PlanetData

Network of Excellence

FP7 – 257641

D17.2 Call 2: Linked Map read-write Linked Data enabled OGC Web map client

Coordinator: Jesús Barrera (GEOSLAB)
With contributions from: Francisco J Lopez-Pellicer (UNIZAR)

1st Quality reviewer: Adrian Brasoveanu (MODUL)
2nd Quality reviewer: Loris Bozzato (FBK)

Deliverable nature:	Prototype (P)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M42
Actual delivery date:	M44
Version:	1.0
Total number of pages:	18
Keywords:	Linked data, geographic information, wms, map viewer

Abstract

This deliverable describes the development of a JavaScript library that deployed in a web client enables it to interoperate with a Linked Map Service. This JavaScript library integrates visual widgets for rendering data and editor widgets for adding or curating data. In order to ease the diffusion of this technology, the implementation is based on an existing open source web map client library named OpenLayers.

Executive summary

The Work Package 17 of the Linked Map subproject of the PlanetData project comprises two tasks:

- The development of an extension to the OGC Web Map Server (WMS) specification compatible with read-write Linked Data.
- The development of a web map client able to use that extension in a web browser.

The first task was fully described in the deliverable D17.1 "*Call 2: Linked Map Read-write Linked Data enabled OGC Web map server*". The developed service was called Linked Map Service (LMS). It provides a semantic web interface and, at the same time, an interface compliant to the OGC WMS standard. This way, the service acts as a reverse proxy for WMS servers and it also behaves as a Linked Data front end.

The current deliverable describes the second task, the development of a LMS client implemented as a JavaScript library. This library deployed in a web application enables it to interoperate with a LMS server. That is, this library is able to manage interfaces offered by the service: the OGC WMS interface and the semantic web interface. The library has been developed by extending the open source software OpenLayers in order to support the semantic web interface of LMS.

Document Information

IST Project Number	FP7 - 257641	Acronym	PlanetData
Full Title	PlanetData		
Project URL	http://www.planet-data.eu/		
Document URL	http://wiki.planet-data.eu/web/D17.2		
EU Project Officer	Leonhard Maqua		

Deliverable	Number	D17.2	Title	Call2: Linked Map read-write Linked Data enabled OGC Web map client
Work Package	Number	WP17	Title	Call2: Linked Map read-write Linked Data WMS framework

Date of Delivery	Contractual	M42	Actual	M44
Status	version 1.0		final ☒	
Nature	prototype □ report □ demonstrator □ other □			
Dissemination level	public ☒ restricted to group □ restricted to programme □ consortium □			

Authors (Partner)	Jesús Barrera (GEOSLAB), Francisco J Lopez-Pellicer (UNIZAR)			
Responsible Author	Name	Jesús Barrera	E-mail	jesusb@geoslab.com
	Partner	GEOSLAB	Phone	+34 976 065152

Abstract (for dissemination)	This deliverable describes the development of a JavaScript library that deployed in a web client enables it to interoperate with a Linked Map Service. This JavaScript library integrates visual widgets for rendering data and editor widgets for adding or curating data. In order to ease the diffusion of this technology, the implementation is based on an existing open source web map client library named OpenLayers. .
Keywords	Linked data, geographic information, wms, map viewer

Version Log			
Issue Date	Rev. No.	Author	Change
2014/04/01	0.1	Jesús Barrera Francés	Template instantiation
2014/04/15	0.2	Jesús Barrera Francés	First draft
2014/04/25	0.3	Francisco J Lopez-Pellicer	Revision and comments
2014/04/29	0.4	Jesús Barrera Francés	Adjustments
2014/04/30	0.5	Francisco J Lopez-Pellicer	First draft
2014/05/09	0.6	Francisco J Lopez-Pellicer	Draft ready for QA
2014/05/19	0.7	Jesús Barrera Francés	Adjustments after reviewers' comments
2014/05/26	1.0	Francisco J Lopez-Pellicer	Final release

