GML2GEOJSON: PYTHON MIDDLEWARE SERVICES FOR A WEB ATLAS INTEGRATED IN A SPATIAL DATA INFRASTRUCTURE

B.J. Köbben [*]

Faculty of GeoInformation Science and Earth Observation (ITC), University of Twente, Netherlands - b.j.kobben@utwente.nl

KEY WORDS: WebCartography, Middleware, Python, OGC standards, GeoJSON, GML

ABSTRACT:

In this paper we describe the concept of an Atlas Services middleware layer that offers intermediate services to enable loose coupling of a Spatial Data Infrastructure (SDI) with a client–side mapping component.
We first describe the challenges in creating quality maps if these have to be generated within a Spatial Data Infrastructure. We introduce a theoretical framework and a proof-of-concept architecture, developed at ITC to overcome some of these challenges. Part of this architecture are the middleware services mentioned above. We discuss in general the philosophy of the service layer, and in particular how we implemented a GML–to–GeoJSON proxy using Python.

1. INTRODUCTION: ATLAS MAPPING IN AN SDI

In Spatial Data Infrastructures (SDIs) many web services, of different types and with different outputs, come together. Some of these outputs are maps, but it is challenging to maintain a good cartographic quality when combining several of these maps. In a proper Web Atlas we want to present a large amount of information, that should be comparable. And this information aims to "tell a story", i.e., enable a comprehensive understanding, which exceeds the understanding obtained from the separate maps on their own. An atlas should as a whole become more than the sum of its parts. Such an atlas would benefit from the up-to-date data in the SDI, and the SDI would benefit from the high-quality integrated visual summaries of the available spatial data.

The SDI technology in itself will allow mapping the different data sets simultaneously. The typical setup in current SDIs is that each separate SDI node, or dataset, has its own mapping service, usually implemented as a OGC Web Map Service (WMS). In this situation, depicted in Figure 1a, both maps can be combined in a (web) client, but because each layer had their styling and symbolising done in isolation, the combination usually is sub-standard, because the maps, and possibly the data themselves, are incompatible.

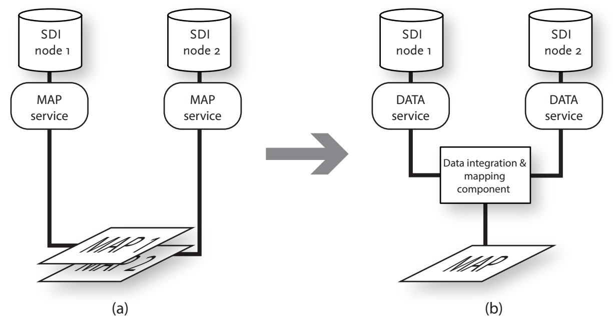

Figure 1. Mapping in an SDI environment using MAP services (a) and using DATA services (b).

[*]Corresponding author

This problem has been recognised early on (Harrie et al., 2011), and several solutions have been researched, e.g., optimising the cartographic quality of the map services themselves (Toomanian et al., 2013). Another strategy, depicted in Figure 1b, is to have one integrated mapping component, separate from the SDI nodes, that constructs maps using the *data* services of these nodes, typically OGC Web Feature Services (WFS). This is the method we use in our testbed.

2. THE NATIONAL ATLAS OF THE NETHERLANDS TESTBED

At ITC, we have been experimenting with a theoretical framework and a proof-of-concept architecture, which we argue could overcome some of the challenges mentioned and indeed allow for a web atlas to be an integral part of an SDI. We have described the architecture and the client-side component in an earlier publication (Köbben, 2013). Here we introduced the experimental third edition of the National Atlas of the Netherlands as our test bed for trying out the theoretical framework and architecture in a real-life use case. We described the Atlas Map Viewer component we created as a web application, using HTML5 technology and the D3 library, and we have made the proof-of-concept available on the web (Website Dutch National Atlas / Nationale Atlas van Nederland, 2017).

