FireWebService
Technical Report

Holger Fritze[1], Thore Fechner[1], and Frederik Loock[1]

Institute for Geoinformatics (ifgi), University of Mnster, Germany
{h.fritze|thore.fechner|f.loock}@uni-muenster.de

February 15, 2010

1 Introduction

Using GeonetCast data enabled this project to not only use almost real time data, but also provide environmental data of high relevance. This project, called FireWebService,aimed at visualizing actual and past wildfires captured by the Active Fire Monitoring (FIR)[1] being embedded in the METEOSAT[2] project. Processing this fire data gave us hands-on experiences in using and linking different technologies, deploying OGC conform web services and finally creating a small SDI.

2 Architecture and Data sources

2.1 data sources

In order to create a service, that provides enhanced information on the fire situation, several elementary different data sources have been brought in.

The core data is of course the Meteosat fire data from the GNC-Network. This data is provided as ascii files, giving information about the location of the fires (in terms of coordinates), a timestamp and the probability of correct fire identification. The before-mentioned quality value of the data has by definition two possible states: Possible means, that the data provider estimates a 60% chance of detecting a fire correctly, while Probable stands for an 80% probability of correct fire identification.

The first step to create a bigger benefit of this source was the development of a java application, accomplishing data preprocessing. This application parses all fires from the provided ascii file and creates fire objects, which are handed over to the WFS-T. The fire object are defined by an own featuretype definition.

[1] http://www.eumetsat.int/Home/Main/Access_to_Data/Meteosat_Meteorological_Products/Product_List/SP_1145431848902?l=en
[2] http://en.wikipedia.org/wiki/Meteosat

As base maps for the portal applicaiton, OpenStreetMap maps were included via WMS using OpenLayers API. To locate a fire in the sense of country and address, the GoogleMaps geocoder was used for reverse-geocoding to derive the country name from the coordinates of the fires. With the country name further information on the location ought to be requested. This project has tied in the Wikimedia API to request and parse Wikipedia articles accordant to the country.

By integrating and combining different data sources in a sophisticated way, the user gains a lot of additional benefit.

2.2 architecture

The architecture of the FireWebService consists of five major building blocks. Those basic parts are cronjobs, which are configured using crontab, a java application, a transactional WebFeatureService (Geoserver), a PostGis database, and a portal application — The architecture in Figure 1 gives an overview. The entire FireWebService runs on a Linux based machine, namely a Debian Lenny.

The provided "raw" data from the Geonetcast Network is tied in using the samba packages. It is stored on a windows machine which is accessible from outside the university network only via a VPN. To mount the data an university account is required. The command to mount the storage into the file system is:

Listing 1.1. Command for mounting the gnc-data in a Linux machine

```
mount -t smbfs //gnc-data.uni-muenster.de/Geonetcast /
    mnt/gnc -o username=u_nam00,workgroup=uni-muenster.de
```

The data served under *gnc-data.uni-muenster.de/Geonetcast* is mounted into the Linux file system */mnt/gnc*. It is needed to supply the command with a valid login data from the university. For further information on the command please check the manual pages. The crontab table defines (Listing 1.2) three jobs, which schedule three shell scripts. They are responsible for starting the java program, stopping and starting the Geoserver.

Listing 1.2. Crontab on GNC-VS01

```
# Shell variable for cron
SHELL=/bin/bash
# PATH variable for cron
PATH=/usr/local/bin:/usr/local/sbin:/sbin:/usr/sbin:/bin
    :/usr/bin:/usr/bin/X11
#
```

```
2,16,32,47 * * * * /usr/local/bin/start.sh >/dev/null 2>&1
29 * * * * /usr/local/bin/up.sh >/dev/null 2>&1
28 * * * * /usr/local/bin/down.sh >/dev/null 2>&1
#
```

The first job is started every 15 minutes each day and calls a shell script (Listing 1.3), which starts the java application and writes log files and sends them by email to a specified address. For this purposes of maintenance and administration an own email account was created.

Listing 1.3. "start.sh" shell script on GNC-VS01

```
#!/bin/sh
date=`date +DATE%d.%m.%yTIME%H:%M:%S`
/usr/lib/jvm/java-6-openjdk/bin/java -jar /usr/local/bin/fireparser.jar | tee /home/thore/logs/$date.log
cat /home/thore/logs/$date.log | uuenview -a -b -m geonetcast@gmail.com -s "$date" -f report@gnc-vs01.uni-muenster.de
```

The "up" and "down" scripts are constructed in a similar fashion and starts respectively stops the Geoserver every hour. This became necessary, because the used version of Geoserver (2.0.0a) has a bug, which allocates more heapspace after each FE insert request to the PostGis database. This resulted in a fatal crash of Geoserver. The restart of the service avoided such a crash. The developers of Geoserver were made aware of this bug and promised to fix it. The next version of Geoserver should fix this bug.