Table of Contents

Abbreviations

CRS Coordinate Reference System

GIS Geographic Information System

KVP Keyword Value Pair

LMS Linked Map Service

OGC Open Geospatial Consortium

UML Unified Modeling Language

VGI Volunteer Geographic Information

WMS Web Map Service

List of figures

1 Introduction

The Linked Map project aims to demonstrate the enormous potential of linking together different sources of geographic information under the paradigm of Linked data. The project has carried out the conversion of a large geographic dataset produced by a public agency into RDF and the enrichment with RDF links to Volunteer Geographic Information (VGI) sources. The aim of this work is to demonstrate the feasibility of Linked Data as a useful tool to address the recording of the provenance and reliability of VGI data in order to develop a set of VGI data quality metrics. This task will be accomplished by using a web portal that will allow volunteers to review the reliability of VGI data by comparing it with official data. Users will make an assessment on RDF link quality and will detect data vandalism. User assessments will be later published as Linked Data. This will allow linking the final report on the quality of VGI data to its sources. The web portal will be an outcome of the next Work Package 18.

This deliverable describes the work made on designing and developing the Linked Map Service Client: a JavaScript library able to interoperate with the Linked Map Service (LMS), which was described in the deliverable D17.1 [1]. The goal of this JavaScript library is to draw interactive maps mixed with Linked Data and offer a way to the users for expressing their opinions about the quality of the RDF links between official data and VGI data managed by the project. This library will be deployed as a component of the client side of the aforementioned web portal.

The Linked Map Service is a service that addresses the interoperability between map services and Linked Data. This service acts as a reverse proxy for servers that implements the Open Geospatial Consortium Web Map Service (WMS) [2] specification (version 1.3.0) and, at the same time, it is able to dereference URIs using Linked Data best practices and techniques [3]. Figure 1 shows the interactions between web clients and the LMS server. WMS clients perform WMS requests to the LMS instance as the LMS instance is advertising himself as a traditional WMS server. WMS requests received from WMS clients are sent to remote WMS servers on behalf of them and server responses are sent back to the clients along with Web links, a kind of HTTP header, that points to alternative representations of the WMS response. Standard WMS clients will ignore these headers. However, a LMS client will be able to deal with the headers added by the LMS server to the response, dereferencing URIs that are either identifier URIs, document URIs or new WMS KVP requests. Then, they can interact again with the LMS instance to request more information about the data related to their initial request.

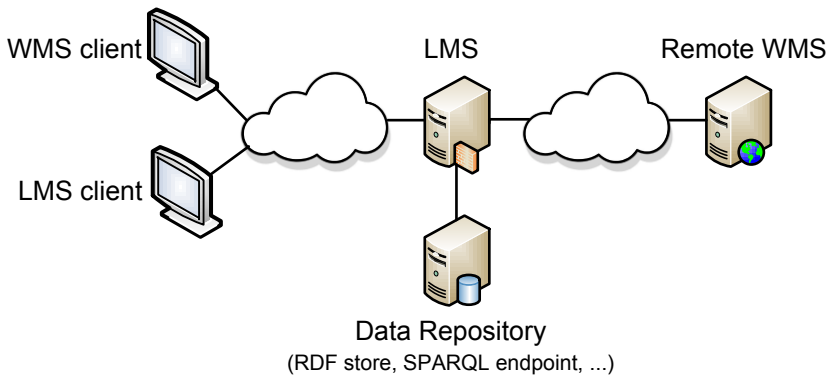

Figure 1 – LMS interactions

This document describes the development of the LMS JavaScript library. The final version of the LMS JavaScript library will be available as part of the source code of the Web portal that will be developed in Work Package 18 [4]. This Web portal will be part of the Linked Map web site[1]. This library is able to manage semantic information about the maps returned by the LMS server and offers the user a way to make content updates. Section 2 describes the main aspects of the architecture and the decisions taken in the design of the application, and Section 3 reports on the integration with the LMS server.

[1] http://linkedmap.unizar.es/

2 System architecture and design

This section describes the LMS JavaScript library functionality, the open source libraries used in the development, the main architecture and details about the implemented logic.

2.1 Main functionalities

The LMS client is a visualisation tool implemented as a JavaScript library. This library enables Web browsers to show RDF data processed in the project inside a map. This functionality gives users means to provide feedback about RDF data. For example, the LMS JavaScript library will be deployed as a component of the client side of the Web portal that will be developed in Work Package 18. The library will make possible to analyse RDF descriptions of geographical entities (also called geographical features) in context, that is, with the help of maps, and then make assessments on the quality of the data.