In the following sections we will briefly re-hash the principle architecture, and then describe the concept of the Atlas Services middleware layer that offers intermediate services to enable loose coupling of the Spatial Data Infrastructure with the client-side mapping component. We will discuss in general the philosophy of the middleware service layer, and in particular how we implemented a GML–to–GeoJSON proxy using Python.

2.1 Architecture

As explained in section 1, we consume data from Web Feature Services (WFS), to overcome the cartographic map matching problems. These WFSes are part of the Dutch National GeoData Infrastructure (NGDI), the official Netherlands Open Data geoportal, which includes a broad range of possible data services. The two services specifically mentioned in Figure 2 are the ones we

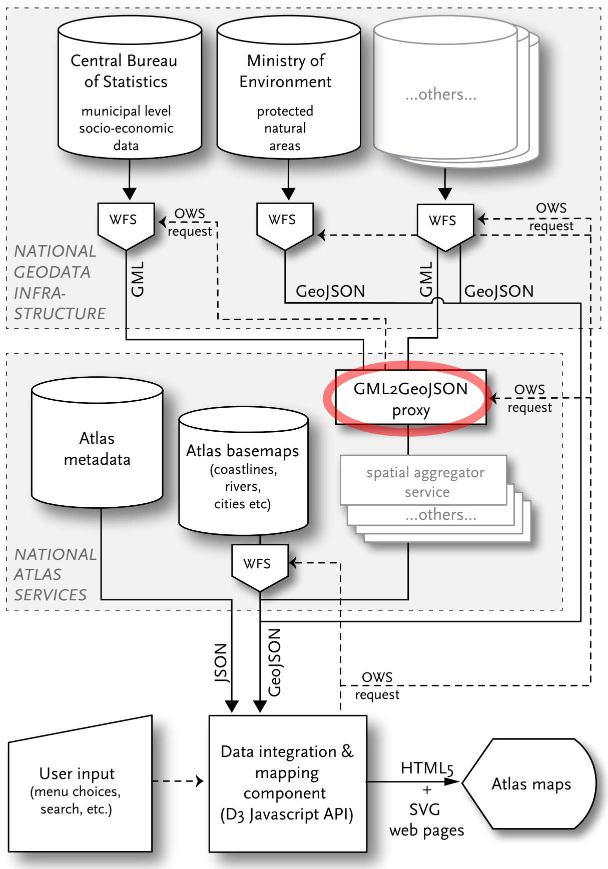
Central Bureau of Statistics
municipal level socio-economic data
Ministry of Environment
protected natural areas
...others...
WFS
OWS request
WFS
WFS
OWS request
NATIONAL GEODATA INFRA-STRUCTURE
GML
GeoJSON
GML
GeoJSON
GML2GeoJSON proxy
OWS request
Atlas metadata
Atlas basemaps (coastlines, rivers, cities etc)
spatial aggregator service
...others...
WFS
NATIONAL ATLAS SERVICES
JSON
GeoJSON
OWS request
User input (menu choices, search, etc.)
Data integration & mapping component (D3 Javascript API)
HTML5 + SVG web pages
Atlas maps

Figure 2. The architecture of the National Atlas of The Netherlands testbed (from Köbben, 2013), with the GML2GeoJSON proxy highlighted.

happen to use in the prototype at the time of writing; which services are actually used by the atlas is determined by the settings in the Atlas metadata.

This metadata is in fact the main atlas *service configuration* component, at the moment implemented as a simple JSON datastore in the National Atlas Services layer. It includes settings for each map, such as: where and how to get the data (service URL, method, service parameters, format), if and how to classify the data, which map type to use, the symbol colours and sizes, etcetera.

These settings are used by the mapping component, that combines and portrays the data. For reasons described in the earlier publication (Köbben, 2013), we have implemented this client–side, as a JavaScript web application, using HTML5 technology and the D3 library (Bostock et al., 2011, D3 website, 2017) to create data-driven graphics.

