex and recursive procedures in hierarchical data. For instance, XQuery can be used for defining transformations. These transformations can be XML to XML, but also XML to HTML, XML to SVG, XML to Text, etc. *XSLT* is also considered a transformation language for XML, but XSLT is too procedural and verbose, and too hard to maintain, while XQuery is a more elegant and high-level language providing, for instance, higher-order functions which make easier the definitions of queries and libraries. Additionally, XQuery is a database query language and implementations are able to efficiently handle large XML datasets.

A programming language able to query and transform data is a *great challenge* in the era of multiple formats, tools and Web data resources. XQuery is already able to handle multiple XML spatial formats: for instance, GML, KML and GeoJSON, as well as XML-based Semantic Web languages: RDF and OWL. XML is adopted in all of them as basis to represent spatial and textual data. CSV is also handled by XQuery due to XQuery libraries.

Thus, the best candidate for integrate multiple spatial datasets is XQuery. In the context of OSM, XQuery can play a key role, offering data transformations from CSV, GeoJSON, KML and RDF (and also XML) to the OSM model. Fortunately, the key-value based representation of spatial data in OSM makes easy the *transformation from other formats to OSM*. Having multiple datasets in OSM format, multiple layers of OSM can be handled, which can be integrated and queried.

In this paper a framework, called *XOSM (XQuery for OpenStreetMap)* (see Figure 1), for integrating and querying OSM and LGOD resources is presented. The framework is equipped with a Web tool (accessible from our Website[8]) and a rich XQuery-based library, enabling the definition of queries combining OSM layers and layers created from LGOD resources (KML, GeoJSON, CSV and RDF (and also XML)). The framework also provides an API to execute XQuery queries using the library.

The framework offers two main components: a *transformation library* and a *spatial/textual query library*. The transformation library provides functions to transform KML, GeoJSON, CSV and RDF (and also XML) formats into OSM format. The spatial query library offers functions to query spatial data with typical *spatial operators* (*crossing*, *touching*, etc), as well as *keyword search operators*, adapted to the OSM key-value structure, and *aggregation operators* which take advantage of the *higher-order* mechanisms of XQuery. While the spatial expressivity power of XOSM can be considered similar to other spatial query languages, for instance, the well-known spatial SQL database management system *PostGIS*, XOSM offers transformation functions which can be used in combination with spatial functions. In such a way that users can define queries involving on-the-fly (via http requests) combinations of several OSM layers representing existing LGOD web resources as well as OSM data.

The LOD data initiative promotes the use of SPARQL as query language, and, in the case LGOD, spatial extensions of SPARQL have been proposed: *GeoSPARQL* and *stSPARQL* [14, 15, 16, 17]. SPARQL works with the RDF model, and some well-known RDF datasets are available, for instance, DBpedia[9] and YAGO[10]. OSM has been integrated in the RDF area due to *OSM Semantic Network*[11] in which OSM map data are available in RDF format. However, when other LGOD Web resources do not provide datasets in RDF format, it imposes a *barrier for integration and querying*.

Transformations in these languages are, in some cases, implemented in external languages (C, Java, etc.,) but the absence of on-the-fly transformations limits the *interoperability of the query language*. On the other hand, SPARQL is still a limited query language in many of their implementations, lacking, for instance, on appropriate nesting mechanisms of queries[12], and the function library is rather than limited.

XOSM handles the main LGOD formats: KML, GeoJSON, CSV, RDF and also XML. Most popular open data sites offer several data formats. For instance, http://opendata.arcgis.com/ provides all the content in GeoJSON format via an API service. The site https://opendatainception.io/ offers links to other open data sites, and the availability of a given format depends on the dataset provider. They do not provide statistics about datasets format. The site https://catalog.data.gov categorizes datasets by nature (spatial and non spatial) and, as the Table 1 shows, the most used format for spatial data is XML, together with CSV and KML. In https://www.europeandataportal.eu and https://data.gov.uk, CSV is the most used structured format

[8] http://xosm.ual.es/XOSM/

[9] http://wiki.dbpedia.org/

[10] http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

[11] http://wiki.openstreetmap.org/wiki/OSM_Semantic_Network

[12] SPARQL 1.1 proposes extensions for covering nesting, recursion and aggregation [18, 19] but it is not still available in most of implementations.

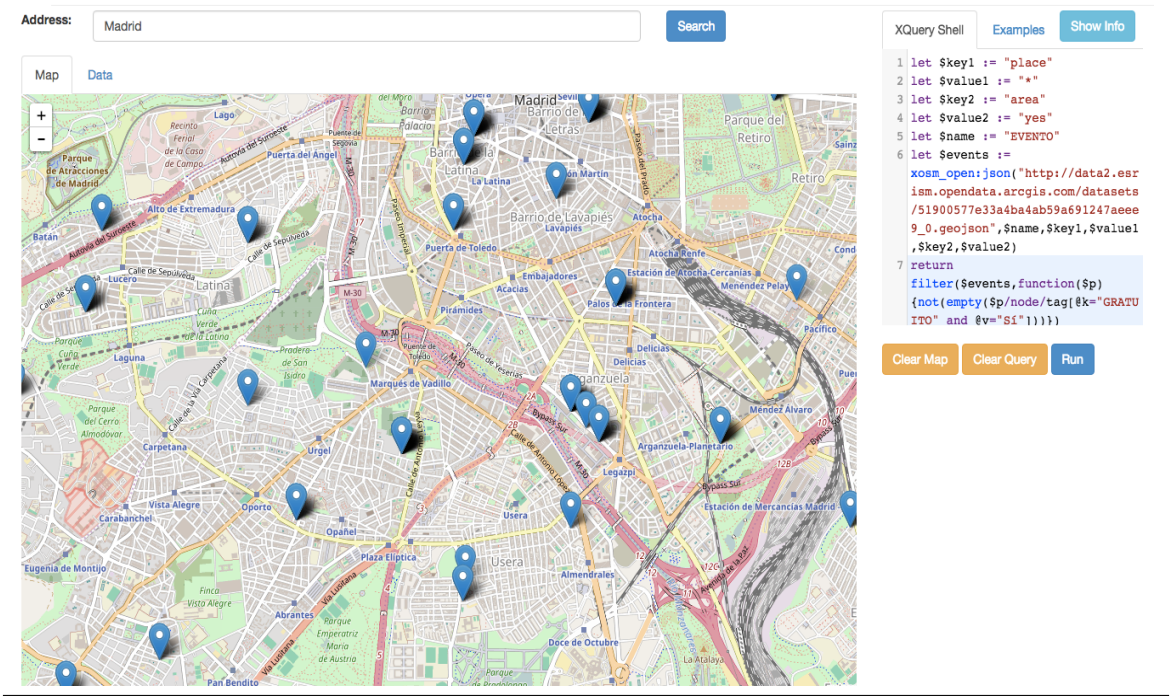

FIGURE 1. XOSM Web Interface

Country	Site	Subject	KML	GeoJSON	CSV	RDF	XML
EU	https://www.europeandataportal.eu	Regions and Cites	1,277	1,252	3,448	45	744
EU	https://www.europeandataportal.eu	Transport	226	133	1,314	48	386
EU	https://www.europeandataportal.eu	Govern. and Public	402	358	8,802	514	1,336
EU	https://www.europeandataportal.eu	Environment	1,267	516	6,895	129	1,676
EEUU	https://catalog.data.gov	GeoSpatial	3,567	989	3,924	-	15,748
UK	https://data.gov.uk	Mapping	895	911	978	-	-
UK	https://data.gov.uk	Towns and Cities	139	78	525	25	206
UK	https://data.gov.uk	Transport	20	32	216	22	61

TABLE 1. LGOD formats of most popular open data sites.

(see Table 1) including both spatial and non-spatial datasets. For spatial information, KML and GeoJSON are extensively used. Unfortunately, HTML, Microsoft Word, and other non structured formats are frequently used in https://www.europeandataportal.eu and https://data.gov.uk. Thus, a challenge of the LGOD initiative is to motivate the use of more suitable formats of spatial information exchanging. In our approach, the transformation library is used to convert spatial datasets represented in KML, GeoJSON, CSV, RDF and XML into OSM format. The advantage of these spatial data formats is that they are XML-compliant languages.

Obviously, other XML-compliant languages could be considered in our approach (for instance, GML and HTML). The definition of specific transformation functions for these kinds of formats will be subject of future development. In general terms, in order to provide a transformation library from LGOD formats to OSM, datasets should provide an uniform/standard way to represent spatial objects and their location, that is, points, lines, polygons, and their coordinates, as well as an uniform/standard way to represent a list of properties on each spatial object, in particular, a name and key-value pairs of properties in order to be mapped into OSM.

As an example of use of XOSM, let us suppose a tourist office wants to offer transport information to tourists. In particular, the taxi stations, close to main monuments of the city. And let us suppose the OSM map does not include information about taxis. In this case, the tourist office staff looks for taxis stations provided by the city government. This is, for instance, the case of Paris, offering taxis stops localization in the Web site https://opendata.paris.fr/explore/dataset/paris_taxis_stations/export/ (see Figure 3). Now, the JSON file of taxi stations can be used on the fly, and the information of the file can be integrated using our

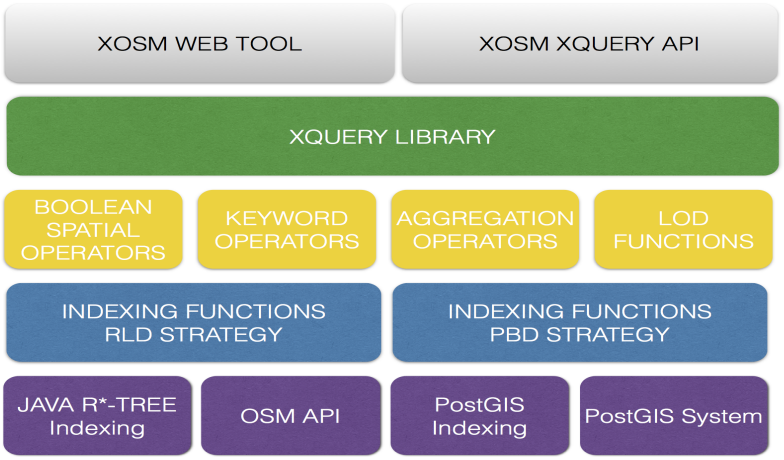

FIGURE 2. XOSM Components

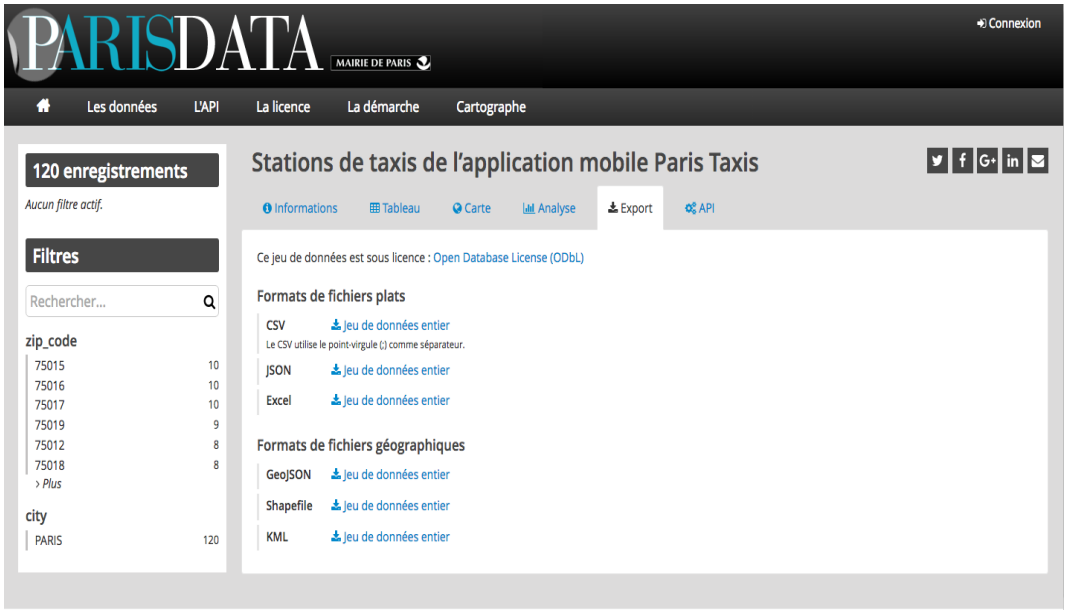

FIGURE 3. LGOD Resource of Paris Taxis

XOSM framework with the information of the OSM map of Paris as follows:

```
let $taxis := xosm_open:json("https://opendata.paris.fr/
     explore/dataset/paris_taxis_stations/download/?
     format=geojson&amp;timezone=Europe/Berlin",
    "address","amenity","taxi","","")
let $building :=
 xosm_rld:getElementByName(.,"Carrousel du Louvre")
return
 fn:filter($taxis,xosm_sp:DWithIn($building,?,500))
```

in which taxi stations at distance of 500 meters to the "Carrousel du Louvre" are requested. Thus not only integration but spatial querying can be carried out with XOSM. This is not the only case in which XOSM can be more suitable than SPARQL and PostGIS. Since XQuery is a fully-fledged programming language, XOSM library can be used to program *more complex queries than SPARQL and PostGIS queries*. For instance, thanks to the recursive nature of the XQuery language, a *routing algorithm* in OSM maps

can be programmed. We will show in the paper the implementation of a routing algorithm using the XOSM library and XQuery.