The library provides the following functionalities:

- Visual representation of geographical features belonging to a specific geographic area. This information will be displayed as icons on the map.
- Display information related to each feature when the user moves the mouse over these areas. The displayed information will be a map tip with the information to highlight.
- Access to detailed information about the feature when the user clicks with the mouse on one of them. The application will show a new floating window with all the detailed information about the selected feature.
- Editable forms to allow users to make contributions. In particular, users will be able to give their opinions about a link between two features from different sources or propose new links.
- Data are displayed in ETRS89 (EPSG:4258) Coordinate Reference System (CRS), the coordinate reference system adopted by the European Commission in Europe, and in common CRS such as the worldwide adopted WGS84 (EPSG:4326).
- Filter of features according to the source dataset they belong to.
- Typical GIS tools for map movements: zoom in, zoom out, panning, previous extent, next extent, back to the initial extent.
- Overview map: the user can interact with the general map by navigating on the overview map. The overview map changes depending on the scale displayed in the general map, because it does not make sense to show a global overview map if we are in a local area.
- Show legend of the layers displayed on the map.

2.2 Interactions with the LMS Server

As it was aforementioned, the LMS JavaScript library will be deployed as a component of the presentation layer of the portal that will serve as testbed for the technology and the place where the experiments to measure the quality of data will be performed. This portal should allow users to better understand the relationship between VGI data and authoritative data by means of maps (via the LMS server), and allow them to assess the quality of the automatic combination of both data sources. Therefore, the main two operations of the LMS client are drawing processed data on a map and allowing users to edit or fix the related data.

Figure 2 shows, using the conventional UML notation for sequence diagrams, the first of these scenarios which will be executed on the client side when a web browser loads a Open Layers map viewer that uses the LMS JavaScript library or when the user causes a zoom or pan event in the Open Layers map viewer, The LMS JavaScript library asks the LMS Server for a map. This is performed by sending to the server the bounding box of the map, that is, the geographical coordinates that define the corners of the rectangle that is shown on the screen. The LMS server process the request and sends a standard *GetMap* request to the external WMS in order to obtain an image (i.e. PNG, JPG…) that represents the requested map. Then, the LMS server overwrites the response response of the WMS by adding a header with URIs related to the

information contained in the map and returns the response to the requester, the LMS JavaScript library. Next, the LMS library draws the map with the support of specialized Open Layer code. Next, the LMS JavaScript library reads the headers added by the LMS server to the response in order to find the URI of a request to recover the list of features related to the image. Once the LMS JavaScript library has extracted this URI, it sends the request to the LMS server which in turn route it to a GeoSPARQL endpoint [5] in order to recover the list of geographic resources. The LMS server encodes the response in the format requested by the LMS JavaScript library (i.e. JSON) and returns it back to the client. At the end, the LMS JavaScript library draws an icon for each feature contained in the response.

