Lessons learned on composing OGC services in spatio-temporal GIS applications

Michael Mock, Rajat Arora, Bishal Pantha, Hans Voss
Fraunhofer IAIS
Schloss Birlinghoven
Sankt Augustin, Germany
michael.mock@iais.fraunhofer.de

Abstract

This paper presents our experiences with using OGC services in developing a web application that visualizes data from 7795200 timed data values on a street network composed of more than 46.000 segments. Web clients, also running on mobile devices, are enabled to perform various kinds of spatial selections on the street network. As result, the visualization of aggregated data values as overlays provides an animated overview on the development of the data values over time. Multiple OGC Services, as well as a spatial database, are involved: WFS, WMS, SOS, and an OpenLS service for driving zone computation. We compare a client-based architecture, in which different OGC services are directly accessed from a web-browser and data is aggregated on the client side, against a server-based architecture using server side composition of services. We provide extensive performance analysis for both architectures. We conclude by giving design guidelines for the different options, depending on data amounts and security requirements. The complete application has been successfully presented on CeBIT 2012.
Keywords: Web GIS, OGC Services, Visual Analytics of spatio-temporal data, Performance Analysis, Mobile Devices.

1 Introduction

Visual analysis of spatial and temporal data for producing interactive animations and time series is established as well in research (see [2,3] for an overview) and in various data analysis products. With the advent of web-based systems, there is a growing demand for visual analysis provided by web-based client/server systems and even on mobile devices. The Open Geospatial Consortium (OGC) has successfully introduced various standards for web-services dealing with spatial data, among which the most prominent are the Web Feature Service (WFS) and the Web Map Service (WMS) are the most prominent, as well as services for handling temporal data, e.g. the Sensor Observation Service (SOS).

In this paper, we explore and evaluate the implementation of a web-based system for the interactive analysis of spatio-temporal data using OGC Web services. In particular, scalability in terms of amount of data to be handled is considered: the implemented example application allows to spatially select from 46.000 spatial features and to visualize time-series of up to 24 data points per selected features as an animation over the web. We implement and evaluate two basic variants of the application:

- In a client-based architecture, different OGC web-services (WFS, OpenLS, SOS [18,16,17]) are accessed directly by the client and data is aggregated on the client side (running on a PC or on a tablet).
- In a server-based architecture, a dedicated application server uses different OGC services (WFS, OpenLS) and a spatial database in the back-end to retrieve and aggregate the data, which is then made available to the client through a WMS service [19].

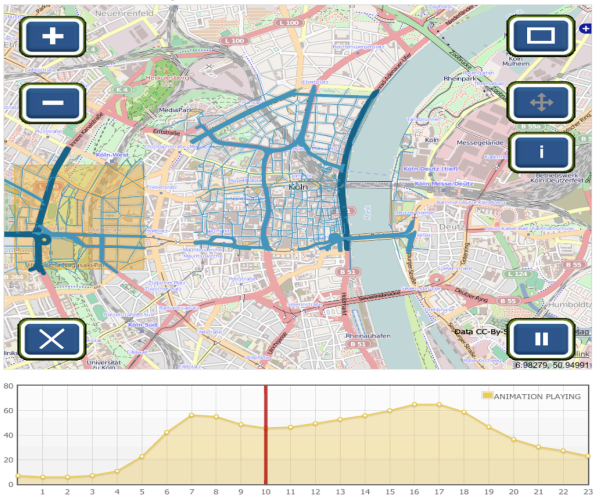
We provide an extensive performance evaluation and report from our experiences regarding performance and conformance of OGC services and libraries such as OpenLayers [9] and Geotools [10]. We also compare OGC service performance against the option of using a spatial database in the dedicated application server. So far, only few work exists on experiences with the composition of OGC services for spatio-temporal data in practical applications [1,6,7,11,12]. We extend this previous work by providing detailed performance insights in a concrete implementation, also comparing PC and tablet based client-versions.

The reminder of the paper is structured as follows: section 2 describes the thick client-based architecture and its performance, section 3 describes the server-based architecture and its performance (also in comparison to the thick client version), and the conclusions section 4 summarizes the lessons learned.