In summary, the presented approach can be seen as a rich infrastructure enabling the definition of complex queries, possibly making use of several LGOD Web resources, and the visualization of results. XOSM architecture (see Figure 2) can be summarized as follows:

1. **XOSM Web Tool**: A Web tool has been developed enabling (1) selection of a bounding box of a OSM map, (2) editing and parsing of queries and (3) running and visualization of the result. When the result is text or numeric, this one is shown in a pop-up window. Otherwise (i.e. OSM spatial elements), ways of the result are highlighted in a different color and nodes of the result are marked with an icon.
2. **XOSM API**: The Web tool is not the only mechanism to execute queries. XOSM provides a service defined by an API restful which allows to run XOSM queries getting results in text/numeric and OSM format. The defined service requires the coordinates of a bounding box of an OSM map. These coordinates are passed as parameters of a service call, simulating the mouse-based selection from the Web tool.
3. **XQuery Library**: In order to handle OSM elements in XOSM, two main XQuery functions are provided: *getLayerByName* and *getElementsbyKeyword*. *getLayerByName* is a spatial function in order to retrieve OSM elements at a certain distance w.r.t. a named element. *getElementsbyKeyword* is a keyword search function in order to retrieve OSM elements labeled with a certain keyword. Both can be used in isolation or in combination to retrieve a set of OSM elements from an OSM map. Once the elements are retrieved, a set of Boolean spatial, keyword and aggregation operators can be used to query OSM maps. Since *getLayerByName* and *getElementsbyKeyword* use distance and keyword information to retrieve elements, both ones need an index in order to improve performance (see next **Indexing** item). The implementation of the Boolean spatial operators by XOSM is based on the use of the *BaseX XQuery Geo Module*[13] available in the BaseX XQuery library. This BaseX XQuery Module contains functions that may be applied to geometry data conforming to the Open Geospatial Consortium (OGC) Simple Feature (SF) data model. It is based on the EXPath Geo Module[14] and uses the *Java Topology Suite (JTS)* library. The implementation of the keyword and aggregation operators by XOSM has been developed in XQuery. Finally, a set of LGOD functions are provided by the XQuery library, enabling automatic conversion from CSV, KML, GeoJSON and RDF (and also XML) to XML OSM format. The implementation of LGOD functions has been developed in XQuery.
4. **Indexing**: Two indexing mechanisms are handled: *RLD* and *PBD*. *RLD* uses a Java implementation of an R*-tree for spatial data indexing, and BaseX XQuery system for keyword indexing. *PBD* uses the PostGIS system for both spatial data and keyword indexing (i.e., *R-Tree-over-GiST and B-tree*). In other words, previous *getLayerByName* and *getElementsbyKeyword* have both two implementations: one with Java and BaseX, and other one with PostGIS. In order to switch from an implementation of RLD/PBD to another, two namespaces are provided in XOSM, for instance, *xosm_rld:getLayerByName* and *xosm_pbd:getLayerByName*.
   The reasons for having two different implementations are the following. Firstly, *RLD* was firstly developed, and served as system test, and *PBD* was later integrated, enabling better performance results with large datasets. They are both maintained in the current release of XOSM. Secondly, *RLD* uses the *OSM API*[15] in order to retrieve OSM data. Therefore, *RLD* works with *live data* (i.e. latest version of OSM data), where *on-the-fly* indexing is carried out. In the case of *PBD*, it works with *backup data* (i.e., the entire planet version March 2017) indexed by PostGIS, and thus, it handles an obsolete version of OSM data. Most of existing OSM querying tools, including spatial extensions of SPARQL and PostGIS, work with backup data (i.e., the entire planet), but in order to have a better approximation to live data a periodic (conversion and) updating is carried out.

In summary, XOSM combines XQuery, Java and PostGIS code. XQuery is responsible for most of the tasks including *RLD* keyword indexing, while Java is used to implement the Java R*-tree of the *RLD* spatial indexing, and finally, *PostGIS* is used to implement the *PBD* spatial and keyword indexing. Our benchmark studies can be summarized as follows.

Live data performance: Firstly, we have analyzed the response time of *RLD* indexing strategy, in order to evaluate the Java R*-tree spatial indexing as well as the BaseX keyword indexing.

Live data versus Backup data: Secondly, we propose to compare backup data with PostGIS indexing (i.e., *PBD*) w.r.t. live data with Java R*-tree/BaseX indexing (i.e., *RLD*). While working with backup data (and thus previously indexed data) is obviously better than on-the-fly retrieval of data and indexing, our intention is to evaluate the size of layers for which the second option is still viable in terms of performance.

[13] http://docs.basex.org/wiki/Geo_Module
[14] http://expath.org/spec/geo
[15] http://wiki.openstreetmap.org/wiki/API_v0.6

On-the-fly retrieval of data has as advantage the handling of updated data.

XOSM versus PostGIS: Finally, whether XOSM with *PBD* is able to work at least with the same performance measures than PostGIS, which can be considered the benchmark to be reached. Existing LGOD query language implementations (*GeoSPARQL* and *stSPARQL*) are also built on top of PostGIS and thus similar performance is assumed. Here, we have compared a spatial query (i.e., streets crossing a given street), and a keyword query (i.e., hotels with at least two stars) in XOSM and PostGIS. Both systems share the same indexing mechanism but the filter processing (i.e., "crossing" and "at least two stars") is carried out by XOSM and PostGIS, respectively.

In summary, the experiments aim (1) to evaluate the performance of live data indexing, (2) the improvements of performance of backup data versus live data and (3) comparison of XOSM and PostGIS both with backup data. The case (3) shows in particular that XOSM is competitive with PostGIS even though XOSM is built on top of PostGIS and adds new processing after PostGIS-based layer retrieval.

Let us remark that the current paper is a continuation of previous works [20, 21, 22, 23] about OSM data query processing with XQuery. In [21] (and the extended version of [20]) the basis of OSM query processing with XQuery was stablished; in particular, a preliminary version of the XOSM XQuery library for the retrieval of layers with Boolean spatial/keyword operators was defined, together with a bath of query examples. In [21, 20] a naive indexing technique was introduced. In [23], distance based queries are handled with a extension of the library proposed in [21, 20]. In [22] another extension of the library has been introduced, included aggregation operators. Here, the main contribution falls on the handling of Linked Open Data and their integration with OSM data. Additionally, here data indexing is improved by proposing a more efficient mechanism based on PostGIS indexing.

1.1. Structure of the Paper

The rest of this article is organized as follows. Section 2 will present the basic elements of OSM, will define the query library. Section 3 will describe the transformation library. Section 4 will show the XOSM tool. Section 5 will present benchmarks for several datasets. Section 6 will compare with related work and finally, Section 7 will conclude and present the future work.

2. XOSM QUERY LIBRARY

OpenStreetMap uses a topological data structure which includes the following core elements: (1) *Nodes* which are points with a geographic position, stored as coordinates (pairs of a latitude and a longitude) according to WGS84. They are used in ways, but also to describe map features without a size like points of interest and mountain peaks. (2) *Ways* are ordered lists of nodes, representing a poly-line, or possibly a polygon if they form a closed loop. They are used in streets, rivers, etc., as well as areas: buildings, forests, parks, etc., (3) *Relations* are ordered lists of nodes, ways and relations. Relations are used for representing the relationship of existing nodes and ways. (4) *Tags* are key-value pairs (both arbitrary strings). They are used to store *metadata* about the map objects such as their type, their name and their physical properties. Tags are attached to a node, a way, a relation, or to a member of a relation.

As an example of OSM map, Figure 4 shows the visualization of a piece of Almería (Spain) city map. In order to represent a map, OSM uses XML labels: *node*, *relation* and *way*, and each label can have several attributes, for instance, *node* has *lat* and *lon*, among others, for representing latitude and longitude of the node. A node, representing a point of interest of the city, can have *tags* for adding information about the point, using attribute pairs key (k) and value (v) with this end. For instance, the museum "Museo Arqueologico" of Almería city is represented as shown in Figure 5(a). OSM main element is *way* that serves not only to represent streets but also buildings, parkings, etc. Ways are described by a sequence of node references, called *nd*, which link ways to nodes, and *tag*s (see Figure 5(b)). When the way is related to a building, park, etc., specific tags are used inside the way item (see Figure 5(c)). Finally, relations are used to relate elements of the map; for instance, bus routes (see Figure 5(d)). In spite of the simplicity of the XML representation of OSM, many features in a OSM layer can be described[16].

2.1. Query Library

Now, we will describe the elements of the spatial/textual OSM library together with some examples of use. A full list of operators together with formal definitions is given in the Appendix. Firstly, we will given some definitions.

Formally, given a set of label values $\mathcal{V}$, an *OSM map* $m = (\mathcal{S}, \mathcal{K})$ is a sequence $\mathcal{S} = \{s_1, \ldots, s_n\}$, where each s_i is an *OSM element* (a node, a way or a relation), together with a set of *key functions* $\mathcal{K}$ from $\mathcal{S}$ to $\mathcal{V}$. We assume that $\mathcal{K}$ contains at least the key function *name* providing a name to each OSM element of an OSM map. Relations are not considered in our approach and thus, OSM elements are restricted to the case of nodes and ways. A *node* n is a pair (p_1, p_2), where p_i is a real number, and a *way* w is a sequence of nodes $\{n_1, \ldots, n_n\}$ defining a poly-line or a polygon. p_1 of a node n is called *latitude* denoted by $lat(n)$, and p_2 is called *longitude* denoted by $lon(n)$. n_1 of a way w is denoted by $first(w)$, and n_n is denoted by $last(w)$.

[16] http://wiki.openstreetmap.org/wiki/Map_Features

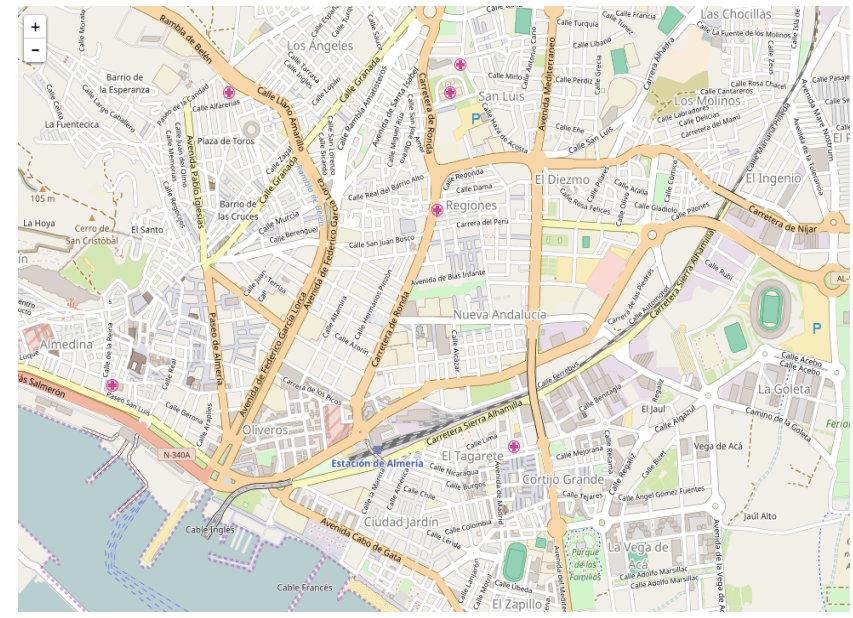

FIGURE 4. (Spain) Almería City Map

```
<node id="1568048883"
 lat="36.8386557"
 lon="-2.4556049">
 <tag k="name"
    v="Museo Arqueologico"/>
  <tag k="tourism" v="museum"/>
</node>
```

(a) Point in OSM

```
<way id="-3731">
  <nd ref="-3625" />
  <nd ref="-3623" />
  <nd ref="-3621" />
  <tag k="highway"
     v="residential"/>
  <tag k="name"
     v="Calle Calzada de Castro"/>
   <tag k="oneway" v="yes" />
</way>
```

(b) Street in OSM

```
<way id="27161540">
    <nd ref="298004115" />
    <nd ref="298004116" />
    <nd ref="298004119" />
    <nd ref="298004128" />
    <nd ref="298004115" />
    <tag k="amenity" v="parking"/>
</way>
```

(c) Parking in OSM

```
<relation id="147091">
    <member type="way" ref="27197940"
            role="3,11,12"/>
    <member type="way" ref="27197939"
            role="3,7,11,12"/>
    <member type="way" ref="35031199"
            role="3,11,12" />
    <member type="way" ref="27197944"
            role="7" />
    <member type="way" ref="27197945"
            role="7" />
<tag k="route" v="bus" />
<tag k="type" v="route" />
</relation>
```

(d) Bus route in OSM

FIGURE 5. XML representation of OSM data

We consider for each OSM element s, the minimum bounding rectangle (MBR), denoted by $mbr(s)$. We denote by $mindist(mbr, mbr')$ the minimum distance between the MBRs mbr and mbr'.

2.1.1. OpenStreetMap Index-based Functions

The following functions are used to retrieve elements of OSM maps. *getElementByName* retrieves an OSM element by name, that is, it returns, given an OSM map $m = (\mathcal{S}, \mathcal{K})$, the OSM element $s \in \mathcal{S}$ such as $name(s) = n$. *getLayerByName(m,n,d)* obtains given the name n of an OSM element, the OSM elements of the OSM map $m = (\mathcal{S}, \mathcal{K})$ at distance d of n. The same can be said for *getLayerByElement(m,e,d)*, but here an OSM element e is passed as argument. *getElementsByKeyword(m,k)* retrieves OSM elements of the OSM map $m = (\mathcal{S}, \mathcal{K})$ by keyword k. The keyword can be either the key function or the value. Finally, *getLayerByBB(m,mlat,Mlat,mlon,Mlon)* retrieves the OSM elements in a certain area of the OSM map $m = (\mathcal{S}, \mathcal{K})$ given by a bounding box *(mlat,Mlat,mlon,Mlon)*.

The indexing process is made following two strategies: R*-tree/BaseX on the fly indexing with live data (RLD) and PostGIS indexing with backup data (PBD), and the

implementation of previous functions is as follows:

- In RLD, a Java implementation of a R*-tree is used, and *getLayerByName* and *getLayerByElement* are implemented in Java, returning the elements at MBR distance *d*. *getElementsByKeyword(m,k)* is implemented in RLD in XQuery, using the BaseX index. Finally, *getLayerByBB* is implemented in RLD with a call to the OSM API.
- In PBD, *getLayerByName* and *getLayerByElement* are implemented in PostGIS and return the elements from a given distance *d*. *getElementsByKeyword(m,k)* and *getLayerByBB* are implemented in PBD in PostGIS, using the PostGIS index.

Our proposed query language mainly uses *getLayerByName*, i.e., queries have to be focused on a certain area of interest, given by the name of a node (park, pharmacy, etc.,), or by the name of a way (street, building, etc.,). Once the layer from the area of interest is retrieved, the repertoire of OSM operators in combination with higher order functions can be applied to produce complex queries. The answer of a query is an OSM layer including OSM elements of the area of interest. Nevertheless, *getElementsByKeyword* can be also used to retrieve OSM elements by keyword in a certain area. And also *getLayerByBB* can be used to retrieve all the elements enclosed by an area defined by a bounding box. In all the cases, the area is selected by the user (manually with the mouse or using the search text field in the Web tool).