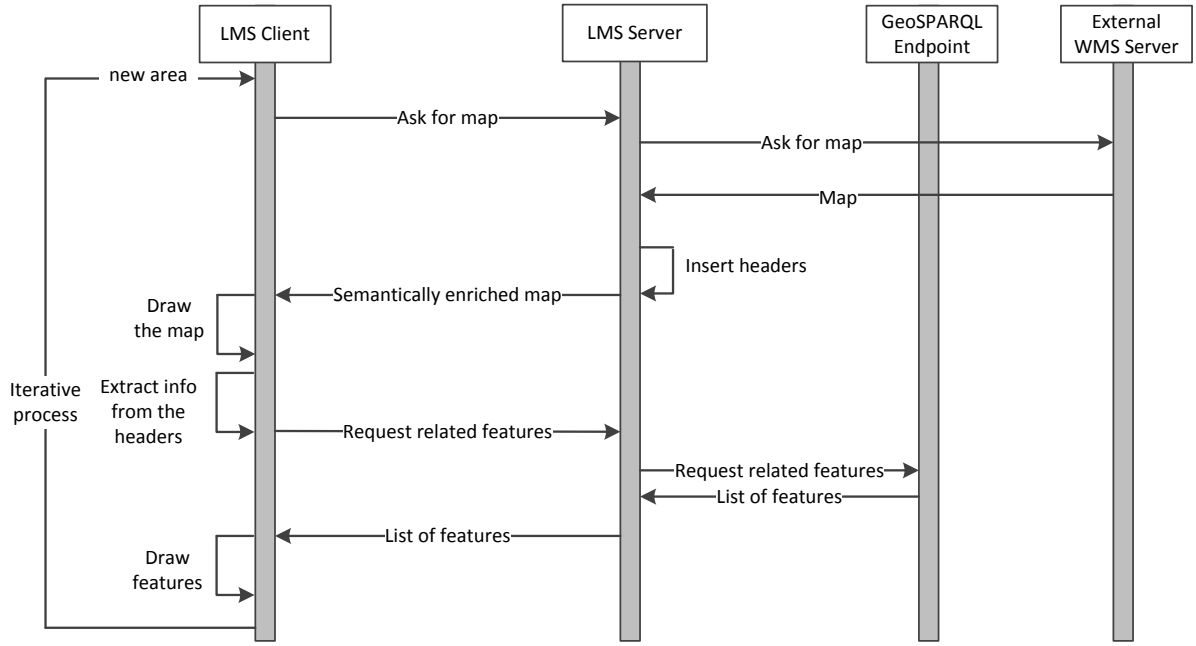

Figure 2 – Sequence diagram for drawing a map

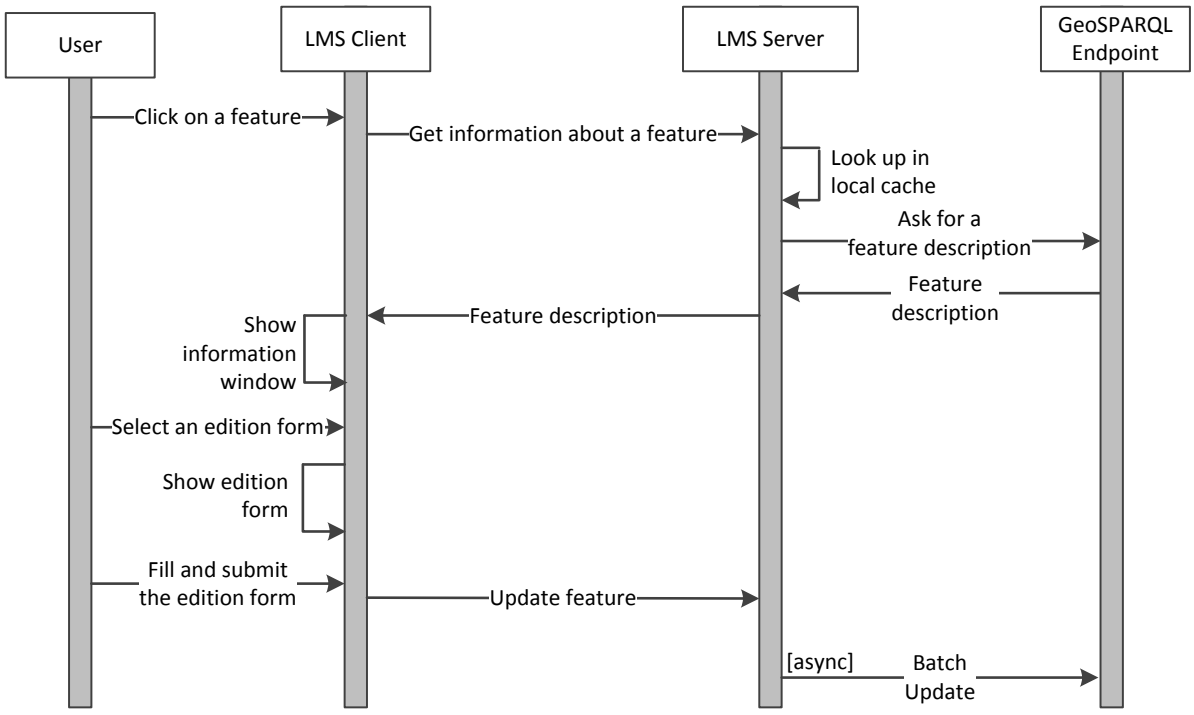

Figure 3 – Sequence diagram for editing a feature

Figure 3 shows the second scenario, which will be executed when a user wants to edit. The sequence of interactions is initiated on the user side when the user chooses a specific featured by clicking on it. The LMS JavaScript library receives the event and then it requests the complete information about this feature to a LMS Server. The LMS Server routes the query to a GeoSPARQL endpoint that contains the description of the resources. Note that GeoSPARQL responses can be cached, so, if the description has been previously cached, it is returned from the cache. Then, the LMS server encodes the response in the format requested by the LMS JavaScript library (i.e. JSON) and returns it back. At this point, the library may, for example, pop up a floating window with more detailed information about the selected feature and buttons associated to actions. If, for example, the user selects an edit option, the LMS JavaScript library may show a form that enables the user to edit the information. When the user accepts the modifications, the library may send the updates back to the server immediately or as part of a larger batch.