If the java application is executed, it creates – according to the current date and time – the latest filename of the ASCII file, which provides the preprocessed information on wildfires. For example if the current date would be 02/09/2010 03:15pm the application would subtract a specific offset (e.g. 4 hours), create the corresponding filename (e.g. 201002091115), search it in the mounted gnc-data folder and start parsing it. The parsing process is done on a character level using a regular expression pattern, which allows retrieving the necessary data (latitude, longitude, timestamps, quality).

Each fire event is send via an HTTP Post request to Geoserver using a Filter Encoding insert request. An example request is shown in Listing 1.4.

Listing 1.4. Filter Encoding Insert Request for the created FeatureTypeFire in the Layer postGISpossibleFire, Workspace geonetcast.fws

```
<wfs:Transaction service="WFS" version="1.0.0"
xmlns:wfs="http://www.opengis.net/wfs"
xmlns:geonetcast.fws="http://purl.org/fws"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wfs http://
    schemas.opengis.net/wfs/1.0.0/WFS-transaction.xsd">
<wfs:Insert>
<geonetcast.fws:postGISpossibleFire>
<geonetcast.fws:the_geom>
<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.
    xml#4326\>
<gml:coordinates decimal="." cs="," ts=" ">latitude,
    longitude</gml:coordinates>
</gml:Point>
</geonetcast.fws:the_geom>
<geonetcast.fws:id>id</geonetcast.fws:id>
<geonetcast.fws:dateillwis>date</geonetcast.fws:
    dateillwis>
<geonetcast.fws:quality>quality</geonetcast.fws:quality>
</geonetcast.fws:postGISpossibleFire>
</wfs:Insert>
</wfs:Transaction>
```

The HTTP Post request is send to Geoserver, which processes it, takes the information according to its settings and writes it into a PostGIS database. Although Geoserver offers various data back ends the PostGIS storage is one of the more complex options, but it is the most convenient one. The first FireWebService prototype had used a shapefile storage, which worked well for the first few days. But shapefiles are not designed to hold such the massive amount of data. Although it should be able to store around 1 billion data sets (DBASE IV specifications) the response times (reading and writing) grew linear with the amount of data stored in it. After a couple of days the response times became inacceptible long. Hence a PostGIS database was set up to avoid such issues and have almost indefinite storage capabilities (only restricted by hard disk capacities).

The portal application was realized by using an apache webserver, a proxy script that was necessary due to javascript security issues, OpenLayers, the Google Geocoder API and the Wikepedia API with JSON.

If a user open the map in the portal OpenLayers fetches a base map from OpenStreetMaps, ties in the fire overlays published by Geoserver as WFS with an "In between Filter" (if necessary through the proxy script) and renders the result. The In between filter is applied to the attribute *dateIlwis* and selects

the datasets between a start and end timestamp. In a click event onto a fire a reverse geocoding process using the Google API is started, which translates the coordinates back into a placemark containing information about the country and eventually, if applicable, an address. Using this information the Wikepdia API is used to retrieve additional informations on the the country or the place through the JavaScript Object Notation (JSON). The returned article is parsed, processed and rendered within a pop up along the retrieved attribute information of the selected fire.

This design allows to use the Google Geocoder API, because information is only retrieved when needed thus the restrictions of the Google API can be met (only 15000 geocoding request per day per person with a 5 second delay between each request).

The portal application is only one way to access the data. It is possible to access the data directly through different interfaces e.g. the PostGIS database, as KML through Geoserver or other filter encoding requests which are not using a time interval instead a spatial (BBOX,Disjoint,Touches,Within,...) operator. Filter Encoding supports not only comparison or spatial operators, it is possible to combined queries those using logical operators (for further informations check the OGC FilterEncoding Specifications).