2 Client-based architecture

Figure 1 shows the web client interface of our application. Please note that the client interface looks identical for the client-based and the server-based implementation. It is implemented with JavaScript using OpenlLayers for accessing WMS and WFS. The interface consists of two areas: the upper map area and the lower time graph area. In the map area, the user can spatially select a region of interest, either by drawing a bounding box, a polygon or by asking for a so-called "driving zone", which corresponds to an area which is reachable by car given a starting point and a maximum driving time. According to the user selection, the application will firstly determine the street segments (maximally 46400 for the complete city) in the selected region of interest based on Navteq street data [14]. The selected set of street segments is denoted as "spatial selection" in the following. Secondly, for each segment in the spatial selection, the application provides a time series of data, which, in our case, gives the average number of persons passing the street segment in a certain time frame (denotes as "frequencies" in the following).

Figure 1: Web Client Interface.



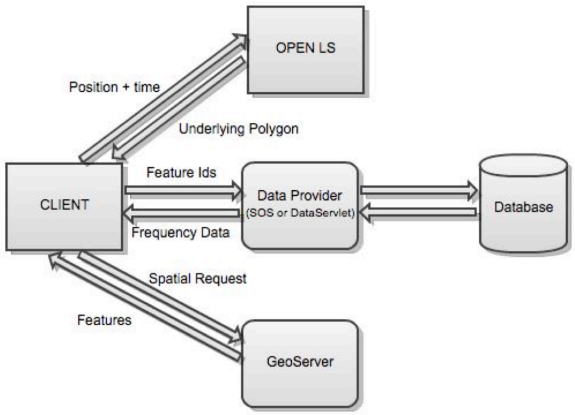
Currently, we support two modes of temporal aggregation of time frames: in "day mode", we give 24 frequencies for each hour of an average day, and in "week mode", we give 7 frequencies for each day of an average week. Other, more complex temporal aggregations are possible in general. The interactive lower area of Figure 1 shows the spatial average of the frequencies over time in a time graph. In the example, the average frequencies of the spatial selection over a day are shown. The particular frequency for each street segment for a specific hour of the day is visualized in a map overlay layer by the thickness in which the street is shown on the map. A click on the time graph (e.g. at hour 10 in the example) visualizes the frequencies for the selected instance of time on the map. In other words, there is one overlay layer for each instance of time in the time series, and selecting a specific time instance makes this layer visible, while hiding the others. Instead of clicking on the time graph for selecting a specific time instance, the selected time instance moves forward periodically, thus yielding a movie like animation of frequencies on the map.

2.1 Client-based architecture design

In the client-based approach shown in Figure 2, the combination of OGC web services such as WFS, SOS and OpenLS is accessed directly from client side to fetch the spatial temporal data for computing the map overlay layers and the time graph on client side.

The SOS was considered as good candidate for aggregating temporal data while WFS offers more capabilities for handling spatial queries. Therefore, the basic architecture of this approach consists of WFS-SOS combination. The OpenLS is used to determine a polygon that corresponds to a driving zone. The OpenLS is provided by [15] and just used to show the ease of functional integration based on OGC services, but it is not part of our performance evaluation. The WFS (we use the geoserver implementation [8]) is used to query the feature of interest for the region selected by user. The street layer is created in Geo-Server that imports the street segment table from Navteq data [14]. The SOS is being used for querying corresponding frequencies. The SOS implementation described in [13] was used for carrying out this experiment. The streets were treated as sensors and corresponding frequency values as observations of particular sensors at given time instance.

Figure 2: Component overview of the client-based architecture.



Our SOS implementation [13] allows for one sensor only in a single getObserverstions request (in contrast to others, e.g. [12]). To overcome this limitation, which caused a much too high amount of message traffic, another data servlet was written which takes multiple street Ids as input and responds with corresponding observations. Therefore, in this approach, there two types of data providers (SOS and data servlet), are used to provide frequency data values for corresponding streets requested. Measurements are reported on the data servlet version.

Figure 3: Sequence diagram for the client-based architecture.