2.3 Architecture

The LMS JavaScript library is based on the OpenLayers[2] framework. OpenLayers is an open source JavaScript library developed by the Open Source Geospatial Foundation (OSGEO) that allows the visualisation of geospatial information on web environments. It provides an API for the development of Web applications similar to other commercial platforms as Google Maps or MSN Virtual Earth.

The choice of OpenLayers as a base for developing the LMS JavaScript library is based on the following criteria:

- It is a JavaScript library that provides the basic tools from which developers can deploy map viewer applications.
- It allows an easy configuration of the components that access to standard Web Map Services (WMS).
- It facilitates the display of vectorial data, and offers the possibility of displaying related data with map tips.
- It allows the integration of GoogleMaps.
- It is distributed under a BSD licence, which allows free distribution and modification of its code. Therefore, it is possible to alter and expand its code to improve and increase the functionality offered.
- It is in constant development, with numerous developers worldwide implementing new features, optimizing code and fixing old bugs. From time to time a new version is released including the latest changes.

One of the most powerful features of OpenLayers is its ability to integrate different types of layers. Figure 4 shows an UML diagram with the set of layers provided by OpenLayers (white and yellow) and the new ones added in the LMS client (orange) in order to implement the behavior described in section 2.2. The yellow ones are the most important classes regarding its functionality:

- `OpenLayers.Map`: This class is responsible for managing all components of the viewer. It has methods for adding layers, inserting controls, defining the level of zoom, etc.
- `OpenLayer.Layer`: This class is the parent of all layers. It implements the basic and common behavior of a layer.
- `OpenLayer.Marker`: This class manages an icon associated with a geographic location.

The classes developed in the context of the project have been colored in orange. According to the main goal of the project, the most important challenge is to make the web client able to integrate with LMS servers. Following the OpenLayers structure, the best approach is to create a new controller able to manage the new `LMS_WMS` Layer. The new layer is defined as an extension of the `WMS` Layer in order to take advantage of its functionality as long as it deals with the headers added by the LMS server to the response, dereferencing the URIs.

The new `LMS_WMS` Layer is responsible of the interaction with the LMS server including:

[2] http://openlayers.org

- Managing the request (similar to WMS request)
- Parsing the answer (including the new headers added by the LMS server)
- Dereferencing URIs and ordering the vectorial layer to refresh with the semantic information related to the visualized area.