2.1.2. OpenStreetMap Spatial Operators

We have considered two types of *Spatial Operators*: *Coordinate and Distance based OSM Operators* and *Clementini based OSM Operators*. Both kinds of spatial operators are designed to cover most of spatial queries involving nodes and ways. Our aim is to express queries related to the distance from points/streets/buildings, etc., and related to the localization of points/streets/buildings: north, points at east, etc.,; additionally, the street where a given point is located; if two points are located at the same street; the intersection point of two streets; and typical Clementini's spatial operators. In order to illustrate the spatial operators library we can consider the following query:

Example 1: Retrieve the streets in London intersecting *"Haymarket"* street and touching *"Trafalgar Square"*.

```
let $layer :=
 xosm_rld:getLayerByName(.,"Haymarket",0)
let $s :=
 xosm_rld:getElementByName(.,"Haymarket")
let $ts :=
 xosm_rld:getElementByName(.,"Trafalgar Square")
return
 fn:filter(fn:filter($layer,
  xosm_sp:intersecting(?,$s)),
       xosm_sp:touching(?,$ts))
```

In this query, the XQuery higher order function *filter* has been used twice in combination with the spatial operators *intersecting* and *touching*. Firstly, the streets next to *"Haymarket"* are retrieved by *getLayerByName* (at distance *0* meters). Next, the streets *"Haymarket"* and *"Trafalgar Square"* are retrieved by *getElementByName*. Finally, those ones intersecting *"Haymarket"* and touching *"Trafalgar Square"* streets are filtered.

2.1.3. OpenStreetMap Keyword Operators

Now, in order to express keyword-based queries, a repertoire of operators has been defined allowing to manipulate pairs k and v in OSM elements. For instance, *searchKeyword(s,kv)* checks the occurrence of a certain keyword kv as key function or value in an OSM element s. Analogously *searchKeywordSet(s,(*kv_1*, ..., *kv_n*))* for a set of kv's. In order to illustrate the keyword-based operators library we can consider the following queries:

Example 2: Retrieve the restaurants in Rome further north to *"Picasso"* hotel.

```
let $layer := xosm_rld:getLayerByBB(.)
let $hotel := xosm_rld:getElementByName(.,"Picasso")
return fn:filter(fn:filter($layer,
       xosm_kw:searchKeyword(?,"restaurant")),
       xosm_sp:furtherNorthPoints($hotel,?))
```

In this query, the layer selected by the Web tool is retrieved by *getLayerByBB* index-based function. Next, *"Picasso"* hotel is retrieved by *getElementByName*. Finally, the function *filter* is used twice to select restaurants further north than the hotel.

Example 3: Retrieve hotels of Vienna close to food venues.

```
for $hotel in
 xosm_rld:getElementsByKeyword(.,"hotel")
    [@type="point" or @type="area"]
let $layer :=
 xosm_rld:getLayerByElement(.,$hotel,200)
where
 count(fn:filter($layer,
 xosm_kw:searchKeywordSet(?,
   ("bar","restaurant"))))>=30
return $hotel
```

In this query, hotels close to food venues (i.e., *bars* and *restaurants*) are retrieved. Firstly, all the city hotels are selected by *getElementsByKeyword* and, next, the layer for each *hotel* is obtained (at distance *200* meters). Finally, the number of *restaurants* and *bars* occurring on each hotel layer are computed, and whenever this number is bigger than 30, the hotel is retrieved.

Example 4: Retrieve the hotels of Munich with the greatest number of churches nearby.

```
let $hotel :=
 xosm_rld:getElementsByKeyword(.,"hotel")
```

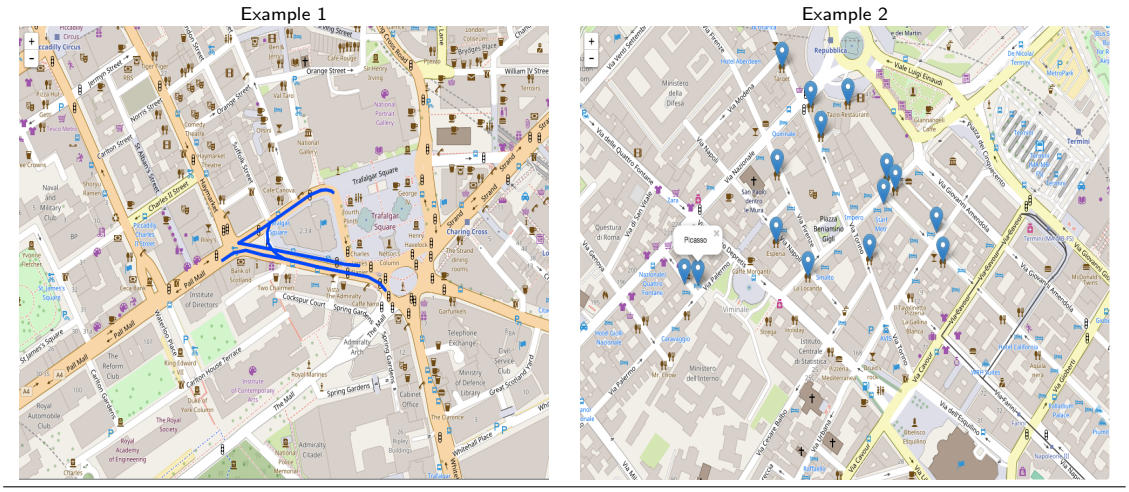

FIGURE 6. Results for OSM spatial operator based queries. Example 1: Retrieve the streets in London intersecting *"Haymarket"* street and touching *"Trafalgar Square"*; Example 2: Retrieve the restaurants in Rome further north to *"Picasso"* hotel.

```
let $f := function($hotel){
 -(count(fn:filter(
   xosm_rld:getLayerByElement(.,$hotel,100),
   xosm_kw:searchKeyword(?,"church"))))}
return fn:sort($hotel,$f)[1]
```

Finally, in this query, hotels with the greatest number of churches nearby are retrieved. With this aim, the XQuery higher order function *sort* is used in order to sort the hotels by the number of churches nearby (at distance *100* meters). An anonymous function is used in order to count churches in the layer of each retrieved hotel (at distance *100* meters).

2.1.4. OpenStreetMap Aggregation Operators

The proposed aggregation operators are inspired by the SOLAP operators. In [24], a taxonomy of operators whose result is numeric is considered[17]. They consider two levels of operators. The first level includes *numeric-spatial* and *numeric-multidimensional* operators. Numeric-spatial operators can be topological (Boolean Clementini's operators), and metric (*area*, *length* and *distance*), while numeric-multidimensional operators are *max*, *min*, *sum*, *count* and *distinct count*, among others. A second level is defined as combinations of operators of the first level. Numeric-multidimensional operators can be classified into *Distributive*, *Algebraic* or *Holistic* [25]. An aggregate function is *distributive* if it can be computed in a distributed manner, that is, the result derived by applying the function to the n aggregate values is the same as that derived by applying the function to the entire data set (without partitioning). An aggregate function is *algebraic* if it can be computed by an algebraic function with m arguments (where m is a bounded positive integer), each of which are obtained by applying a distributive aggregate function. Finally, an aggregate function is *holistic* when there does not exist an algebraic function with m arguments that characterizes the computation.

For instance, the distributive operators *metricMin(sq,m)* and *metricMax(sq,m)* return the objects of *sq* having the minimum, resp. maximum, value of a given metric operator m; and the distributive operator *metricSum(sq,m)* returns the result of adding the values of a given metric operator m in a sequence *sq*. In order to illustrate the aggregation operators library we can consider the following queries:

Example 5. This query requests the size of park areas close to *"Karl-Liebknecht-Strasse"* in Berlin. It is expressed as follows:

```
let $layer :=
 xosm_rld:getLayerByName(.,
   "Karl-Liebknecht-Strasse",350)
let $parkAreas :=
 fn:filter($layer,xosm_kw:searchKeyword(?,"park"))
return xosm_ag:metricSum($parkAreas,"area")
```

In this query, *filter* selects the *"park"*s from the *Karl-Liebknecht-Strasse* layer (at 350 meters), and the distributive operator *metricSum* computes the size of park areas.

Example 6. Next query requests the top-star rating biggest hotels close to *"Via Dante"* in Milan. This query can be expressed as follows:

```
let $layer :=
 xosm_rld:getLayerByName(.,"Via Dante",350)
let $hotels :=
 fn:filter($layer,
```

[17] In [24] they also consider *spatial operators* whose result is spatial (*ConvexHull*, *Envelope*, *Centroid*, *Boundary*, *Intersection*, *Union*, *Difference* and *Buffer*). They also consider navigation and temporal operators.

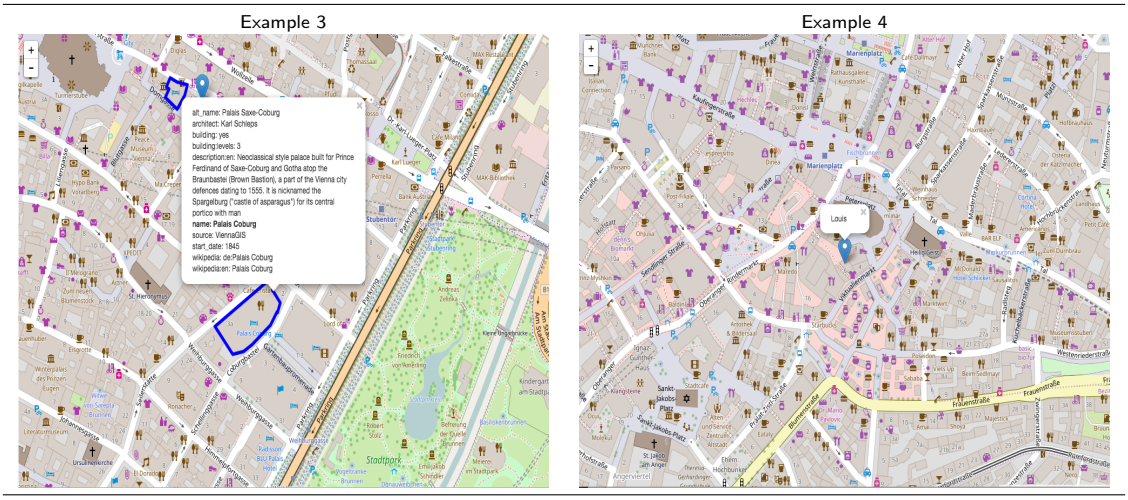

FIGURE 7. Results for OSM Keyword operator based queries. Example 3: Retrieve hotels of Vienna close to food venues; Example 4: Retrieve the hotels of Munich with the greatest number of churches nearby.

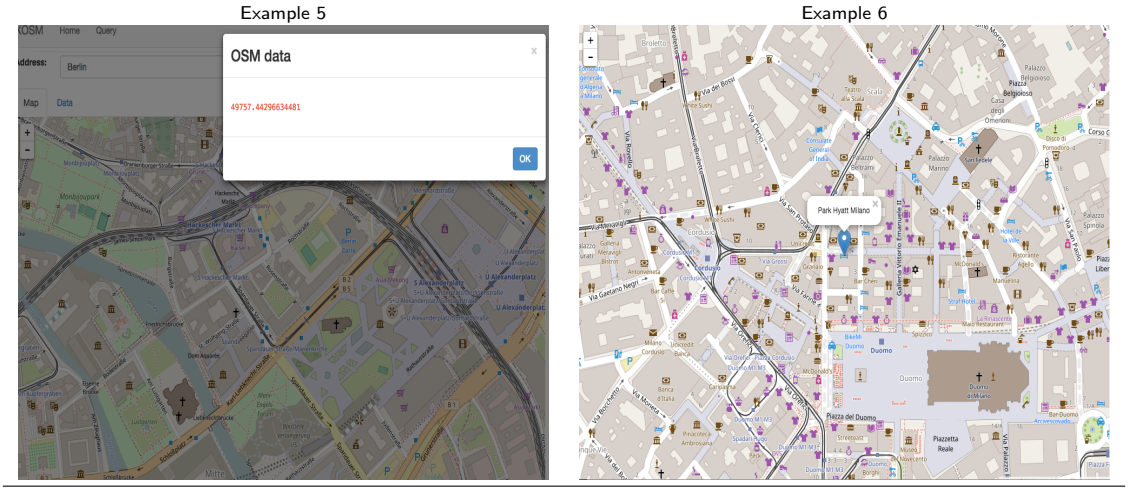

FIGURE 8. Results from OSM Aggregation operator based queries. Example 5: Size of park areas close to *"Karl-Liebknecht-Strasse"* in Berlin; Example 6: Top-star rating biggest hotels close to *"Via Dante"* in Milan.

```
  xosm_kw:searchKeyword(?,"hotel"))
return xosm_ag:metricMax(
 xosm_ag:metricMax($hotels,"stars"),"area")
```

In this case, *filter* is used to search the keyword *"hotel"*, and next the distributive operator *metricMax* is used twice in order to compute the top-star rating and biggest hotels.

Let us remark that our proposal of aggregation operators is richer than the proposed in [24]. In all cases of metric operators, functions passed as arguments can be one of *area*, *length* and *distance*, as well as any function which computes numeric values from OSM elements. This is the case of *metricMin*, *metricMax*, *metricSum*, *metricAvg*, *metricStdev*, *metricTopCount*, *metricBottomCount*, *metricMedian*, *metricMode* and *metricRank*. Additionally, operators *metricMin* and *metricMax* which return an unique value in [24], can here return more than one value. There *metricMin* and *metricMax* are used for area and length which rarely are equal for more than one element. Here, *metricMin* and *metricMax* can be applied to any operator returning a numeric value, for instance, number of stars of hotels, which can be the same for several hotels. Finally, *metricMode* can be applied to any operator, not only numeric.