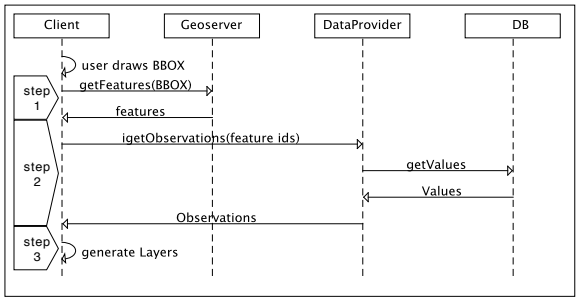


Figure 3 shows the typical flow of the application. When the user selects a region of interest on the map, OpenLayers provides the corresponding spatial filter once the selection is drawn. This spatial filter is sent via a WFS GetFeature Request to the Geoserver, yielding a list of features (spatial selection) about streets (step 1 in Figure 3). The client obtains the street Ids by parsing the received feature list. These street Ids are further used to fetch the frequency data values from data provider (step 2 in Figure 3). Using this frequency data and the geometries received previously from the Geoserver, the client populates the local overlay layers with newly created vector features whose thickness represents the frequency of the given street segment (step 3 in Figure 3).

2.2 Client-based architecture evaluation

Figure 4a, 4b: Client-based Performance (PC)

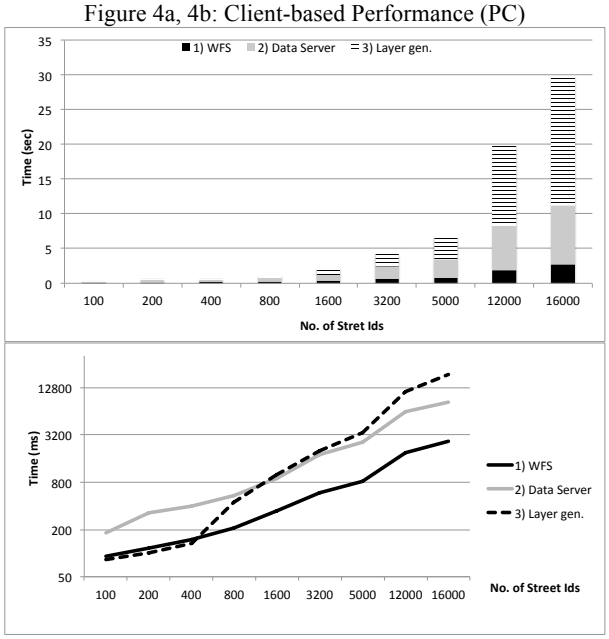
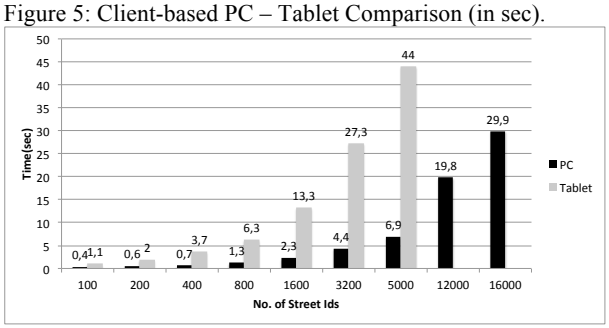


Figure 4a) shows the total time in ms taken by steps 1-3 (refer to Figure 3) on a PC for "day mode", this is when generating 24 local layers (on a 1.73 GHz Intel i7 QM 820 machine, 4 GB RAM, in the Chrome browser, see [4] for more browser comparisons). Note that the x-axis has a logarithmic scale. Basically, the total time is increasing linearly with the number of streets. We notice from Figrue 4b) (with a logarithmic scale on the x-axis) that almost all steps increase linearly, but that step 3 (local layer generation) dominates over the communication steps 1 and 2 with increasing number of streets. As each browser restricts the computation time of local JavaScript/AJAX by timeout, we reach a limit when the number of streets gets too high. On the test machine, we managed to handle up to 16.000 streets with 24 data points each. Figure 5 shows the timing comparison with the client running on tablet (Samsung Galaxy 10.1, 1GHz, 1GB RAM), in which the maximum is reached after 5.000 number of streets.

Figure 5: Client-based PC – Tablet Comparison (in sec).