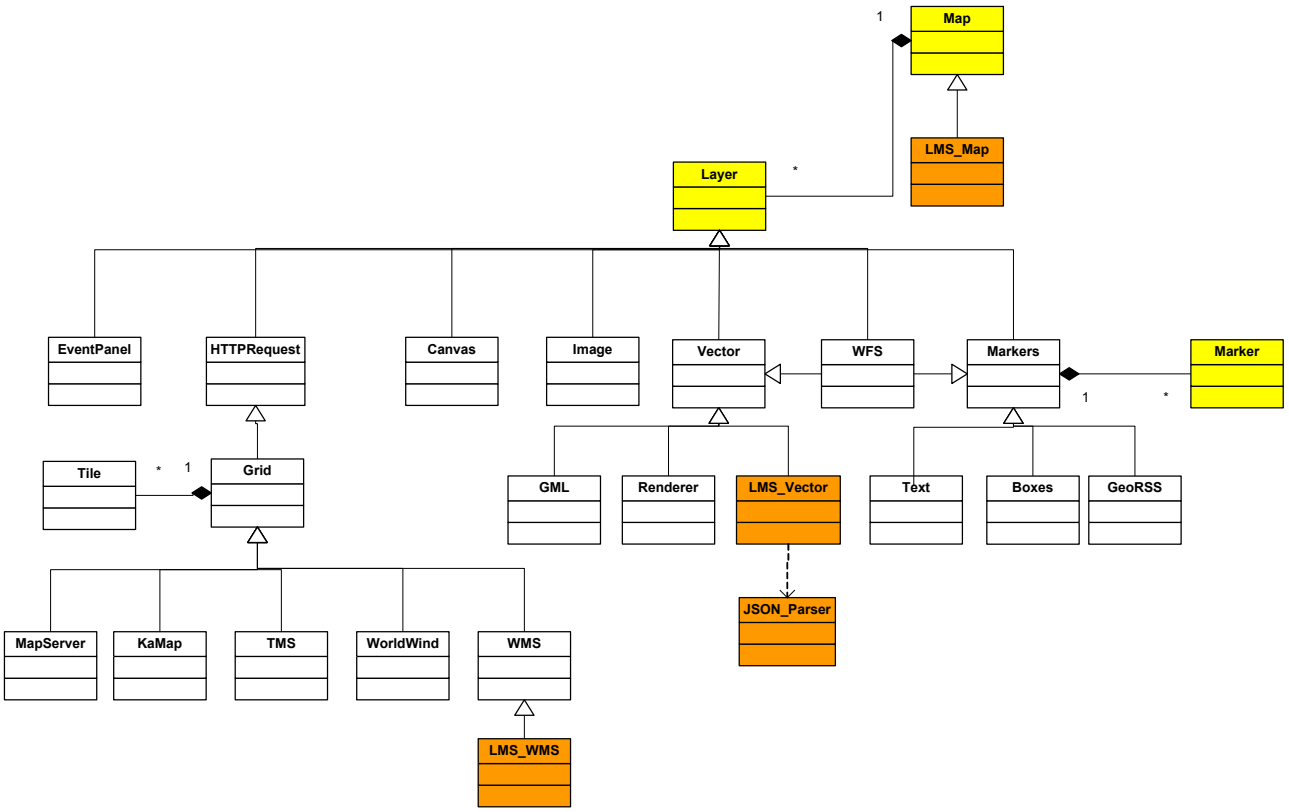

Figure 4 – Class diagram

Therefore, semantic vectorial data are printed in the map by a `LMS_Vector` class which is able to request according to the dereferencing URIs and also able to manage them following the defined behaviour. Vectorial data's behaviour is detailed below:

- A specific icon is drawn to represent each feature.
- A map tip is shown when the pointer moves over the icon.
- A floating window is shown when the pointer clicks in the icon.
- Vectorial data are refreshed when the extent is modified.
- Vectorial data answer is parsed by the `JSON_Parser` class.

Finally, we need to extend the main `Map` class with the `LMS_Map` class in order to define specific interactions with the viewer.

2.4 LMS Client Interface

One of the most important objectives defined in the design phase is to ensure the abstraction and reuse of the software component. The LMS client is based on a JavaScript Interface that provides a functionality façade. This interface contains a set of methods designed to provide LMS JavaScript library with the required functionality detailed in the sequence diagrams described in section 2.2.

The interface specification is detailed in Appendix I.

3 Integration with LMS

The LMS JavaScript library is being tested with the in development version 1.0 of the LMS server. This version implements the interaction described in previous sections. LMS 1.0 will support a read-write REST API interface for accessing, updating, creating and deleting some resources. Also, it will implement practices and techniques described in *Linked Data Platform* version 1.0 [6].

The LMS JavaScript library expects that the LMS server interacts with a GeoSPARQL endpoint. Figure 5 describes the planned interaction between a LMS instance and a Strabon endpoint [7]. Strabon is a semantic spatiotemporal RDF store developed in the context of the European FP7 project SemsorGrid4Env[3] (*Semantic Sensor Grids for Rapid Application Development for Environmental Management*). Strabon is built by extending the well-known RDF store Sesame[4] to manage thematic, spatial and temporal data that are stored in a backend repository in a PostgreSQL database extended with the PostGIS module [8]. PostGIS is a spatial database extension of PostgreSQL that adds support for geographic objects.