Finally, let us remark that since we are working with XQuery in order to list other data items (coordinates, key-value pairs, etc.,) from OSM documents, XQuery mechanisms can be used, in particular, using XPath on a node n, n/@lat, n/@lon returns the coordinates of the node. In the case of a way w, w/nd/@ref is the set of nodes of a way. With regard to key-value pairs, they can

be retrieved by n/tag/@k and n/tag/@v or w/tag/@k and w/tag/@v.

2.1.5. Some Implementation Details

The implementation of an OSM query library is away to be a trivial task. The OSM initiative aims that any user can edit maps. Unfortunately, due to contributions of multiple users main tags are not always filled, and sometimes query answers are further than the user expects. The same happens with ways editing by users, in which the lack of precision in geometries can lead to wrong spatial objects. On the other hand, OSM handles only two geometries (nodes and ways). Firstly, streets are defined by segments in which street directions are defined by the order of segment points. Thus, the full geometry of a street is composed by several ways and, the set of points is not necessarily in geographical order. Secondly, buildings (and, in general, polygons) are defined as closed ways: ways in which the end point is equal to the start point. Thus, streets and buildings have the same kind of representation. It makes more sophisticated to detect spatial objects and their relationships. Additionally, we have found that allowing several layers, the definition of the library becomes even more complex. In a unique OSM layer, usually ways do not overlap. Except when areas, metro lines, etc., are defined. When several layers involving areas are considered, overlapping arises, and thus ways can share nodes. We have followed a *name-based approach* to handle OSM maps. We have decided to adopt names as mechanism to identify spatial objects. Thus, objects without name are ignored. Names, for instance, are used to collect segments of streets in order to build the full geometry.

We have defined functions in XQuery to transform OSM geometries into GML geometries, which is required by the EXPath/JTS library. This is a key point of the implementation due to the following reasons: (1) ways are used in OSM to represent both LineStrings and Polygons; (2) streets are built from segments, each one represented by a way; (3) buildings (and other areas) are built as closed ways. Thus, in order to transform from OSM geometries to GML geometries, we have to: (1) distinguish cases in the form of ways: closed ways and open ways; (2) collect ways of the same street. This transformation enables, for instance, to implement *wayIn* as follows:

```
declare function xosm_sp:wayIn($node as node(),
     $way as node())
{
if ($node/way) then false()
else
let $p := xosm_gml:_osm2GmlPoint($node/node/@lat,
     $node/node/@lon)
let $l := xosm_gml:_osm2GmlLine($way)
return geo:distance($l,$p)=0
};
```

EXPath/JTS library is also used to implement spatial operators. For instance, *crossing* is defined as follows using a higher order Boolean pattern query *booleanQuery*:

```
declare function xosm_sp:crossing($e1 as node(),
     $e2 as node())
{
 xosm_sp:booleanQuery($e1,$e2,"geo:crosses")
};
```

XQuery is equipped with a rich query language including *some* and *every* statements. It makes possible to implement *searchKeywordSet* as follows:

```
declare function xosm_kw:searchKeywordSet(
    $e as node(),$ks as xs:string*)
{
some $kw in $ks
 satisfies xosm_kw:searchKeyword($e,$kw)
};
```

wherein *searchKeyword* is defined as follows:

```
declare function xosm_kw:searchKeyword(
    $e as node(),$kw as xs:string)
{
let $item := $e//*[name(.)="tag"]
return
(some $att in $item/@v satisfies ($att = $kw)) or
(some $att in $item/@k satisfies ($att = $kw))
};
```

Higher order is also used in the implementation of aggregation operators. For instance, *topologicalCount* is implemented as follows:

```
declare function xosm_ag:topologicalCount(
   $sq as node()*,$e as node(),$b as xs:string)
{
count(fn:filter($sq,function($eRef){
     xosm_sp:booleanQuery($eRef,$e,$b)}))
};
```

using the higher order function *filter*, *count* as well as *booleanQuery*. Finally, *metricBottomCount* has been defined in terms of the higher order function *sort*, the keyword operator *getTagValue*, the built-in XQuery function *subsequence*, and an auxiliary function *metricList*:

```
declare function xosm_ag:metricBottomCount(
   $sq as node()*,$m as xs:string,$k as xs:integer)
{
let $list := xosm_ag:metricList($sq,$m)
return
fn:subsequence(fn:sort($list,function($w){
  xosm_kw:getTagValue($w,$m)}),1,$k)
};
```

Now, we will define the transformation library of XOSM together with some examples of use.

3. XOSM TRANSFORMATION LIBRARY

XOSM is also equipped with functions enabling the retrieval and transformation of Linked Geo Open Data (see Table 2). Linked Geo Open Data can have KML, GeoJSON, CSV, RDF (and also XML).

However, the conversion to OSM format is a non-trivial task. Firstly, the main information provided from spatial datasets is the spatial location of objects. The way in which they represent the location of spatial objects can vary from one format to another. For instance, GeoJSON represents spatial objects by Point, LineString and Polygon geometries as features. This is not the case of OSM, in which a simpler representation by nodes (and list of node references) is used. In the case of KML, Placemark is used to store spatial objects, and again Point, LineString and Polygon are used to distinguish object geometry. Key-value pairs of OSM data are extracted from GeoJSON from object properties. In CSV, columns are transformed into key-value pairs. To extract coordinates of points from CSV, the transformation function requires to pass as argument the name of the columns including the point coordinates. The case of RDF is less sophisticated requiring to extract geo:lat and geo:long RDF properties to locate spatial objects.

In this context, the function *json* can be used to transform GeoJSON data into OSM data. It transforms GeoJSON geometries into OSM geometries, as well as it adds GeoJSON textual data into OSM key-value pairs. It has as parameters, the *url* of the resource and the *name* of the label in the given GeoJSON resource representing the name of the spatial object. It serves to map GeoJSON names into OSM names. Additionally, it has four parameters *kp,vp,kw* and *vw*. The transformation from GeoJSON to OSM assigns to points the key-values pairs *(kp,vp)* and to ways the key-values pairs *(kw,vw)*. It permits, for instance, to map restaurants represented by GeoJSON into OSM ("amenity","restaurant"), as well as to map parkings represented by GeoJSON into OSM ("amenity","parking"). The function *kml* has similar behavior.

Function *csv* can be used to transform CSV data. In this case, the parameters specify the column name containing the *name* of the spatial object as well as the column names where latitudes and longitudes are specified. CSV only works for points. The rest of columns are transformed into key-value pairs.

RDF data and, in particular, *Wikipedia* resources, can be accessed and integrated to OSM data with three functions: *wiki_element*, *wiki_coordinates* and *wiki_name*. The functions for Wikipedia work with spatial coordinates. With this aim the service *Geonames*[18] is used. This service provides information about places nearby to given coordinates, as well as the corresponding *DBpedia* links for these places. Thus, the previous function *wiki_coordinates* has as parameters the spatial coordinates, and it calls to *Geonames* service in order to retrieve information of DBpedia from the links. The information extracted from DBpedia is transformed into key-value pairs of OSM. Function *wiki_element* is similar to *wiki_coordinates*, but it has as parameter an OSM element. Function *wiki_name* is similar but having as parameter an OSM layer and the name of an element of the layer.

With regard to the XML format, *Tixik* API[19] allows to retrieve information about famous places around the world in XML format. Given the spatial coordinates, the API retrieves names, descriptions, images, and links to *Tixik.com* of nearby places. In this case, the XQuery functions *tixik_name*, *tixik_coordinates* and *tixik_element* transform these data provided by Tixik into OSM data (coordinates and key-value pairs) similarly to Wikipedia functions. In order to illustrate the transformation library we can consider the following queries:

Example 7: The following query requests taxi stops close to "Carrousel du Louvre" in Paris. Taxi stops are retrieved from the LOD service of Paris[20], and *DWithIn* is used in order to filter those ones at distance 500 meters.

```
let $taxis := xosm_open:json("https://opendata.paris.fr/
     explore/dataset/paris_taxis_stations/download/?
     format=geojson&amp;timezone=Europe/Berlin",
     "address","amenity","taxi","","")
let $building :=
 xosm_rld:getElementByName(.,"Carrousel du Louvre")
return
fn:filter($taxis,xosm_sp:DWithIn($building,?,500))
```

Example 8: The following query retrieves free events of Madrid. The events are retrieved from the LGOD ARCGIS site[21], and filtered by an anonymous function selecting free events[22].

```
let $events :=
     xosm_open:json("http://data2.esrism.opendata.arcgis.
     com/datasets/51900577e33a4ba4ab59a691247aeee9_0.
     geojson","EVENTO","place","*","area","yes")
return fn:filter($events,function($p)
    {not(empty($p/node/tag[@k="GRATUITO" and @v="Si"]))})
```

Example 9: The following query uses the function *wiki_element* to retrieve Wikipedia information about places nearby to the intersection point of *"Calle Mayor"* and *"Calle de Esparteros"* in Madrid.

```
let $x := xosm_rld:getElementByName(.,"Calle Mayor")
let $y :=
 xosm_rld:getElementByName(.,"Calle de Esparteros")
return
for $i in xosm_sp:intersectionPoints($x,$y)
return xosm_open:wiki_element($i)
```

Example 10: Finally, the following query retrieves the information provided by *tixik.com* around *"Picadilly"* in London.

[18] http://www.geonames.org/

[19] http://www.tixik.com/info/api/

[20] http://opendata.paris.fr/

[21] http://opendata.arcgis.com/

[22] "GRATUITO" means free in spanish.

XQuery Function	*LOD format*
json(url,name,kp,vp,kw,vw)	GeoJSON
kml(url,name,kp,vp,kw,vw)	KML
csv(url,name,lat,lon)	CSV
wiki_element(osm)	RDF
wiki_coordinates(lat,lon)	RDF
wiki_name(layer,name)	RDF
tixik_element(osm)	XML
tixik_coordinates(lat,lon)	XML
tixik_name(layer,name)	XML

TABLE 2. XQuery Functions for Linked Open Data

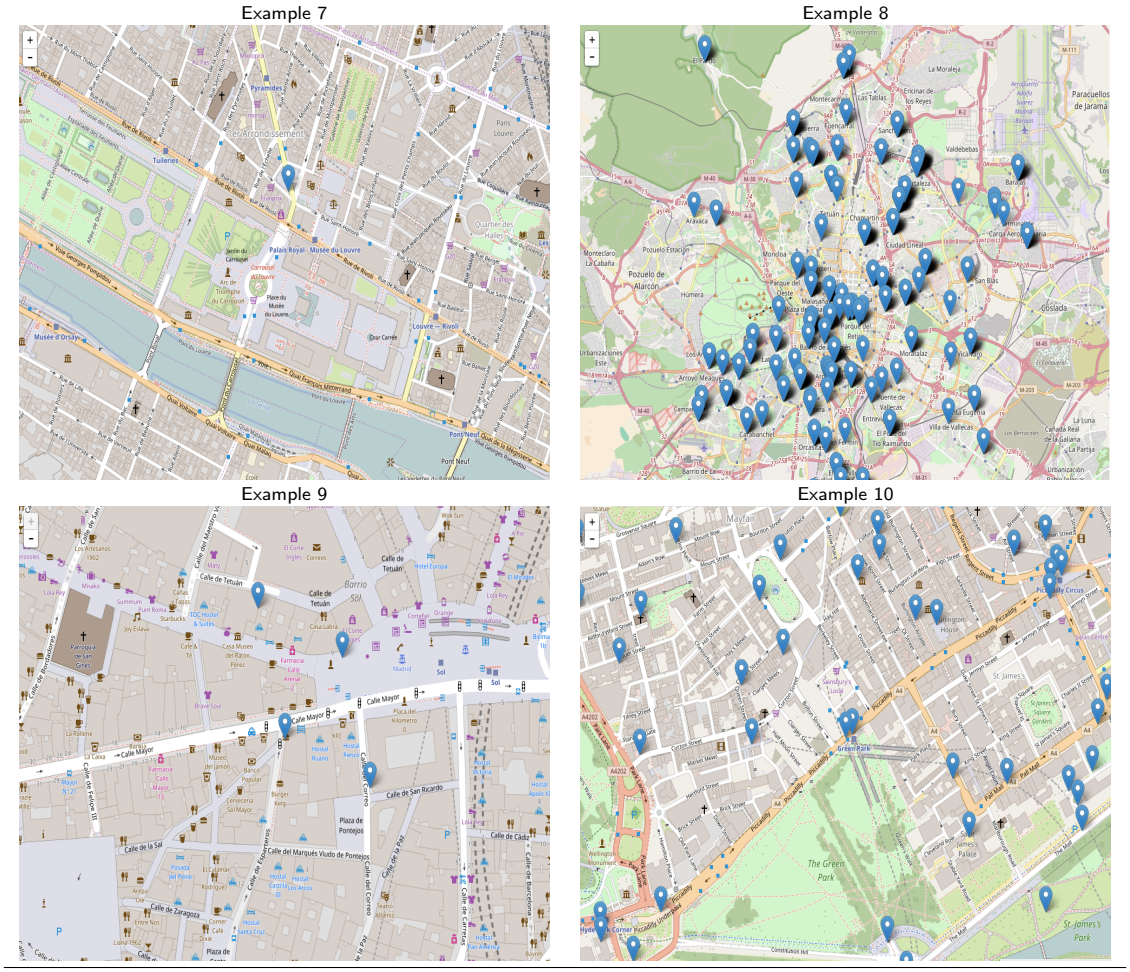

FIGURE 9. Results from Linked Open Data based queries. Example 7: Taxi stops close to "Carrousel du Louvre" in Paris; Example 8: Free events of Madrid; Example 9: Wikipedia information about places nearby to the intersection point of *"Calle Mayor* and *"Calle de Esparteros"* in Madrid; Example 10: Information provided by *tixik.com* around *"Picadilly"* in London.

```
let $x := xosm_rld:getElementByName(.,"Piccadilly")
return xosm_open:tixik_element($x)
```

4. XOSM TOOL

In this section, XOSM (XQuery for OpenStreetMap) Web tool developed by our group, hosted at the url http://xosm.ual.es/XOSM, is presented. The Web-based tool permits to query OSM maps (and LGOD resources), enabling the selection of an area of the OSM map, and the execution of queries from the XQuery shell (see Figure 4). Additionally, the tool is equipped with a batch of pre-defined spatial, keyword, aggregation and LGOD queries (among them, the included in this paper). XOSM highlights the results of queries with a different color and an icon for ways and nodes,

respectively (see Figure 4). In the case of the result is a value (text or numeric) XOSM visualizes the result in a pop-up window.