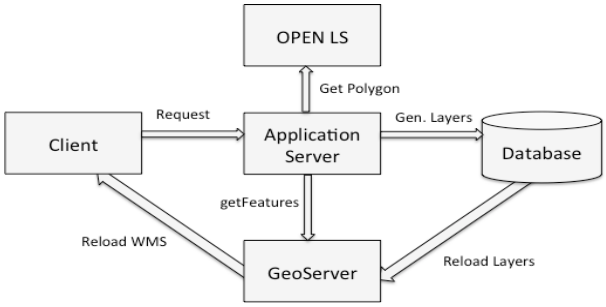


In sum, the client side performance is surprisingly fast and robust against usage of different browsers and platforms. 16.000 street segments in a single selection process is about a third of the maximum number of 46.000 street segments for the complete city. The user can "add" multiple selections to cover the complete area. A performance comparison with the server-based design (which of course scales up to the complete number of streets in a single selection) will be given in section 3.

3 Server-based architecture

3.1 Server-based architecture design

Figure 6: Component overview of the server-based architecture.



The server-based architecture shown in Figure 6 is based on a napplication server, which encapsulates a geo-server, a spatial database, and an OGC OpenLS server (for driving zone computation). The client interface is the same as depicted in Figure 1 and also the visualization and animation technique is the same as in the client-side architecture, based on showing different overlay layers for different points of time. However, all overlay layers are computed on the server side by the application server and made available as raster image data to the client via a WMS service.

Figure 7: Sequence diagram for the server-based architecture.

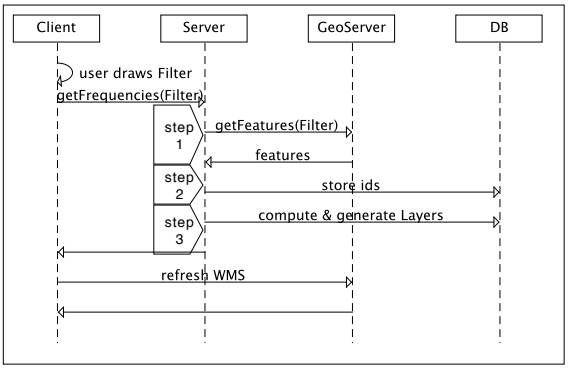


Figure 7 shows the execution sequence in the server-side architecture, which essentially executes the same steps 1-3 as shown in Figure 3, with the major difference, that all steps are executed on the server side. Step 1 determines the spatial selection; step 2 stores the street ids of the spatial selection in a specific database table, and step 3 computes the overlay layers as database views visualizing the frequencies as thickness. These views are mapped via WMS to the client. We additionally implemented alternatives for computing the spatial selection (step 1):

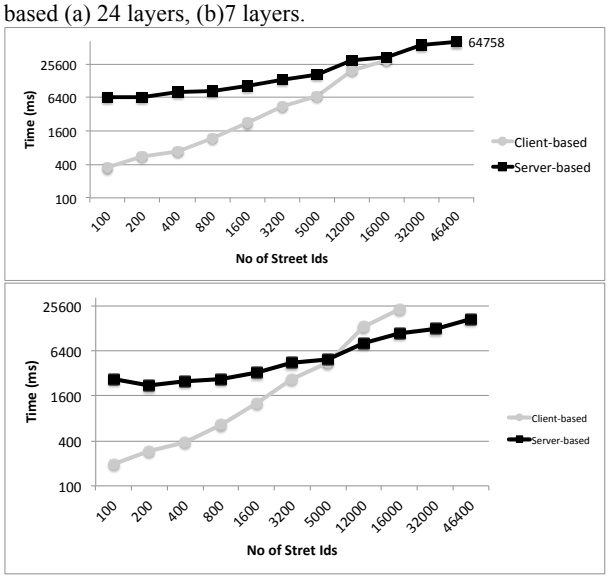
a) WFS GetFeatures with external Geoserver (Figure 7).
b) WFS GetFeatures with the Geotools library operating on a layer stored in the spatial database.
c) Native operations of the spatial database directly.

3.2 Server-based architecture evaluation

The performance of steps 1-3 mentioned above with all three implementation alternatives is shown in Table 1, more details are described in [20]. The application and the Geoserver run on an Intel i7 8 Core CPU 1.73 GHz, 4 GB RAM Ubuntu machine using an external Oracle database 11g with spatial extension, running on an Intel Xeon X5670 24 core processor machine at 2.93 GHz and 128 GB RAM..