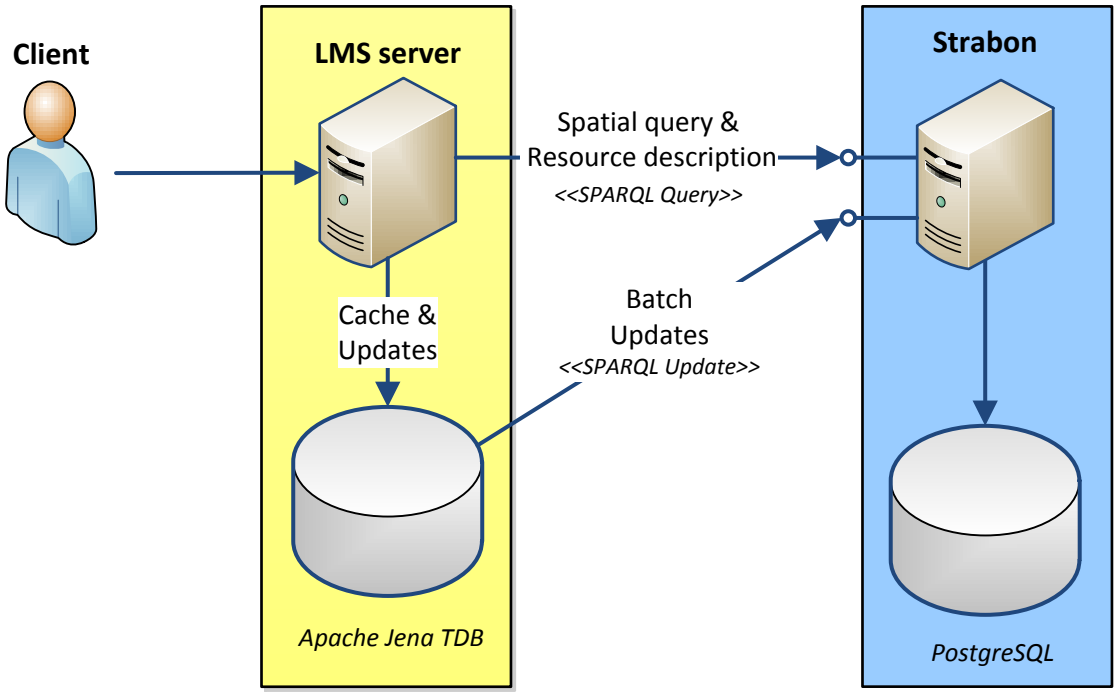

Figure 5 – Interaction between a LMS server and the access/update service

On behalf of the requests of the LMS JavaScript library, the LMS server makes spatial queries to the Strabon endpoint for retrieving spatial objects that may be present in maps and RDF descriptions of such resources. There are two typical cases:

- When the user request a machine-readable description of a map produced by a WMS GetMap request, the LMS server queries for RDF resources stored in Strabon that may appear in the spatial extent of the map (a bounding box) and belonging to the datasets shown in the map. For this task, the LMS server constructs a `SELECT` query that retrieves information about each matching resource. This is the interaction described above in Figure 2.
- When the user request a machine-readable description of a resource the LMS server makes a `DESCRIBE` query for such resource. This is the interaction described above in Figure 3.

[3] http://www.semsorgrid4env.eu/

[4] http://www.openrdf.org/

Since version 1.0, the LMS server will cache responses in a local triple store (Apache Jena TDB[5]) avoiding making redundant queries to Strabon. The LMS JavaScript library may request the LMS instance to create, to update or to delete some resources. These requests are stored locally in the Apache Jena TDB, and, if required, stored permanently in Strabon by means of SPARQL Updates. These modifications are performed as a batch operation periodically.

[5] http://jena.apache.org/documentation/tdb/

4 Conclusions

In this document, we have described the work made on designing and developing the first prototype of the Linked Map Service client, that is, a JavaScript library able to interoperate with the Linked Map Service (LMS). The library has been developed by extending the open source software OpenLayers in order to add semantic information recovering. This way the client is able to interact with the service to recover and draw maps (as a traditional WMS client) but, at the same time, it can also deal with the headers added by the LMS to the response, in order to obtain more information about the data related to the map, as well as send modifications to the server.

The LMS library will be deployed as a component of the client side a web portal enabling volunteers to review the reliability of such data comparing with official data. Reviews will be later available as Linked Data. This will allow linking the final report on the quality of VGI data to its sources. The web portal will be an outcome of the next Work Package 18. The final version of the LMS Java Script library will be available as part of the source code of the Web portal developed in such Work Package [4]. This Web portal will be part of the Linked Map web site[6].