4.1. XQuery API

The methods of the XQuery API for OSM are shown in Table 3. The API is built on top of the PBD (for getting short answer times), and enables to query OSM data by the operators: *getElementByName*, *getElementsByKeyword*, *getLayerByName*, *getLayerByElement* (here the coordinates are specified) and *getLayerByBB*. XQuery queries can be directly executed from the API, using the method *Query*. The API permits to pass the bounding box of the map to be queried. In the Web tool this bounding box is replaced by the selected area.

For instance, the following API request, executes the following query: *Retrieve hotels at 50 meters of "Trafalgar Square" in London*:

http://xosm.ual.es/xosmapi/Query/minLon/-1.0135918/minLat/51.0900174/maxLon/0.8389607/maxLat/51.8863973?query=
let $x := xosm_pbd:getLayerByName(.,"Trafalgar Square", 50) return filter($x,xosm_kw:searchKeyword(?,"hotel"))

4.2. An Application with XOSM

Now, an example of application with the XOSM library is shown. This application retrieves routes between streets of a given city. The application shows that the recursive capabilities of XQuery enable to implement without effort a path recursive algorithm[23] allowing to retrieve the routes. A piece of the code in shown below, and the depicted result in the Web tool can be seen in Figure 11. The route of Figure 11 is computed from Wollzeile to Lichtensteg streets of Vienna.

```
declare function xosm_sp:path(
     $layer,$name1,$name2,$dlayer)
{
if ($name1=$name2) then
 xosm_sp:print_already_here()
else
let $e1 :=
 xosm_rld:getElementByName($layer,$name1)
let $e2 :=
 xosm_rld:getElementByName($layer,$name2)
return
if (empty($e1)) then xosm_sp:print_out($name1)
else if (empty($e2)) then xosm_sp:print_out($name2)
else
let $layer1 :=
 xosm_rld:getLayerByName($layer,$name1,
     $dlayer)[@type="way"]
return
xosm_sp:write_path(
   xosm_sp:path_aux($layer1,$e1,$e2,$e1))
};

declare function xosm_sp:path_aux(
     $layer,$e1,$e2,$path)
{
if (empty($layer)) then ()
else
 if (xosm_sp:getDistance($e1,$e2)=0)
     then ($path,$e2)
     else
     let $can :=
          fn:filter($layer,xosm_sp:DWithIn(?,$e1))
     return
           xosm_sp:loop_path($can,$layer,$e2,$path)
};

declare function xosm_sp:loop_path(
     $can,$layer,$e2,$path)
{
if (empty($can)) then ()
else
let $c := head($can)
let $p := xosm_sp:path_aux($layer
         [every $name in data(($path,$c)/@name)
          satisfies not($name=data(@name))],
          $c,$e2,($path,$c))
return
if (empty($p))
then xosm_sp:loop_path(tail($can),
          $layer,$e2,$path)
else $p

};
```

[23]The algorithm recursively finds one of the paths between streets, not necessarily the shortest path.

The route application uses the spatial operators *DWithIn*, *furtherNorthWays*, *furtherNorthEast*, etc., It returns a string in which the route is described in terms of streets to be followed and directions to be taken (i.e., north, east, west, east).

5. BENCHMARKS

In this section, we show the benchmarks of the proposed tool. We analyze the following cases:

(5.1) **Live data performance**: We analyze the response time for the retrieval of layers using distances and keywords (i.e., *getLayerByName* and *getElementsByKeyword*) (Section 5.1).
(5.2) **Live data versus Backup data.** We compare the response time for query answering of the examples shown (Section 5.2).
(5.3) **XOSM versus PostGIS**. We compare the response time on spatial and keyword queries (Section 5.3).

For this benchmarking, we have used a HP Proliant (one quad core and 12GB RAM Memory) with Ubuntu Server (version 16.10). For the implementation, we have used the BaseX XQuery processor (version 8.3) and the PostGIS system over PostgreSQL (version 9.5).

5.1. Live data performance

The goal of these benchmarks is to show the response time of RLD for the retrieval of layers by distances and keywords. We have used the datasets of Figure 12. We will consider the following benchmark measures:

(1) **Spatial Indexing and Layer Retrieval** (see Figure 13); i.e., the time required to generate a spatial index (R*-tree structure) and to retrieve a certain layer by name (i.e., *getLayerByName*). Times are taken in *seconds* and they are computed with distance values *0*, *100* meters and *1* km for all datasets. Figure 13 shows that *getLayerByName* in

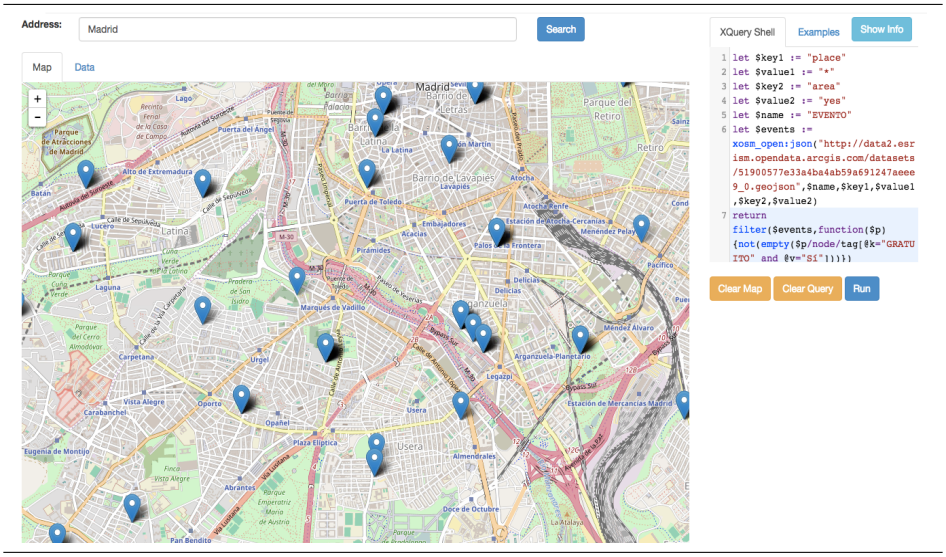

FIGURE 10. XOSM: XQuery Shell

GetLayerByName http://xosm.ual.es/xosmapi/getLayerByName/minLon/{minLon}/minLat/{minLat} /maxLon/{maxLon}/maxLat/{maxLat}/name/{name}/distance/{distance}
GetLayerByElement http://xosm.ual.es/xosmapi/getLayerByElement/minLon/{minLon}/minLat/{minLat} /maxLon/{maxLon}/maxLat/{maxLat}/lon/{lon}/lat/{lat}/distance/{distance}
GetElementByName http://xosm.ual.es/xosmapi/getElementByName/minLon/{minLon}/minLat/{minLat} /maxLon/{maxLon}/maxLat/{maxLat}/name/{name}
GetElementsByKeyword http://xosm.ual.es/xosmapi/getElementsByKeyword/minLon/{minLon}/minLat/ {minLat}/maxLon/{maxLon}/maxLat/{maxLat}/keyword/{keyword}
GetLayerByBB http://xosm.ual.es/xosmapi/getLayerByBB/minLon/{minLon}/minLat/{minLat} /maxLon/{maxLon}/maxLat/{maxLat}
Query http://xosm.ual.es/xosmapi/Query/minLon/{minLon}/minLat/{minLat}/maxLon/ {maxLon}/maxLat/{maxLat}?query={query}

TABLE 3. XQuery API

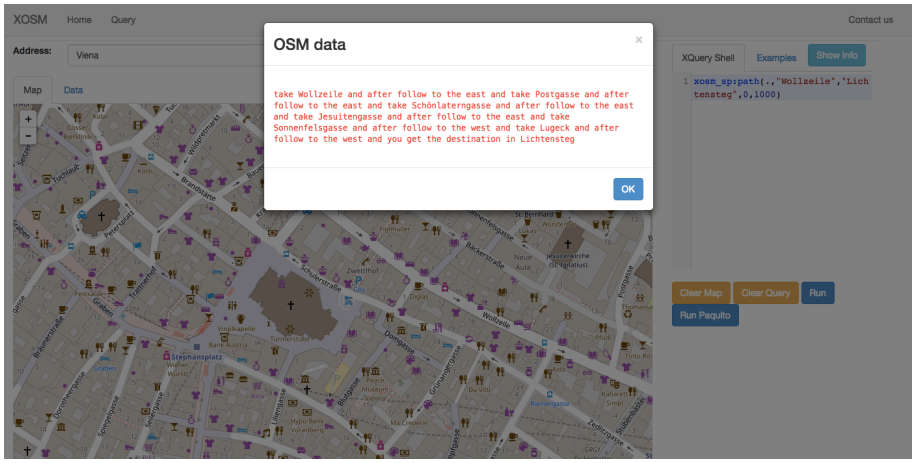

FIGURE 11. Route from Wollzeile to Lichtensteg streets of Vienna.

RLD has a reasonable response time in small and medium-size layers, and it gets worse (120 seconds in the worst case (1km)), for a dataset of around 120 MB, with one million of OSM elements. RLD

Almería	(19,929 Objects: 2,698 Ways + 17,231 Nodes (1,6MB)),
Alexandria (Greece)	(3,866 Objects: 5,495 Ways + 33,171 Nodes (4,1MB)),
Santa Barbara (USA)	(49,287 Objects: 4,244 Ways + 45,583 Nodes (6MB)),
Alburquerque (USA)	(102,620 Objects: 17,783 Ways + 88,837 Nodes (12MB)),
Cusco (Peru)	(168,048 Objects: 7,019 Ways + 161,029 Nodes (16MB)),
Cork (Irland)	(207,357 Objects: 21,258 Ways + 186,099 Nodes (20MB)),
Waterloo (Canada)	(559,754 Objects: 61,929 Ways + 497,825 Nodes (63MB))
Brisbane (Australia)	(1,034,520 Objects: 131,042 Ways + 903,477 Nodes (115MB))

FIGURE 12. Datasets for RLD-based retrieval of layers by distances and keywords

response time includes R*-tree on the fly indexing, and retrieval of the layer using the index. Let us remark that with the same layer (i.e., one million of OSM elements), the PBD strategy returns the same result in 6,6 seconds.

(2) **Textual Indexing and Keyword Retrieval** (see Figure 13); i.e., the time required to retrieve the set of OSM elements annotated with a certain keyword by using the BaseX textual indexing (i.e., *getElementsByKeyword*). Times are taken in *seconds*, and they are computed for two keywords: *hotel* and *park*. Response times range from less than one second to around 1.8 seconds for a dataset of around 120 MB, with one million of OSM elements. Response times for the keyword *hotel* are better than for the keyword *park*, given that the number of *park*s is (in all the cases) greater than the number of *hotel*s.

5.2. Live data versus Backup data

We have considered the following benchmark measures:

(1) **Query answering time** (see Figure 15) for the given **Example 1** to **Example 10**. Response times are measured in *milliseconds*. We have used the datasets of Figure 14. We have to take into account the following considerations:
Here, the datasets have been selected to cover the city center. Response times are in most of the cases less than 2 seconds (for both PBD and RLD). The only case exceeding this quantity is **Example 2**. In **Example 2**, *getLayerByBB* is used. In the case of RLD strategy, this function makes a call to the OSM API, retrieving all the OSM elements inside a bounding box. The answer time of this function depends on the answer time of the OSM API, and the number of elements to be retrieved. In the case of PBD, *getLayerByBB* uses the spatial PostGIS index, having a response time considerably better: 1,206.5 ms.

(2) **Query answering time** (see Figure 15) for **Example 1**. We have used the following datasets: *London*: (21,598 Objects : 2,261 Ways + 19,337 Nodes), (43,914 Objects : 5,193 Ways + 38,721 Nodes), (80,313 Objects : 5,937 Ways + 74,376 Nodes), (114,877 Objects : 10,241 Ways + 104,636 Nodes), (202,564 Objects : 20,304 Ways + 182,226 Nodes), 303,907 Objects : 36,877 Ways + 267,030 Nodes).
Here our intention is to measure how is better PBD and RLD. As we can see for small and medium-size areas, less than one hundred thousand objects, PBD and RLD response times are similar. However, for bigger areas PBD is better than RLD.

5.3. XOSM versus PostGIS

We have considered the following datasets: *Almería (Spain)* (43,914 Objects: 5,193 Ways + 38,721 Nodes (10MB)), *Barcelona (Spain)* (949,458 Objects: 33,459 Ways + 915,999 Nodes (190MB)), *Zurich (Switzerland)* (2,438,759 Objects: 234,876 Ways + 2,203,883 Nodes (435MB)) and *London (UK)* (4,041,926 Objects: 598,841 Ways + 3,443,085 Nodes (832MB)).

We have considered the following benchmark measures:

(1) **Spatial query crossing for XOSM PBD strategy and PostGIS** (see Figure 16), i.e., time for spatial indexing, and retrieval of OSM elements holding a spatial relation (crossing) in a certain area. Here our intention is to compare XOSM and PostGIS for spatial operations. XOSM is built in the PBD strategy on top of PostGIS, but PostGIS is only used to retrieve layers (by distance and keyword). In the PBD strategy, when a query involves to filter the elements of the layer, for instance, the retrieval of the streets crossing another street, then XOSM (and XQuery) filters the streets retrieved by PostGIS. Thus, we are interested in analyzing whether the same query (retrieval and filtering) in PostGIS and XOSM have similar response time. We have analyzed two cases: (1) the dataset is loaded by first time (i.e., non-cached data); and (2) the dataset has been already loaded (i.e., cached data). We can see that both systems XOSM PBD and PostGIS have similar behavior with response times in the worst case of around 100 seconds for (1) and 3.5 seconds for (2) with a dataset of 832 MB.