From Table 1 we can see that basically all steps increase linearly in time with the number of street ids selected. We end up with 46.000 street ids because this is the maximum number for the complete city, but not a limitation of the architecture. Also, we see that alternative a), which includes communication with a WFS server in the back end, takes roughly twice the time than the other two alternatives for computing the spatial selection (step 1), while alternative b) and c), both using the spatial database directly, perform quite similarly. Native access to the spatial database (alternative c) outperforms the access via the Geotools library (alternative b) only slightly, making Geotools an attractive option that combines the performance of a database with the ease of use and abstraction level of OGC spatial operations.

Figure 8a, 8b: Performance in ms client-based vs. server-based (a) 24 layers, (b)7 layers.



Figures 8a) and 8b) compare the overall execution time measured on the client (sum of steps 1-3 in Figure 2) for the client-side and the server-side architecture (alternative c) in day and week mode, respectively. Note that the charts have a logarithmic scale on both x- and y-axis. First of all, we see that the server based still performs well for the maximum number of street segments (46000), while the client gets unstable (crashes non-deterministically) after 16.000 street segments. Both exhibit a rather linear behaviour, with the server-side solution experiencing a smaller gradient and scaling better for large numbers, but having a significant basic over-head for lower numbers of street segments. Especially, when working with a higher number of layers (24 in Figure 8a), this basic overhead, which includes layer generation on the server and mapping on the client, is about 6,4 seconds. In contrast, the client-side solution scales linearly in time al-so for small number of segments, thus yielding a break even point be-tween the two solutions at about 16.000 street ids (roughly a third of the complete city), which is also the maximum number the client-side architecture can handle in a single selection request. When having fewer layers (Figure 8b), the basic overhead in the server-side architecture is much smaller (3.2 seconds), such that the break event point is already reached at about 5.000 street ids.

4 Conclusions and lessons learned

We have learned that the use of OGC standards in spatio-temporal web-based applications provides a high degree of interoperability and many practical benefits when programming – in particular because of the seam-less and easy interworking of OpenLayers and Geotools libraries. The server-side composition of OGC services is easy to program and does not much slow-down the application compared to using the native, more complex interface of a spatial database (see Table 1). Animation and visualization of spatio-temporal data via providing different overlay layers for different points of time worked well with the OGC services and for both of the architectures that have been investigated.

The client-side architecture for service composition turned out to perform surprisingly well and is robust, but as expected is outperformed by the server-side architecture for large number of features (see Figure 8). For very large number of features, server side aggregation, possibly according to different zoom levels, is definitely needed. Please note that the

Table 1: Performance break-down for the server-based architecture (in ms).

a) Geoserver	Step 1	502	1473	2012	3400	6058	12049	16782
	Step 2	126	367	352	421	1391	3248	2551
	Step 3	485	1531	2049	2794	4355	6845	9535
	Sum	**1113**	**3371**	**4502**	**6615**	**11804**	**22141**	**28867**
b) GeoTools	Step 1	359	617	673	843	1134	1805	2339
	Step 2	105	279	308	493	950	2221	2511
	Step 3	493	1569	2067	2800	4168	7095	9503
	Sum	**957**	**2465**	**3047**	**4136**	**6252**	**11122**	**14353**
c) Oracle	Step 1	-	-	-	-	-	-	-
	Step 2	397	546	542	723	1032	1644	1772
	Step 3	528	1573	2149	3170	5030	9305	10870
	Sum	**925**	**2119**	**2691**	**3893**	**6062**	**10949**	**12642**
No. of Street Ids		100	2000	4000	8000	16000	32000	46400

client-side architecture requires all data to be delivered to the client, making it unpractical when data is confidential for some reasons. In our example, even delivering the geometry of the street segments to the client is only feasible when the user holds a corresponding Navteq license. In contrast, the server-side architecture aggregates data on the server-side and delivers aggregated raster data to the client only. Both, client-server-side architecture, work well on PCs and tablets, with the client-side architecture reaching its limits faster on the tablet than on the PC (refer to Figure 5). On the other hand, the client-side solution may scale better in number of concurrent clients because the computational effort is distributed automatically rather than being centralized as in the server-side architecture. Being fast enough for a few thousand features, client-side architectures can also profit from advanced visualization libraries such as