This approach we have chosen has one considerable drawback: The standardised OGC WFS services typically provide their data in the Geography Markup Language (GML). GML is a versatile and powerful format, but it is also a complicated and verbose one. And because we can and want to use a very wide range of existing data services, we therefore can expect GML data of different versions and with a large variation in GML application schemata. We decided it would not be sensible to try parsing this client–side in JavaScript. Instead we chose to supply the mapping component with GeoJSON data. This geographic extension of the JavaScript Object Notation format is light-weight and optimized for use in client–side web applications. Although it is at present an IETF standard (Butler et al., 2016), and some services in the NGDI do actually supply data in GeoJSON format, many others only support GML output, as that is the format required by the OGC WFS standard.

To overcome that limitation, we have introduced the GML2GeoJSON proxy, highlighted in Figure 2, as a middleware service in our National Atlas Services layer.

2.2 The Atlas Middleware

The GML2GeoJSON proxy is needed because the client-side mapping component can only handle GeoJSON for the spatial data, and that would only work if the system would be *tightly–coupled*, to specific data services only. In order to maintain a *loosely–coupled* setup, the conversion from GML to GeoJSON was realised as an independent proxy service, that could in theory be used by anyone needing such a conversion. The National Atlas Services is the middleware layer we use to provide for several of such intermediate and supporting services.

Another component in this National Atlas Services layer is the Atlas basemaps service. This serves data for several map layers that are used repeatedly, such as coastlines, major waterways and administrative borders. This enables us to provide a common look and feel to the maps. Note that this is also implemented as a loosely-coupled, standard WFS, and as such is a fully independent stand–alone webservice node. It could therefore be considered as part of the NGDI layer just as well.

At present, the metadata is included in the National Atlas services layer as a static JSON datastore. For reasons elaborated in (Köbben, 2013), the metadata settings are maintained 'by hand'. Because of this the National Atlas cannot function without an editorial team. This staff is responsible for the cartographic quality of the atlas, and for example should also keep track of new geospatial information being made available by national providers, as well as taking account of the changing needs and interests of the users. Several researchers have been looking into middleware components to automate some of the editorial tasks. For example (Zumbulidze, 2010) has investigated automated updating mechanisms, and (Chanthong et al., 2012) proposed business processes to securely manage the administration; but as these both were of an experimental nature they have not been integrated yet in the current system.

The middleware services can be implemented in any manner, as long as they use the same OGC and other standards employed in the rest of the system. The Atlas Basemaps service is a MapServer WFS instance, the metadata is (as explained before) a simple JSON file. The GML2GeoJSON proxy is a Python CGI application, and in the next section we will elaborate how this service was implemented.

3. THE GML2GEOJSON SERVICE

We investigated various possibilities to implement the conversion from GML coming from existing WFS services to GeoJSON to be consumed by our JavaScript mapping client. One option was to use an existing WFS implementation that is capable of returning GeoJSON as its output format, such as GeoServer (`http://geoserver.org/`). Because this software can also act as a WFS *client*, a so–called 'cascading server' can be set up,

WFS	size (in Mb)		load time (in s)		load time
	GML3	GeoJSON	WFS GML3	GML2GeoJSON	% increase
A: 31 airports	0.58	0.5	0.26	0.77	196.2
B: 12 provinces	2.9	3.9	6.75	9.5	40.7
C: 431 municipalities	73.7	98.4	166.2	230.4	38.6

Table 1. Results of limited performance testing (load times are averages of 50 attempts).

meaning the input data for its WFS can be another WFS service. We tested that option and got promising results, both in robustness and performance. But we decided against further development for two reasons: Firstly, using a complex software suite such as GeoServer for this purpose alone seems like overkill, and would introduce serious maintenance and installation efforts; Secondly, it would move part of the National Atlas metadata to the GeoServer administration system, because that is where you would have to set up the details of how the GeoServer WFS client would contact the original WFS service from the NGDI. We prefer to keep all metadata in one coherent system.