(2) **Keyword query hotel for XOSM PBD strategy and PostGIS** (see Figure 16), i.e., time for textual indexing, and retrieval of OSM elements

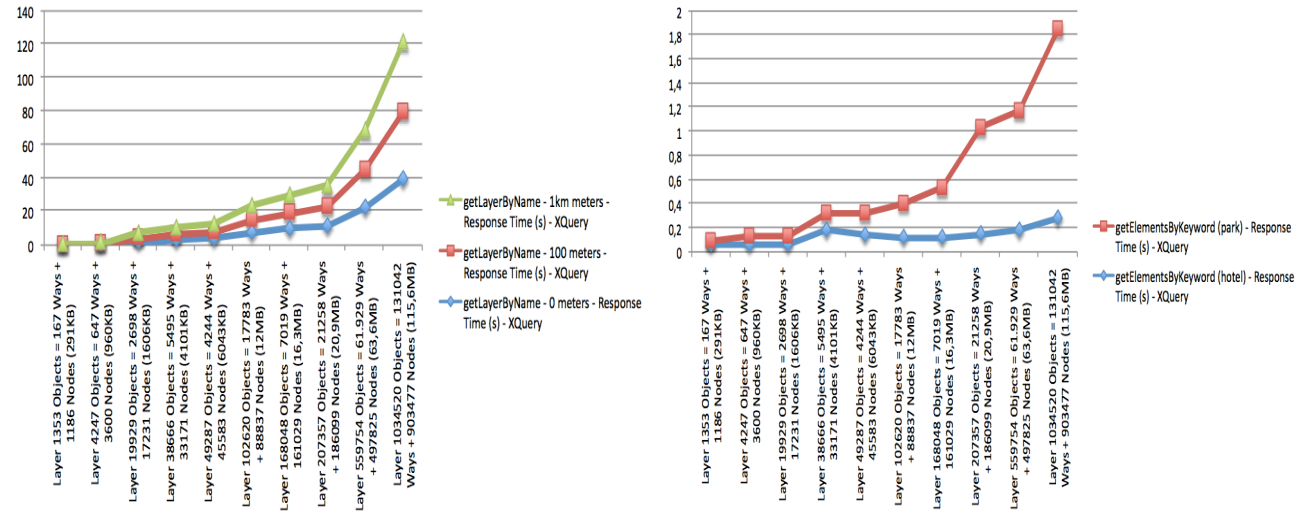

FIGURE 13. Java R*-tree/BaseX indexing: *getLayerByName* Response Time (Layer Retrieval for *0* meters, *100* meters and *1* km); *getElementsByKeyword* Response Time (keywords *hotel and park*).

Example 1	*London (UK)*	(7,315 Objects : 860 Ways + 6,310 Nodes)
Example 2	*Rome (Italy)*	(30,912 Objects: 5,993 Ways + 25,063 Nodes)
Example 3	*Vienna (Austria)*	(11,805 Objects: 1,164 Ways + 10,431 Nodes)
Example 4	*Munich (Germany)*	(8,124 Objects: 1,243 Ways + 6,708 Nodes)
Example 5	*Berlin (Germany)*	(18,799 Objects : 2,362 Ways + 16,146 Nodes)
Example 6	*Milan (Italy)*	(27,267 Objects : 3,857 Ways + 23,022 Nodes)
Example 7	*Paris (France)*	(35,076 Objects : 3,766 Ways + 30,846 Nodes)
Example 8	*Madrid (Spain)*	(14,073 Objects : 1,771 Ways + 12,025 Nodes)
Example 9	*Madrid (Spain)*	(14,073 Objects : 1,771 Ways + 12,025 Nodes)
Example 10	*London (UK)*	(12,276 Objects : 1,705 Ways + 10,453 Nodes)

FIGURE 14. Datasets for Query Answering Time

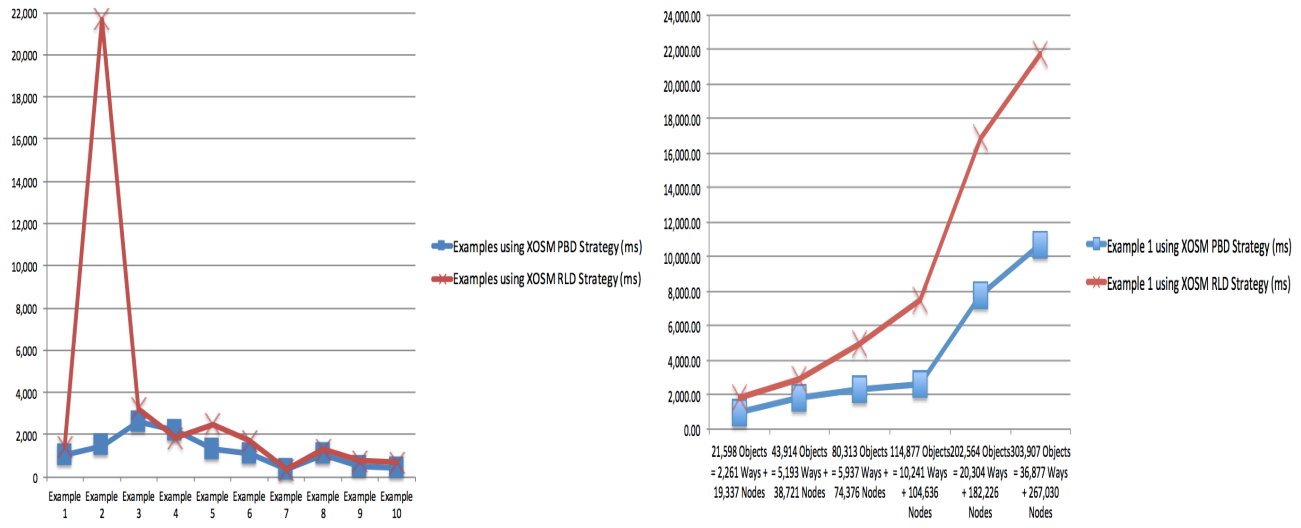

FIGURE 15. Live data versus Backup data: Examples 1 to 10 by RLD and PBD strategies; Example 1 in RLD and PBD strategies increasing size of layers.

annotated by a keyword (hotel) in a certain area. Here our intention is to compare XOSM and PostGIS for keyword retrieval. Similarly to the previous case, when a query involves to filter the elements of a layer by keyword, for instance, "hotel with at least two stars" then in the PBD strategy, XOSM (and XQuery) filters the elements retrieved by PostGIS keyword indexing. Thus, we are interested in analyzing whether the same query (retrieval and filtering) in PostGIS and XOSM have similar response time. As the previous case, we have analyzed two cases: (1) Dataset loaded by first time and (2) dataset already loaded. We can see that both cases have similar behavior, obtaining response times in the worst case of 90 seconds for (1) and 0.5 seconds for (2) with a dataset of 832 MB.

As conclusion, we can make the following considerations. While live data performance (i.e., XOSM RLD strategy) can be considered reasonable with small and medium-size layers, the response time of layer retrieval for a dataset of around 120 MB, with one million of OSM elements is 120 seconds for 1km, which can be considered too much. However layer retrieval from backup data (i.e., XOSM PBD strategy) is considerably lower. And it is even lower for cached results. The same layer is retrieved by PBD in 6.6 seconds without catching. In addition, such as shown in Figure 15, for less than one hundred thousand objects PBD and RLD response times of query processing are similar, getting worst results for instance in **Example 1** from 80,313 objects and beyond. However, RLD strategy has as advantage to work with updated data, and when it is used for small and medium-size layers, it has still a reasonable response time.

On the other hand, XOSM is competitive with PostGIS in both spatial and keyword queries. Let us remark that XOSM has an advantage against PostGIS. XOSM query results are given in OSM format (downloadable from the Web site and returned by the XOSM API), and thus enabling direct import of results to other OSM tools. However, PostGIS offers results in a table-like format, and external tools (for instance, Osmosis[24]) are required to obtain data in OSM format. It limits the interoperability of PostGIS with OSM tools. Thus, XOSM main advantage (i.e., on-the-fly integration of LGOD data) as well as the OSM format of XOSM results can motivate the use of XOSM by the OSM community.

6. RELATED WORK

6.1. XQuery and Spatial Data

GQuery [26] is an early proposal for adding spatial operators to XQuery. Manipulation of XML trees is carried out by XQuery, while spatial processing is performed using geometric functions and *JTS*. *GeoXQuery* approach [27] extends the *Saxon* XQuery processor [28] with functions that provide spatial operations. It is also based on *JTS* and a GML to SVG transformation is defined in order to show query results. *GML Query* [29] stores GML documents in a spatial RDBMS. This approach performs a simplification of the GML schema to be mapped to a relational schema. The basic values of spatial objects are stored as values of the tables. Once the document is stored, spatial queries can be expressed using the XQuery language with spatial functions. Queries are translated to their equivalent in SQL which are executed by the spatial RDBMS. Some modern XQuery implementations handle spatial data in GML format. This is the case of *eXist-db*[25], *BaseX*[26] and *MarkLogic*[27]. *MarkLogic* has limited spatial capabilities enabling point query –matches a single point–, box query –any point within a rectangular box–, radius query –any point within a specified distance around a point–, polygon query –any point within a specified n-sided polygon–, and some distance based queries. *BaseX*, in which XOSM is built, handles GML with the EXPath/JTS library[28]. *eXist-db* handles a JTS-like spatial library on GML data. XOSM is specifically designed for OSM, while the quoted approaches are focused on GML. XOSM is also able to handle Linked Geo Open Data in GeoJSON, KML, CSV and RDF formats. XOSM is also equipped with an API and a Web tool.

6.2. OSM APIs

Now, we summarize OSM based tools of the literature (see Table 6.1). Most tools are able to query OSM with simple commands: keyword and name searching. This is the case of *JOSM* and *Xapiviewer*. The *OSM Extended API or XAPI* is an extended API that offers queries in OSM with XPath flavoring. The *Overpass API (or OSM3S)*[30] is an extension of the API to select parts of an OSM layer. *OSM3S* has a proper query language which can be specified by an XML template. *OSM3S* offers more sophisticated queries than *XAPI*, but it is still a rather limited query language. *OSM3S* is specifically designed for searching location, types of objects, keywords, proximity or combinations of them. *Overpass API* is equipped with the query languages Overpass XML and Overpass QL. Both languages are equivalent. They handle OSM objects ((a) standalone queries) and set of OSM objects ((b) query composition and filtering). With respect to (a), the query language expresses queries for searching a particular object, and it is equipped with forward or backward recursion to retrieve links from an object (for instance, it allows to retrieve the nodes of a way). With respect to

[24] http://wiki.openstreetmap.org/wiki/Osmosis

[25] http://www.exist-db.org/

[26] http://basex.org/

[27] http://www.marklogic.com/

[28] http://expath.org/spec/geo

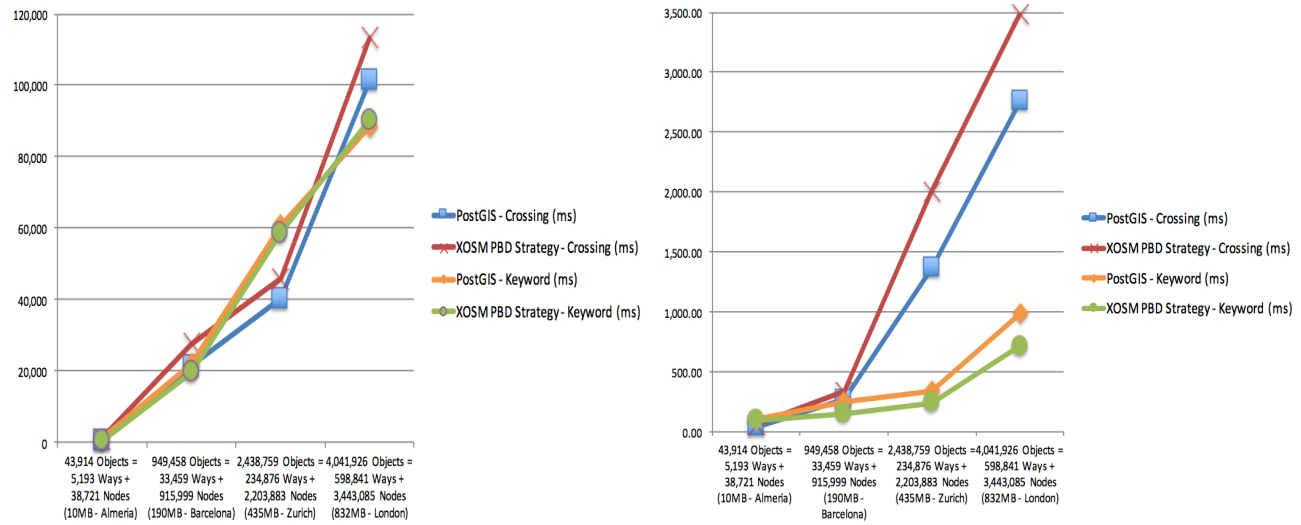

FIGURE 16. XOSM versus PostGIS; Spatial query crossing and keyword query hotel with at least two stars for non-cached data; Spatial query crossing and keyword query hotel with at least two stars for cached data.

Tool	*Description*	*Url*
OSM APIs		
JOSM	OSM Viewer	https://josm.openstreetmap.de/
Xapiviewer	OSM Viewer	http://osm.dumoulin63.net/xapiviewer/
XAPI	OSM API	http://wiki.openstreetmap.org/wiki/Xapi
Overpass API	OSM API	http://overpass-api.de/
OSM Query Languages		
GeoSPARQL	SPARQL extension	http://geosparql.org/
stSPARQL	SPARQL extension	http://www.strabon.di.uoa.gr/stSPARQL
Parliament	GeoSPARQL impl.	http://parliament.semwebcentral.org/
uSeekM	GeoSPARQL impl.	https://dev.opensahara.com/projects/useekm/
Virtuoso	RDF engine	http://virtuoso.openlinksw.com/
OWLIM	RDF engine	http://ontotext.com/products/ontotext-graphdb-owlim/
Allegrograph	RDF engine	http://franz.com/agraph/allegrograph/
NoSQL and Hadoop		
GeoCouch	NoSQL	https://github.com/couchbase/geocouch/
MongoBD	NoSQL	https://docs.mongodb.org/manual/applications/geospatial-indexes/
Neo4j	NoSQL	https://github.com/neo4j-contrib/spatial
Spatial Hadoop	MapReduce	http://spatialhadoop.cs.umn.edu/
Hadoop-GIS	MapReduce	https://web.cci.emory.edu/confluence/display/hadoopgis
TAREEG	Hadoop	http://siwa-umh.cs.umn.edu/app/webroot/osme/

TABLE 4. Related Work

(b), the query language expresses queries using several searching criteria. Among others, it can express: to find all elements in a bounding box, to find all elements near something else, to find all elements by keyword (exact value, non-exact value and regular expressions), negation, union, difference, intersection, and filtering, with a rich set of selectors, and by polygon, by area pivot, and so on. However, *Overpass API* facilities (i.e., query composition and filtering) cannot be combined with Clementini's spatial operators. In *Overpass API*, only intersection is considered (proximity equal to 0 by using the across selector). For instance, the query *"Retrieve the streets crossing Calzada de Castro street and touching Nuestra Señora de Montserrat street"* is not allowed in *Overpass API*, but allowed in our approach. On the other hand, *Overpass API* has a rich query language for keyword searching. Aggregation operators are not covered by Overpass API. The XOSM API is considerably more expressive than previous approaches, allowing to execute XQuery queries involving LGOD resources.