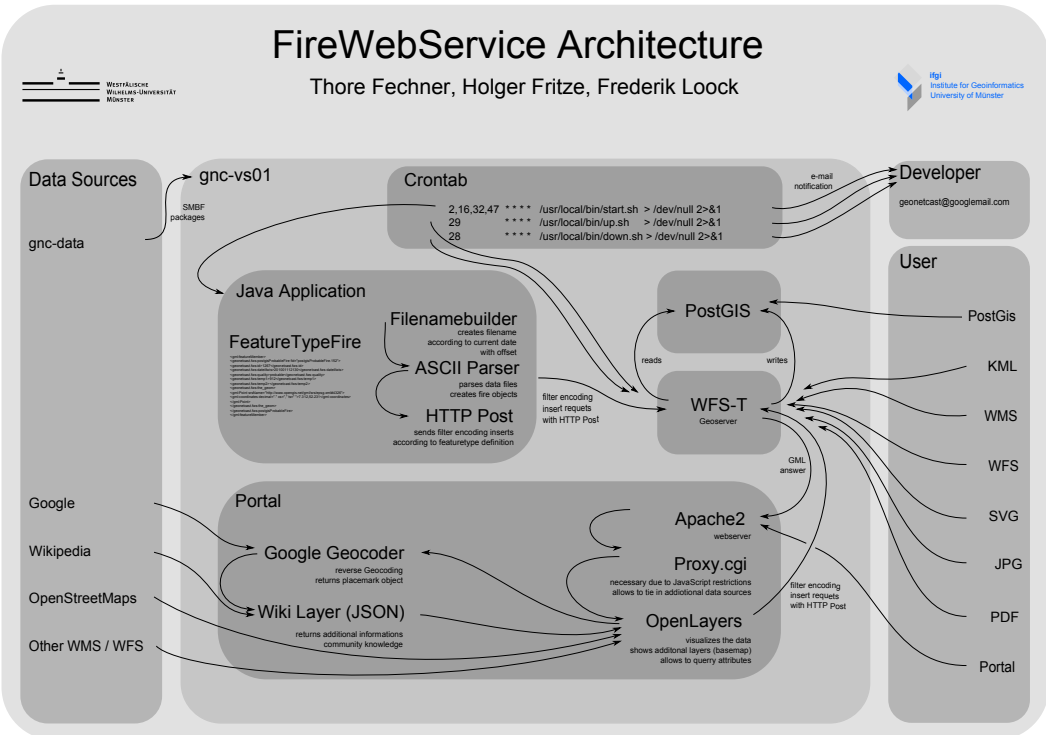

Fig. 1. Architecture of the FireWebService

3 Portal

The portal application is designed as an easy accessible homepage displaying the data on an interactive map. This front end connects all pieces described in the architecture and data sources. The homepage bases on the OpenLayers API[3], which is an open source JavaScript library for creating maps on client side. It is for example used by OpenStreetMap[4] and OpenRouteService[5]. The maps claims the Spatial Reference EPSG:4326, which is the World Geodetic System in its revision from 1984 (WGS 84). Basically the map consists of two parts: the base maps, from which at least one has to be displayed at any time and the overlays, which can be chosen individually. The OpenStreetMap layers Mapnik and Osmarender are integrated as base maps with the following command.

Listing 1.5. OpenLayers Command of integrating a WMS Base Layer

```
oSMmapnik = new OpenLayers.Layer.OSM.Mapnik("OSM Mapnik"
    );
oSMOsmarender = new OpenLayers.Layer.OSM.Osmarender("OSM
    Osmarender");
map.addLayers([oSMOsmarender, oSMmapnik]);
```

The current fire datasets are displayed as two separate vector overlays. The possible fires having a certainty of 60 % are displayed as yellow dots and the probable fires having a certainty of 80 % are displayed as red dots. Both layers are integrated as WFS overlays with Filter Encoding. The “In between Filter” is used to visualize only the fires of a timestamp defined by the user. Normally the last available timestamp is predefined. It is also possible to display fires during a time interval by specifying a start and end date. Due to memory reasons a maxFeature attribute is set to 1,000.

Listing 1.6. OpenLayers Command to integrate a WFS with FilterEncoding as vector overlay

```
probableFire = new OpenLayers.Layer.Vector("probable
    Fire", {
     strategies: [new OpenLayers.Strategy.BBOX()],
     protocol: new OpenLayers.Protocol.WFS({
          url: "http://gnc-vs01.uni-muenster.de:8081/
              geoserver/wfs",
```

[3] http://openlayers.org/
[4] http://www.openstreetmap.org/
[5] http://www.openrouteservice.org/

```
        featureType: "postgisProbableFire",
        featureNS: "http://purl.org/fws",
        extractAttribute:true,
        maxFeatures: 1000
    }),
    filter:new OpenLayers.Filter({
        type:OpenLayers.Filter.Comparison.BETWEEN,
        property: 'dateIlwis',
        lowerBoundary: startDate,
        upperBoundary: endDate
    }),
    styleMap: styleProbableFire
});
```

Figure 2 shows the map displaying all fires from February 2, 2010 between 11:30 and 14:00 o'clock. With a click on a fire a pop-up window opens (Figure 3) showing the attributes of our self-defined FeatureType definition. As already described in the architecture the coordinate pair is used by geocoding modul get the country name, which is displayed in the pop up window. The country code in turn is handed over to the wikipedia modul, which returns and integrates the nation article from the english wikipedia into the pop up window as well.