While looking into and testing GML to GeoJSON conversion in general, we had successfully used the `ogr2ogr` command line programme that is part of the GDAL/OGR library (`http://www.gdal.org/`). This is the foremost open source software library for reading and writing raster and vector geospatial data formats, and provides simple and powerful abilities for data conversion. We therefore decided to implement a simple Python CGI application ourselves, to wrap the `ogr2ogr` functionality in a web service. The UML sequence diagram in Figure 3 shows how our setup functions.

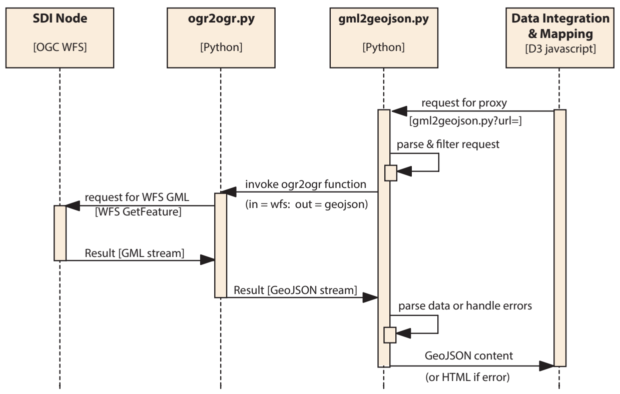

Figure 3. UML sequence diagram of the GML2GeoJSON service.

The client–side mapping component will have retrieved the source for the data from the ATLAS metadata store. In case this is a WFS returning GML, the URL pointing to this service will be sent to our proxy, by means of adding the original URL to the proxy url: `.../gml2geojson.py?url=originalURL`

The Python CGI application parses the original URL, and does some limited checking and filtering. The service then invokes the `ogr2ogr.py` module. This is a direct Python port of the original `ogr2ogr` C++ code shipped with GDAL/OGR, and can be downloaded from the code repository at `github.com/sourcepole/ogrtools/`.

The `infile` parameter of `ogr2ogr` is provided with the filtered URL, preceded with the keyword `wfs:` to invoke the OGR WFS client. The filtering mentioned includes retrieving the `TYPENAME` parameter from the original URL, as that has to be supplied separately to `ogr2ogr` in the `layername` parameter. The `ogr2ogr` module then retrieves the GML output from the original data service and converts the resulting output into GeoJSOn and feeds it back to our proxy service. This resulting data is then returned to the javascript client, in the form of a mappable GeoJSON stream, or, when appropriate, as an HMTL encoded error message.

The URL for the original WFS service can be any valid request. The filtering and parsing mentioned earlier might therefore seem to be unnecessary, but is implemented nevertheless for the following reasons:

- The request may be a valid OGC WFS request, as described in the OGC WFS standard (OGC, 2010), but one that does not make sense as input for the National Atlas, such as the GetCapabilities or DescribeFeature requests. Therefore the system checks if the URL is a proper WFS GetFeature request.
- Although we implement the 2.0 version of the standard, we do not support the request of more then one data layer at a time that this version introduced (using the TYPENAMES parameter instead of TYPENAME). The reason is that the `ogr2ogr` module and the javascript mapping client are both not equipped to handle multi-layer input;
- The WFS GetFeature standard includes a parameter RESULTTYPE, that is used to either ask for actual data output (`=results`), or only for a count of the items to be returned (`=hits`). The latter can not be processed in our set-up, and thus will be caught in an error message.

3.1 Results

While developing the GML2GeoJSON proxy, we had mixed experiences with the performance of the system: While the system worked and was generally robust, the throughput was not impressive, to say the least. We have undertaken limited testing, which resulted in the data depicted in Table 1. We used three WFS data services: A is a very simple set of 31 airports modelled as point data, with only a few attributes (name and ID), loaded from an ITC internal server running MapServer. B is the Dutch Central Bureau of Statistics outlines of the 12 provinces of the Netherlands, served by a GeoServer WFS in the NGDI, with only the names and IDs as attributes. C comes from that same server, and consists of the 431 municipalities of the Netherlands, with a set of 47 “key statistics” for each object.