[6] http://linkedmap.unizar.es/

Appendix I LMS Client JavaScript interface

This is the JavaScript interface specification of the LMS Client.

```
/**
 * Method: addVectorialInfo
 * Loads a file and draws features contained in it
 *
 * Parameters:
 * url - {String}
 * title - {String}
 * options - {Object}
 *  Supported properties:
 *  - parserOptions {Object} – parser options
 *  - featureOptions {Object} – feature options
 *  Supported properties:
 *  - Category: defines feature style
 *  - clusterize: clusterizes the layer
 *  - centerOnLoad: center in whole extent
 *  - processFeatures {Funcion} – specific function call during parse
 *  - vectorialStyle {Object} – layer style
 *  - appendFeatures {Boolean} – append features in other layer?
 *
 * Returns:
 * {Boolean}
 */
function addVectorialInfo(url, type, title, options)

/**
 * Method: updateVisibility
 * Updates the layer visibility
 *
 * Parameters:
 * layerIndex - {Array(Integer)} Index of the layer
 * visibility - {Boolean}
 * Returns:
 * {Boolean}
 */
function updateVisibility(layerIndex, visibility)

/**
 * Method: drawFeatures
 * Draws features in the map loading a GeoJSON file.
 *
 * Parameters:
 * json - {Object} Path of the geoJSON file
 * options - {Object}
 *  + category - {String} defines feature style
 *  + clusterize - {Boolean} clusterizes the layer
*/
function drawFeatures(json, options)
```

```
/**
 * Method: deleteFeatures
 * Deletes the features asociated with the defined layer
 *
 * Parameters:
 * index - {Integer} Layer index
 */
function deleteFeatures(index)

/**
 * Method: locateSearch
 * Locates and draw in the map the result of the search
 *
 * Parameters:
 * name - {String} Result name
 * x - {Integer} Result X coordenate
 * y - {Integer} Result Y coordenate
 * srs - {Integer} Coordenate Reference System
 */
function locateSearch(name, x, y, srs)

/**
 * Method: deleteSearch
 * Deletes the last search result draw in the map
 *
 * Returns:
 * {Boolean}
 */
function deleteSearch()

/**
 * Method: defineEvents
 * Define the capture of an event or a list of events from the viewer
 * and provides functions to be executed after the capture of the event.
 *
 * Parameters:
 * object - {Object} Object affected by the events.
 * eventList - {<String>|Array<String>} Event or list of events defined
to be captured.
 * functionList - {function|Array<function>} Function or list of
functions defined to be executed
 * when the events are captured.
 *
 * Returns:
 * {Boolean}
*/
function defineEvents(object, eventList, functionlist)
```

References

[1] J. Barrera and F. J. Lopez-Pellicer, “D17.1 Call 2: Linked Map Read-write Linked Data enabled OGC Web map server,” PlanetData, 2014.

[2] J. de la Beaujardiere, Ed., “OpenGIS® Web Map Server Implementation Specification,” Open Geospatial Consortium Inc., OGC 06-042, Mar. 2006.

[3] T. Heath and C. Bizer, *Linked Data: Evolving the Web Into a Global Data Space*, vol. 1, no. 1. Morgan & Claypool Publishers, 2011.

[4] J. Barrera and F. J. Lopez-Pellicer, “D18.2 Call 2: Linked Data Platform Beta version ,” PlanetData.

[5] M. Perry and J. R. Herring, Eds., “OGC GeoSPARQL,” OGC 11-052r4, Sep. 2012.

[6] S. Speicher, J. Arwe, and A. Malhotra, Eds., “Linked Data Platform 1.0,” *W3C*, 30-Jul-2013. [Online]. Available: http://www.w3.org/TR/2013/WD-ldp-20130730/. [Accessed: 28-Jan-2014].

[7] K. Kyzirakos, M. Karpathiotakis, and M. Koubarakis, “Strabon: A Semantic Geospatial DBMS,” in *Lecture Notes in Computer Science*, vol. 7649, no. 19, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 295–311.

[8] C. Strobl, “PostGIS,” *Encyclopedia of GIS*, pp. 891–898, 2008.