6.3. OSM Query Languages

In the context of RDF and SPARQL, there are several proposals of languages and tools for working with spatial data. GeoSPARQL [14] (standard of the Open Geospatial Consortium) and stSPARQL [15] are the most relevant contributions to this area. Both are very similar. Omitting aggregate functions and updates from stSPARQL, stSPARQL is a subset of GeoSPARQL. GeoSPARQL provides a vocabulary to express spatial data in RDF, and defines an extension of SPARQL for querying. stSPARQL uses SELECT, FILTER and HAVING clauses of SPARQL in combination with spatial predicates to query RDF spatial data. FILTER can be combined with them to define spatial selections and joins. These can be also used in the SELECT and HAVING clauses. Aggregation is present in stSPARQL in the form of union, intersection and extent (i.e., minimum bounding box of a set of geometries). stSPARQL uses a B-tree to index non spatial data, while R-Tree-over-GiST is used is used for spatial indexing (provided by PostGIS). StSPARQL is mapped to SQL queries executed under PostGIS. Parliament [16] is an implementation of the GeoSPARQL using Jena as RDF SPARQL engine. uSeekM uses the RDF engine Sesame as well as PostGIS, and implements GeoSPARQL features. stSPARQL is supported by Strabon [17], which extends the RDF engine Sesame with spatial data stored in PostGIS. On the other hand, Virtuoso, OWLIM and AllegroGraph are RDF-based engines supporting geometries of points.

While the technologies are different (RDF/SPARQL versus XML/XQuery) we found analogies with the existing approaches. With regard to spatial operators, our spatial library is built on top of EXPath/JTS and thus, providing similar expressivity. Indexing is based on PostGIS in RDF/SPARQL approaches, and the same happens in our approach with the PBD strategy. Additionally, R*-tree indexing is adopted for live data in our RLD strategy. With regard to expressivity of the query language, RDF/SPARQL based languages can be considered similar to our proposal, except for aggregation operators. The only case of language equipped with aggregation is stSPARQL, including union, intersection and extent. Thus our XOSM library provides a richer set of aggregation operators. We are also equipped with higher order functions, enabling a more concise specification of queries. Unfortunately, SPARQL and its spatial dialects are not equipped with higher order (although there exists some proposal [31] for SPARQL).

6.4. NoSQL and Hadoop approaches

NoSQL databases [32] have been used to import and query OSM data. This is the case of *GeoCouch* enabling bounding box queries and filtering by keyword and/or geometry, *MongoDB* expressing queries about the inclusion/intersection in a polygon and nearest points of a point, and *Neo4j* performing spatial operations like searching for elements within specified regions or within a specified distance of a point of interest. Recently, *MapReduce* based systems [33, 34] have risen as a scalable and cost effective solution for massively parallel data processing. Hadoop, the open source implementation of MapReduce, has been applied to support big data analytics, in particular in *Hadoop-GIS* [35], to support declarative spatial queries, and in *Spatial Hadoop* [36] enabling spatial operators, including range query, kNN, and spatial join. In [37] Hadoop has been used to extract information from OSM. The developed system, called TAREEG, is an easy and efficient online Web service to extract spatial data (road networks, lakes, buildings, rivers, and parks) from OSM in CSV, KML, Esri shapefiles and WKT format, and also to visualize them. R-tree indexing is achieved over multiple machines as a means for achieving scalability. TAREEG shares with our proposal to handle OSM data as well as to provide a Web service to extract OSM data. However, the query language (a range query with an area and a predicate filter for an spatial feature) is more limited than our proposal. In any case, we believe that the use of Hadoop in our approach, similarly to TAREEG, can be considered as future work.

6.5. Aggregation of Spatial Data

Finally, several proposals of aggregation operators for spatial data have been proposed in the literature [24, 38, 39, 40, 41, 42]. They have been studied in the context of *Spatial Data Warehouses* and *OLAP*. The term *SOLAP* was coined in this framework, and extensions to the well-known OLAP have been proposed. Additionally, R-tree based structures have been proposed to deal with spatial indexing and aggregation [43, 44, 45]. Our proposal, even though cannot be properly considered a SOLAP approach, is inspired in this framework, providing a library to express queries involving aggregation operators.

7. CONCLUSIONS AND FUTURE WORK

In this paper a framework called XOSM for integrating and querying OSM and LGOD resources is presented. The framework is equipped with a Web tool and a rich XQuery-based library, enabling the definition of queries combining OSM layers and layers created from Linked Geo Open Data resources (KML, GeoJSON, CSV, RDF and also XML). The framework also provides an API to execute XQuery queries using the library. We have shown a batch of examples of queries and benchmarks for several datasets. The complete code of XOSM can be found at https://github.com/ualabecerra/XOSM-Tool.

One of the contributions of the work is the definition of an OSM spatial library for XQuery. An XQuery programmer can find very useful to express queries

on OSM data using an XQuery library. While XQuery is already equipped with a spatial library, called EXPath Geo Module, and usually integrated with XQuery implementations, the library is limited to spatial operations on GML geometries: Points, LineStrings, Polygons, etc. For handling OSM data, an OSM to GML geometry transformation is required, and OSM textual data have to be handled. Thus, a library for OSM facilitates the definition of queries, hiding to the programmer the details of the representation in XML and conversion tasks of OSM.

While a specific OSM library for XQuery could be considered too restrictive, we have shown that the transformation library opens the application domain of the library. Not only OSM data can be handled, but also most commonly used geo-spatial data formats: KML, GeoJSON, CSV, RDF and XML. This is the key of success of our proposal, hiding all the implementation details to the programmer.

Anyone can argue that the expressivity power of the spatial library is similar to existing spatial query languages, and it is right. Our main aim was to provide an unified framework in which querying and integration is possible without using external languages, and in which on-the-fly integration is possible.

With regard to indexing, we are only concerned to reach at least the same performance of existing systems. For this reason, we have compared with PostGIS for backup data, which is, the facto, the commonly used spatial datastore and data management for existing systems (among them, geo-spatial SPARQL extensions). Additionally, our live data indexing makes possible to work with updated data.

As future work, we would like to extend our work as follows.

1. *Relations on OSM*: Our query language will be extended to work with relations. The set of OSM operators will be enriched with operators for working with relations (bus routes, groups of buildings, postal addresses, house numbers, etc).
2. *Other Web resources*: Some other Web APIs can be considered in order to combine their datasets with OSM data. This is the case, for instance, of Twitter which provides geo-located tweets [46, 47].
3. *OSM validation*: The quality of OSM maps has been recently studied for several authors [3, 48, 49, 50, 51, 52, 53, 54]. We have found in many cases that OSM maps are of poor quality and thus queries cannot find the expected results. The validation of properties on OSM maps will be subject of study.
4. *Spatial Hadoop*. We study how to build XOSM on top of Spatial Hadoop using the XQuery-based proposals of [55, 56].

ACKNOWLEDGEMENTS

This work was supported by the EU (FEDER) and the Spanish MINECO Ministry (*Ministerio de Economía y Competitividad*) under grant CAVI-TEXTUAL TIN2013-44742-C4-4-R.

REFERENCES

[1] Goodchild, M. F. (2007) Citizens as Sensors: the World of Volunteered Geography. *GeoJournal*, **69**, 211–221.

[2] Goodchild, M. F. and Li, L. (2012) Assuring the Quality of Volunteered Geographic information. *Spatial statistics*, **1**, 110–120.

[3] Ballatore, A. and Mooney, P. (2015) Conceptualising the Geographic World: the Dimensions of Negotiation in Crowdsourced Cartography. *International Journal of Geographical Information Science*, **29**, 2310–2327.

[4] Lin, W. (2016) Openstreetmap in giscience: experiences, research and applications. *International Journal of Geographical Information Science*, **30**, 823–824.

[5] Bennett, J. (2010) *OpenStreetMap: Be Your Own Cartographer* Community Experience Distilled. Packt Publishing, Limited, Birmingham, UK.

[6] Ramm, F., Topf, J., and Chilton, S. (2011) *OpenStreetMap: Using and Enhancing the Free Map of the World*. UIT Cambridge, Cambridge, UK.

[7] Coast, S. (2011) How OpenStreetMap Is Changing the World. In Tanaka, K., Fröhlich, P., and Kim, K.-S. (eds.), *Web and Wireless Geographical Information Systems: 10th International Symposium, W2GIS 2011, Kyoto, Japan, March 3-4, 2011. Proceedings*. Springer LNCS 6574, Berlin, Heidelberg.

[8] Bizer, C., Heath, T., and Berners-Lee, T. (2009) Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pp. 205–227. IGI-global, Hershey, Pennsylvania (USA).

[9] Yu, L. (2011) Linked Open Data. *A Developer's Guide to the Semantic Web*. Springer, Berlin, Heidelberg.

[10] Auer, S., Lehmann, J., and Hellmann, S. (2009) Linkedgeodata: Adding a spatial dimension to the web of data. *Proceedings of the 8th International Semantic Web Conference, ISWC '09*, Chantilly, VA, USA, October, pp. 731–746. Springer, LNCS 5823.

[11] Robie, J., Chamberlin, D., Dyck, M., and Snelson, J. (2014) XQuery 3.0: An XML Query Language. Technical report. World Wide Web Consortium (W3C), https://www.w3.org/TR/xquery-30/.

[12] Bamford, R., Borkar, V., Brantner, M., Fischer, P. M., Florescu, D., Graf, D., Kossmann, D., Kraska, T., Muresan, D., Nasoi, S., et al. (2009) XQuery Reloaded. *Proceedings of the VLDB Endowment*, **2**, 1342–1353.

[13] Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J., and Siméon, J. (2010) XML Path Language (XPath) 2.0. Technical report. World Wide Web Consortium (W3C), http://www.w3.org/TR/xpath20/.

[14] Battle, R. and Kolas, D. (2011) GeoSPARQL: Enabling a Geospatial Semantic Web. *Semantic Web Journal*, **3**, 355–370.

[15] Koubarakis, M. and Kyzirakos, K. (2010) Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL. *The semantic web: research and applications*, pp. 425–439. Springer, Berlin, Heidelberg.

[16] Battle, R. and Kolas, D. (2012) Enabling the Geospatial Semantic Web with Parliament and GeoSPARQL. *Semantic Web*, **3**, 355–370.

[17] Kyzirakos, K., Karpathiotakis, M., and Koubarakis, M. (2012) Strabon: a Semantic Geospatial DBMS. *11th International Semantic Web Conference, ISWC 2012*, pp. 295–311. Springer, LNCS 7649, Boston, MA, USA.

[18] Kaminski, M., Kostylev, E. V., and Cuenca Grau, B. (2016) Semantics and expressive power of subqueries and aggregates in sparql 1.1. *Proceedings of the 25th International Conference on World Wide Web, WWW '16*, Republic and Canton of Geneva, Switzerland, April, pp. 227–238. International World Wide Web Conferences Steering Committee.

[19] Atzori, M. (2014) Computing recursive sparql queries. *2014 IEEE International Conference on Semantic Computing*, Newport Beach, CA, USA, June, pp. 258–259. IEEE.

[20] Almendros-Jiménez, J. M. and Becerra-Terón, A. (2016) Geographical Information Systems Theory, Applications and Management: First International Conference, GISTAM 2015, Barcelona, Spain, April 28-30, 2015, Revised Selected Papers. Springer, Berlin, Heidelberg.

[21] Almendros-Jiménez, J. M. and Becerra-Terón, A. (2015) Querying Open Street Map with XQuery. *GISTAM 2015 - Proceedings of the 1st International Conference on Geographical Information Systems Theory, Applications and Management*, Barcelona, Spain, April, pp. 61–71. SciTePress.

[22] Almendros-Jiménez, J. M., Becerra-Terón, A., and Torres, M. (2015) Aggregation Operators in Geospatial Queries for Open Street Map. *OTM 2015 Conferences, ODBASE 2015, Proceedings*, Rhodes, Greece, October, pp. 501–518. Springer LNCS 9415.

[23] Almendros-Jiménez, J. M. and Becerra-Terón, A. (2015) Distance Based Queries in Open Street Map. *Twenty-Sixth International Workshop on Database and Expert System Applications, Proceedings*, Valencia, Spain, September, pp. 235–239. IEEE.

[24] Ruiz, C. V. and Times, V. C. (2009) A Taxonomy of SOLAP Operators. *XXIV Simpósio Brasileiro de Banco de Dados*, Fortaleza, Brasil, October, pp. 151–165. Biblioteca Digital Brasileira de Computacao.

[25] Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., and Pirahesh, H. (1997) Data Cube: A Relational Aggregation Operator Generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, **1**, 29–53.

[26] Boucelma, O. and Colonna, F. (2004) GQuery: a Query Language for GML. *Proc. of the 24th Urban Data Management Symposium*, Chioggia, Italy, October, pp. 27–29. Citeseer.

[27] Huang, C.-H., Chuang, T.-R., Deng, D.-P., and Lee, H.-M. (2009) Building GML-native web-based geographic information systems. *Computers & Geosciences*, **35**, 1802–1816.

[28] Kay, M. (2008) Ten Reasons Why Saxon XQuery is Fast. *IEEE Data Eng. Bull.*, **31**, 65–74.

[29] Li, Y., Li, J., and Zhou, S. (2004) GML Storage: A Spatial Database Approach. In et al. [57], pp. 55–66.