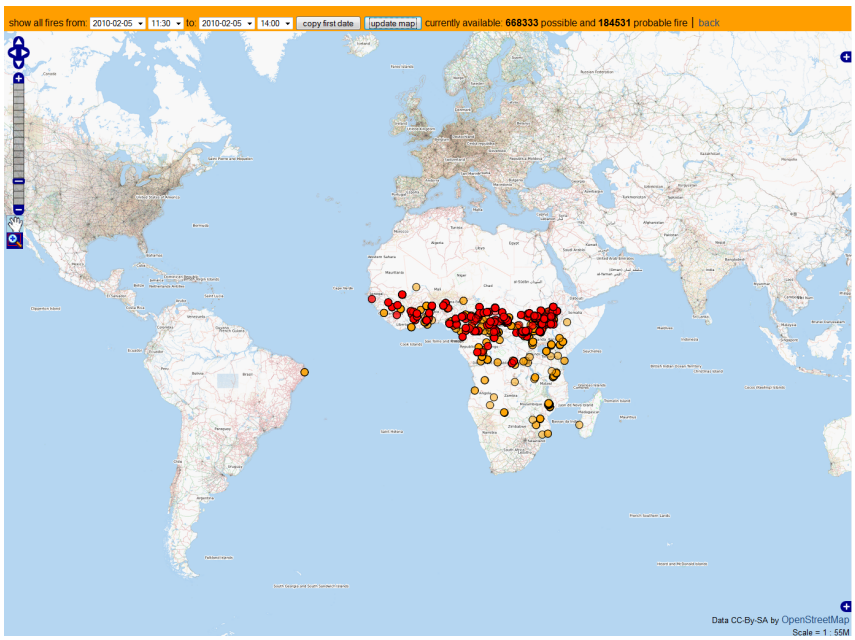

Fig. 2. Screenshot of the final map

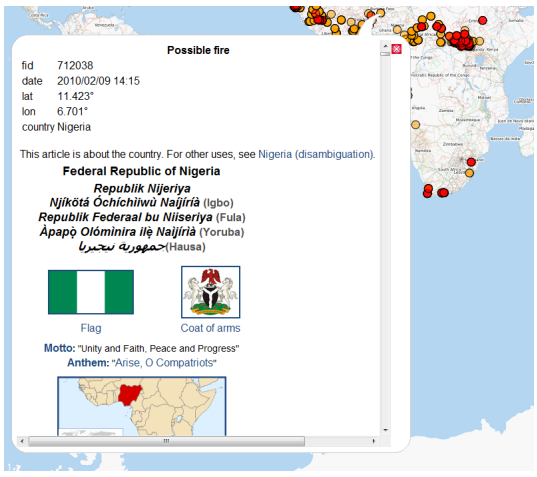

Fig. 3. Screenshot of the pop up window

When integrating a WFS in OpenLayers the data is retrieved via an XMLHttpRequest. But this functionality is restricted in JavaScript due to security issues. To retrieve data from a remote server it was necessary to set up a proxy script web accessible on the server hosting the portal application and add the address of the Geoserver to the list of allowed Hosts.

The address of the portal applciation is `http://gnc-vs01.uni-muenster.de:1337/firewebservice`

4 Evaluation

In technical terms we can resume that we have accomplished most of our tasks and achieved the goals we defined at the beginning of the project. Table 4 shows the sub-goals and the methods and technologies for realizing them.

The only major change we made is the replacement of MODIS with METEOSAT data, albeit this happened at an early stage. Everything else worked out as it was planned, which is last but not least a result of rather intensive planning and estimation of the project's feasibility. Nevertheless this project had minor or major obstacles which had to be solved. One of the biggest obstacles was the Geoserver and the usage of shapefiles. Due to the collapse aroused by too many entries we decided to set up a PostGIS database, which in turn however caused Geoserver heapspace problems. This could be solved by shutdown- and startup scripts for Geoserver. Another obstacle was posed by Google's geocoding request limitations. This could be solved by adapting the architecture to an on-the-fly request when the user selects an fire object on the map. The third big obstacle was to set up a proxy script, which became necessary due to Javascript XMLHTTP Post security issues. But all this difficulties have been overcome and solution and/or workarounds were found.

Table 1. Aims of the project

Goal	Approach
using live GeoNetCast data	java application smbf packages cronjobs
setting up an own SDI	Geoserver *Postgis*
map spatial distribution of wildfires	OpenLayers create featuretype definition OpenStreetMap
use additional resources	link Wikipedia articles (dynamically) Google geocoder
Homepage	PHP, JavaScript, HTML, XML HTP POST, *JSON*, GML
allow multiple ways to access the data	wms, wfs, *jpg, svg, postgis, pdf*
create logging	*uuenview*

In general everybody of us has experienced that working with a new software systems is always linked to different technologies, programming concepts and ways of thinking. Acquainting yourself with each new system is time consuming and cumbersome. But ones they are up and running powerful interactions are possible. Further, without these interaction such architecture designs have not been possible.