In general one can see that WFS requests of large datasets are not efficient to start with, as has been remarked before on by various authors (Giuliani et al., 2013, a.o.) As can be concluded from the results, retrieving the data through the GML2GeoJSON proxy adds, not surprisingly, considerable processing time to any request. But one can also observe that this extra time becomes relatively smaller if the data to be retrieved is larger.

Although certainly not fast, we did find the system to work reliably and without errors. Note that we have only tested it properly for the limited use cases in the National Atlas testbed. Therefore, although in theory it should work for any valid WFS request, in practice it should be treated very much as a bèta version for any purpose outside our testbed.

The Python source code is available as Open Source (under the GPL license) at out github repository (`https://github.com/kobben/gml2geojson`) and we welcome anybody to help us develop it further.

4. CONCLUSIONS

The original research funding that started our efforts in creating the National Atlas of the Netherlands Testbed already ended in 2009. Since then, we have been continuing the development of the National Atlas prototype as an informal project, and the progress has been slow and limited in scope. We consider it as an excellent testing ground for our students' and staff's research in the field of Spatial Data Infrastructures, Spatial Services development and Web Cartography.

The GML2GeoSJON proxy described in this paper is an example of such effeorts, and we foresee the implementation of more middleware services. Currently work is ongoing on a spatial aggregator service, that for example would perform the generalization of socio-economic data at the municipal level into higher level provincial data, including spatial aggregation of the geometries as well as attribute aggregation by e.g., averaging, summarizing or other statistics.

We think that the current proof-of-concept already demonstrates that high-quality atlas mapping using services from a national SDI is feasible, and that is potentially provides many advantages in up-to-dateness, flexibility, extensibility as well as interoperability and adherence to standards.

REFERENCES

Bostock, M., Ogievetsky, V. and Heer, J., 2011. D3: Data-Driven Documents. *IEEE Transactions in Visualization & Computer Graphics (Proc. InfoVis).*

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S. and Schaub, T., 2016. The GeoJSON Format. Standards specification - Proposed Standard RFC 7946, Internet Engineering Task Force (IETF).

Chanthong, B., Köbben, B. and Kraak, M., 2012. Towards a National Atlas - geo web service. In: *Proceedings of ACIS 2012- The First Asian Conference on Information Systems (6-8 Dec 2012)*, Siem Reap, Cambodia, p. 9.

D3 website, 2017. http://d3js.org.

Giuliani, G., Dubois, A. and Lacroix, P., 2013. Testing OGC Web Feature and Coverage Service performance: Towards efficient delivery of geospatial data. *Journal of Spatial Information Science* (7), pp. 1–23.

Harrie, L., Mustière, S. and Stigmar, H., 2011. Cartographic quality issues for view services in geoportals. *Cartographica: The International Journal for Geographic Information and Geovisualization* 46(2), pp. 92–100.

Köbben, B., 2013. Towards a National Atlas of the Netherlands as part of the National Spatial Data Infrastructure. *The Cartographic Journal* 50(3), pp. 225–231.

OGC, 2010. OpenGIS Web Feature Service 2.0 Interface Standard. Technical Report 09-025r1, Open Geospatial Consortium.

Toomanian, A., Harrie, L., Mansourian, A. and Pilesjö, P., 2013. Automatic integration of spatial data in viewing services. *Journal of Spatial Information Science.* (6), pp. 43–58.

Website Dutch National Atlas / Nationale Atlas van Nederland, 2017. http://www.nationaleatlas.nl/.

Zumbulidze, M., 2010. The updating process of a national atlas in a GDI environment: The role of atlas editors. Msc thesis, ITC - University of Twente, Enschede.