[30] Olbricht, R. M. (2015) Data Retrieval for Small Spatial Regions in OpenStreetMap. *OpenStreetMap in GIScience*, pp. 101–122. Springer, Berlin, Heidelberg.

[31] Atzori, M. (2014) Toward the Web of Functions: Interoperable Higher-Order Functions in SPARQL. *13th International Semantic Web Conference, ISWC 2014*, pp. 406–421. Springer LNCS 8797, Riva del Garda, Italy.

[32] de Souza Baptista, C., Pires, C. E. S., Leite, D. F. B., and de Oliveiraa, M. G. (2014) NoSQL geographic databases: an overview. *Geographical Information Systems: Trends and Technologies*, **73**.

[33] Dean, J. and Ghemawat, S. (2010) MapReduce: a flexible data processing tool. *Communications of the ACM*, **53**, 72–77.

[34] Lin, J.-C., Leu, F.-Y., and Chen, Y.-p. (2016) Impacts of Task Re-Execution Policy on MapReduce Jobs. *The Computer Journal*, **59**, 701.

[35] Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. (2013) Hadoop GIS: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, **6**, 1009–1020.

[36] Eldawy, A. and Mokbel, M. F. (2015) SpatialHadoop: A MapReduce Framework for Spatial Data. *2015 IEEE 31st International Conference on Data Engineering*, Seoul, South Korea, April, pp. 1352–1363. IEEE.

[37] Alarabi, L., Eldawy, A., Alghamdi, R., and Mokbel, M. F. (2014) TAREEG: A MapReduce-based System for Extracting Spatial Data from OpenStreetMap. *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, New York, NY, USA, November, pp. 83–92. ACM.

[38] da Silva, J., de Oliveira, A. G., Fidalgo, R. N., Salgado, A. C., and Times, V. C. (2010) Modelling and Querying Geographical Data Warehouses. *Information Systems*, **35**, 592–614.

[39] Gómez, L., Haesevoets, S., Kuijpers, B., and Vaisman, A. A. (2009) Spatial Aggregation: Data Model and Implementation. *Information Systems*, **34**, 551–576.

[40] Bédard, Y., Rivest, S., and Proulx, M.-J. (2007) Spatial. Online Analytical. Processing (SOLAP): Concepts, Architectures, and Solutions. *Data Warehouses and OLAP: Concepts, Architectures, and Solutions, Idea Group Inc*, pp. 298–319. IGI global, Hershey, Pennsylvania (USA).

[41] Bimonte, S., Bertolotto, M., Gensel, J., and Boussaid, O. (2012) Spatial OLAP and Map Generalization: Model and Algebra. *International Journal of Data Warehousing and Mining (IJDWM)*, **8**, 24–51.

[42] Baltzer, O., Rau-Chaplin, A., and Zeh, N. (2013) Building a Scalable Spatial OLAP System. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, Coimbra, Portugal, March, pp. 13–15. ACM.

[43] Jurgens, M. and Lenz, H.-J. (1998) The R/sub a/*-tree: an Improved R*-tree with Materialized Data for Supporting Range Queries on OLAP-data. *Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on*, Vienna, Austria, August, pp. 186–191. IEEE.

[44] Papadias, D., Kalnis, P., Zhang, J., and Tao, Y. (2001) Efficient OLAP Operations in Spatial Data Warehouses. *Advances in spatial and temporal databases*, pp. 443–459. Springer, Berlin, Heidelberg.

[45] Rao, F., Zhang, L., Yu, X. L., Li, Y., and Chen, Y. (2003) Spatial Hierarchy and OLAP-favored Search in Spatial Data Warehouse. *Proceedings of the 6th ACM international workshop on Data warehousing and OLAP*, New Orleans, Louisiana, USA, November, pp. 48–55. ACM.

[46] Weiler, A., Grossniklaus, M., and Scholl, M. H. (2017) Survey and Experimental Analysis of Event Detection Techniques for Twitter. *The Computer Journal*, **60**, 329.

[47] Ghahremanlou, L., Sherchan, W., and Thom, J. A. (2015) Geotagging Twitter Messages in Crisis Management. *The Computer Journal*, **58**, 1937.

[48] Girres, J.-F. and Touya, G. (2010) Quality Assessment of the French OpenStreetMap Dataset. *Transactions in GIS*, **14**, 435–459.

[49] Mooney, P., Corcoran, P., and Winstanley, A. C. (2010) Towards Quality Metrics for OpenStreetMap. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, San José, California, November, pp. 514–517. ACM.

[50] Jilani, M., Corcoran, P., and Bertolotto, M. (2014) Automated Highway Tag Assessment of OpenStreetMap Road Networks. *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Dallas, Texas, November, pp. 449–452. ACM.

[51] Mondzech, J. and Sester, M. (2011) Quality Analysis of OpenStreetMap Data based on Application Needs. *Cartographica: The International Journal for Geographic Information and Geovisualization*, **46**, 115–125.

[52] Mooney, P. and Corcoran, P. (2012) The Annotation Process in OpenStreetMap. *Transactions in GIS*, **16**, 561–579.

[53] Koukoletsos, T., Haklay, M., and Ellul, C. (2012) Assessing Data Completeness of VGI through an Automated Matching Procedure for Linear Data. *Transactions in GIS*, **16**, 477–498.

[54] Neis, P., Goetz, M., and Zipf, A. (2012) Towards Automatic Vandalism Detection in OpenStreetMap. *ISPRS International Journal of Geo-Information*, **1**, 315–332.

[55] Carman, E. P., Westmann, T., Borkar, V. R., Carey, M. J., and Tsotras, V. J. (2015) A scalable parallel XQuery processor. *Big Data, 2015 IEEE International Conference on*, Santa Clara, CA, USA, October-November, pp. 164–173. IEEE.

[56] Khatchadourian, S., Consens, M., and Siméon, J. (2011) Chuql: processing xml with XQuery using Hadoop. *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, Toronto, Ontario, Canada, November, pp. 74–83. IBM Corp.

[57] et al., S. W. (ed.) (2004) *Conceptual Modeling for Advanced Application Domains, ER 2004 Workshops CoMoGIS, COMWIM, ECDM, CoMoA, DGOV, and ECOMO, Shanghai, China, November 8-12, 2004, Proceedings*, Berlin, Heidelberg, November. Springer LNCS 3289.

Function	*Informal Definition*
getElementByName(m,n)	OSM representation of a given OSM element n in OSM map m
getLayerByName(m,n,d)	OSM elements of the OSM map m at distance d of a given OSM element of name n
getLayerByElement(m,s,d)	OSM elements of the OSM map m at distance d of a given OSM element s
getElementsByKeyword(m,k)	OSM elements annotated with a keyword k in OSM map m
getLayerByBB(m,mlat,Mlat,mlon,Mlon)	OSM elements of a Bounding Box

TABLE 5. Index-based Functions

Function	*Formal Definition*
getLayerByName(m,n,d)	$RLD : \{s \mid s \in \mathcal{S} \wedge getElementByName(m,n) = l \wedge mindist(mbr(s), mbr(l)) \leq d\}$ $PBD : \{s \mid s \in \mathcal{S} \wedge getElementByName(m,n) = l \wedge d(s,l) \leq d\}$
$getElementsByKeyword(m,k)$	$\{s \mid s \in \mathcal{S} \wedge ((k \in \mathcal{K} \wedge \exists v.k(s) = v) \vee (\exists f \in \mathcal{K}.f(s) = k))\}$
getLayerByBB(m,mlat,Mlat,mlon,Mlon)	$\{s \mid s \in \mathcal{S} \wedge \forall n \in s.mlat \leq lat(n) \leq Mlat \wedge mlon \leq lon(n) \leq Mlon\}$

TABLE 6. Index-based Functions

Operator	*Informal Definition*	*Spatial Operation*
DWithIn(s_1,s_2,d)	**true** whenever the shortest distance between the elements s_1 and s_2 is shorter than d	Distance
furtherNorthNodes(n_1,n_2)	**true** whenever the node n_1 is further north than the node n_2.	Using latitudes and distinguishing cases in each hemisphere.
furtherNorthWays(w_1,w_2)	**true** whenever the way w_1 is further north than the way w_2.	Using *furtherNorthNodes*.

TABLE 7. Coordinate and Distance Based OSM Operators

Operator	*Formal Definition*
$DWithin(s_1, s_2, d)$	**true** iff $dist(s_1, s_2) \leq d$
$furtherNorthNodes(n_1, n_2)$	**true** *if* $lat(n_2) - lat(n_1) > 0$ $lat(n_1) \geq 0, lat(n_2) \geq 0$ **true** *if* $(-lat(n_2)) - (-lat(n_1)) < 0$ $lat(n_1) \leq 0, lat(n_2) \leq 0$ **false** *if* $lat(n_1) \geq 0, lat(n_2) < 0$ **true** *if* $lat(n_1) < 0, lat(n_2) \geq 0$
$furtherNorthWays(w_1, w_2)$	**true** iff $\forall n_1 \in w_1.\forall n_2 \in w_2.furtherNorthNodes(n_1, n_2)$

TABLE 8. Coordinate and Distance Based OSM Operators

Operator	*Informal Definition*	*Clementini's Operator*
wayIn(n,w)	**true** whenever n (node) belongs to w (way)	Distance
waySame(n_1,n_2,m)	**true** whenever n_1 (node) and n_2 (node) belongs to the same way of the OSM map m	wayIn and Equals
intersectionPoints(w_1,w_2)	the intersection points of w_1 (way) and w_2 (way)	
crossing(w_1,w_2)	**true** whenever w_1 (way) crosses w_2 (way).	Crosses
nonCrossing(w_1,w_2)	**true** whenever w_1 does not cross w_2.	crossing Negation

TABLE 9. Clementini Based OSM Operators

Operator	*Formal Definition*
$wayIn(n, w)$	**true** iff distance(n,w)=0
$waySame(n_1, n_2, m)$	**true** iff $\exists$ $ways$ $w_1, w_2 \in \mathcal{S}.wayIn(n_1, w_1) \wedge wayIn(n_2, w_2) \wedge equals(w_1, w_2)$
$intersectionPoints(w_1, w_2)$	$\{n \mid n \in w_1 \wedge n \in w_2\}$

TABLE 10. Clementini Based OSM Operators

Operator	*Informal Definition*
searchKeyword(s,kv)	**true** whenever the OSM element s has some key function k or value v equal to kv
searchKeywordSet(s,($kv_1, \ldots, kv_n$))	**true** whenever the OSM element s has some key function k or value v equal to some kv_i
searchTag(s,k_0,v_0)	**true** whenever the OSM element s has some key function k and value v equal to k_0 and v_0, respectively
getTagValue(s,k_0)	the value v in the OSM element s associated to the key function k_0

TABLE 11. Keyword Based OSM Operators

Operator	*Formal Definition*
$searchKeyword(s, kv)$	**true** iff $(kv \in \mathcal{K} \wedge \exists v.kv(s) = v) \vee (\exists f \in \mathcal{K}.f(s) = kv)$
$searchKeywordSet(s, (kv_1, \ldots, kv_n))$	**true** iff $\exists kv_i.searchKeyword(s, kv_i)$
$searchTag(s, (k, v))$	**true** iff $k \in \mathcal{K} \wedge k(s) = v$
$getTagValue(s, k)$	v iff $k \in \mathcal{K} \wedge k(s) = v$

TABLE 12. Keyword Based OSM Operators

Type	*Operators*
Distributive	*topologicalCount(sq,e,b)*, *metricMin(sq,m)*, *metricMax(sq,m)*, *metricSum(sq,m)*, *minDistance(sq,e)* and *maxDistance(sq,e)*
Algebraic	*metricAvg(sq,m)*, *metricStdev(sq,m)*, *avgDistance(sq,e)*, *metricTopCount(sq,k,m)*, *metricBottomCount(sq,k,m)*, *topCountDistance(sq,k,e)* and *bottomCountDistance(sq,k,e)*
Holistic	*metricMedian(sq,m)*, *metricMode(sq,m)* and *metricRank(sq,m,k)*

TABLE 13. Aggreggation Based OSM Operators

Type	*Formal Definition*
$topologicalCount(sq, e, b)$	$\#\{s \mid s \in sq, b(e, s)\}$
$metricMin(sq, m)$	$\{s_1, \ldots, s_n\}$ if there exist $s_1, \ldots, s_n \in sq$ such that $\forall s' \in sq, s' \neq s_i, m(s_i) < m(s')$
$metricSum(sq, m)$	$\sum_{s \in sq} m(s)$
$minDistance(sq, e)$	s if $s \in sq, dist(s, e) < dist(s', e)\ \forall s' \in sq.s' \neq s$
$metricAvg(sq, m)$	$\frac{\sum_{s \in sq} m(s)}{\#sq}$
$avgDistance(sq, e)$	$\frac{\sum_{s \in sq} dist(s,e)}{\#sq}$
$metricStdev(sq, m)$	$\sqrt{\frac{\sum_{s \in sq}(m(s) - metricAvg(sq,m))^2}{\#sq}}$
$metricTopCount(sq, k, m)$	$\{s_1, \ldots, s_k\}$ if there exist distinct $s_1, \ldots, s_k \in sq$ such that $\forall s' \in sq, s' \neq s_i, m(s_i) > m(s')$
$metricMediam(sq, m)$	k whenever $\#sq = n$ and, if $n\ mod\ 2 = 1$ then $k = m_{n\ div\ 2}$ and if $n\ mod\ 2 = 0$ then $k = \frac{m_{n\ div\ 2} + m_{n\ div\ 2+1}}{2}$ where $\{m_1, \ldots, m_n\}$ is an ordered permutation of $\{m(s) \mid s \in sq\}$
$metricMode(sq, m)$	k if there exists $S \subseteq sq$ such that $m(s) = k\ \forall s \in S$, and there not exists S' and k' such that $k' \neq k$, $\#S' \geq \#S$ and $m(s') = k'\ \forall s' \in S'$
$metricRank(sq, m, e)$	k if there exist S, S' such that $m(e) < m(s)\ \forall s \in S$ and $m(e) > m(s')\ \forall s' \in S'$, $\#S = k - 1$ and $sq = S \cup \{e\} \cup S'$

TABLE 14. Aggreggation Based OSM Operators