Regarding the use of OGC SOS for handling temporal data, our experience is twofold: on the one hand, we see that in principle, a client-based architecture can integrate temporal data easily in this way, on the other hand, we experienced performance problems that were overcome by re-placing the SOS with a dedicated data server. Also, library support for accessing SOS is by far not elaborated as for accessing WFS or WMS. We conclude that the use of SOS in a setting for visualization of spatio-temporal data is only necessary when the data is actually coming from re-al, possibly non-uniform sensors via the transactional interface in real-time to the SOS. When the data is already contained in an application specific database, as it was in our case, we experienced no benefit in using SOS. In particular, WFS turned out to be fast enough for spatial selection (refer to Figure 4), such that the spatial query functionality offered by the SOS GetObservations request was not needed either.

References

[1] Arash Amiri. Automatic geospatial web service composition for developing a routing system. Master Thesis, Lund University, 2012.

[2] Natalia Andrienko, Gennady Andrienko, Peter Gatalsky. Exploratory spatiotemporal visualization an analytical review. Journal of Visual Languages and Computing, 14 (6):503-541, 2003.

[3] Gennady Andrienko, Natalia Andrienko Hans Voss. GIS for Everyone: The CommonGIS Project and Beyond. In Peterson, Michael (ed.) Maps and the Internet. (131-146) Amst: Elsevier Press. 2012.

[4] Rajat Arora. Building a GIS Web application using OGC web services. Master Thesis, University of Bonn, 2012.

[6] Timothée Becker, Barend Köbben, Connie Block. Visualizing Moving Object Data using WMS Time and SVG SMIL Interactive Animations. SVGOpen 2009: 7th international conference on scalable vector graphics, 2009.

[7] Nengcheng Chen, Liping Di, Genong Yu, Min Min. A flexible geospatial sensor observation service for diverse sensor data based on Web service. ISPRS Journal of Photogrammetry and Remote Sensing, 64(2):234 – 242, 2009.

[8] http://geoserver.org/, last access Jan. 29th, 2013.

[9] http://openlayers.org/, last access Jan. 29th, 2013.

[10] http://www.geotools.org/, last access Jan. 29th, 2013.

[11] Markus Innerebner, Michael Bohlen, and Igor Timko. A web-enabled extension of a spatio-temporal DBMS. In Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, GIS '07, pages 34:1– 34:8, New York, NY, USA, 2007. ACM.

[12] Simon Jirka, Carsten Hollmann, Christoph Stasch. SOS vs. WFS Coupling 52 degree North's Sensor Observation Service and Geoserver's Web Feature Service. FOSS4G, 2010.

[13] Roland Müller, Manuel Fabritius, and Michael Mock. An OGC compliant Sensor Observation Service for mobile sensors. AGILE International Conference on Geographic Information Science, 2011.

[14] NAVTEQ. NAVSTREETS Street Data Reference Manual v3.1, Jan. 2009.

[15] Pascal Neis, Leonhard Dietze, Alexander Zipf. A Web Accessibility Analysis Service based on the OpenLS Route Service. AGILE International Conference on Geographic Information Science 2007.

[16] OGC. OpenGIS Location Services (OpenLS): Core Services. OpenGIS Implementation Specification OGC 07-074, Open Geospatial Consortium Inc., Sept. 2008.

[17] OGC. OGC Sensor Observation Service 1.0, Open Geospatial Consortium, June 2007.

[18] OGC. Web Feature Service Implementation Specification. OpenGIS Implementation Specification OGC 04-094, Open Geospatial Consortium Inc., May 2005.

[19] OGC. OpenGIS Web Map Server Implementation Specification. OpenGIS Implementation Specification OGC 06-042, Open Geospatial Consortium Inc., March 2006.

[20] Bishal Pantha. Building a GIS Web Application using OGC Services along with Spatial Selection and Driving Zone Computation and Evaluation. Master Thesis, University of Bonn, 2012.

[21] Visualization library: http://d3js.org/, last access Jan. 29th, 2013.