FOSS4G Certification Issues in the Development of a Large Telecommunication Application

Jorge Gustavo Rocha[1], Rui Alves[2], Paulo Machado[3]

[1]Universidade do Minho, jgr@di.uminho.pt, Portugal
[2]XLM Lda, ralves@xlm.pt, Portugal
[3]PT Inovação, SA, paulo.machado@ptinovacao.pt, Portugal

Abstract

In this paper we describe and discuss several issues related with the development of a new FOSS4G (Free Open Source Software for Geospatial) based application for a large telecommunication company (abbreviated by telco).

We describe the overall development process. At different stages, even in early ones, demonstration and quality assessment of the system was required. The level of maturity and completeness of the FOSS tools used was suitable enough, but these were always questionable by the contractor, able to read hundred of open issues and open bugs related to FOSS tools (while COTS tools hide such vulnerabilities from the public).

Specifically, in this article we describe the effort made to: import the data from the former legacy GIS; support the required editing capabilities on the web; and to support complex features (i.é. topology) and their necessary visualizations tools. Schematics (which uses just edges and nodes) was absolutely necessary to address several network operations.

We conclude by pointing out how difficult was to (even informally) certify the FOSS4G based GIS. More attention must go towards formal methods, when developing large scale FOSS4G based solutions, is our final remark.

1. Introduction

This paper discusses some issues related with the development of a new FOSS4G based application for a large telecommunications company in Portugal. Their network covers all country and it a huge number of features. This telco, as others, requires very tight inventory control, maximization of installed equipment usage and high quality of uninterrupted service to the final customers or to other telco renting their equipments.

Until now, their operations were supported by a legacy application, based on Siemens SICAD.

In 2006 we prepare a demo showing that it would be possible to create such a large GIS in a modular fashion, integrating interoperable FOSS4G components, and also making the application interoperable with some existing legacy tools. After their acknowledgement, a consortium was created to develop such application.

ISBN 978-0-620-42117-1

1.1 The Consortium

The project was coordinated by a company of the same telecommunication group, PT Inovação SA, focused on the development of innovative value-added services, products and applications contributing for the leveraging the group companies' competitiveness and business. The consortium included another small but highly skilled IT company, XLM Lda, and the University of Minho.

1.2 The Problem

Accordingly to (Ramsey, 2007), "The Open Source GIS space includes products to fill every level of the OpenGIS spatial data infrastructure stack". Therefore, it seems plausible to develop a new GIS application simply by composition. As we will show, it is not (yet) easy as we might think.

1.3 Development Phases

After the demo was approved by the telco, a prototype was built and installed to be fully evaluated in the production environment, and data from three different areas was imported. When the prototype was stable enough with all required functionalities, the final application was developed. This application is now running in parallel with the former application.

2. GIS Modeling

We started with the great advantage of modeling a new system from scratch.

As in almost application, gathering the necessary requisites is a time consuming and iterative process. We had two major sources of information: the former legacy system and the users of it. There were formal scheduled meetings with the contractor, useful to validate the work under progress and to fine tune the application.

A lot of reengineering was necessary to model the telco GIS. The previous GIS had data in one Oracle RDBMS and in the SICAD file format (*.sqd). Data in both representations were studied in detail, to fully understand the model already in use. But after more than 10 years of usage of the legacy system, almost no updated documentation exists. Several small applications were developed in the meanwhile (even a web viewer), providing more and more functionalities, but no documentation at all exists.

Also new requirements were integrated, requested by the contractor. The major new requirements were most related with interaction, hiding the complexity of certain tasks. Only a few imposed changes to the data model. For example, o more detailed inventory of infra structures rented to other telcos was a new requirement.

2.1 The Three Levels of Modeling

Modeling is an abstraction process, where we select just the essential elements of the real world. We started with the more abstract one, the conceptual model, for reasoning and to discuss it with the contactor, and only afterwards we derived the physical models accordingly to the RDBMS used. In fact, several iterations were necessary, but we always started by reviewing the conceptual model and generated another physical one. Conceptual modeling has always been one of the cornerstones

for information systems engineering as it describes the general knowledge of the system in the so-called conceptual schema. That is why we worked a lot around this conceptual model, which is independent of the RDBMS.

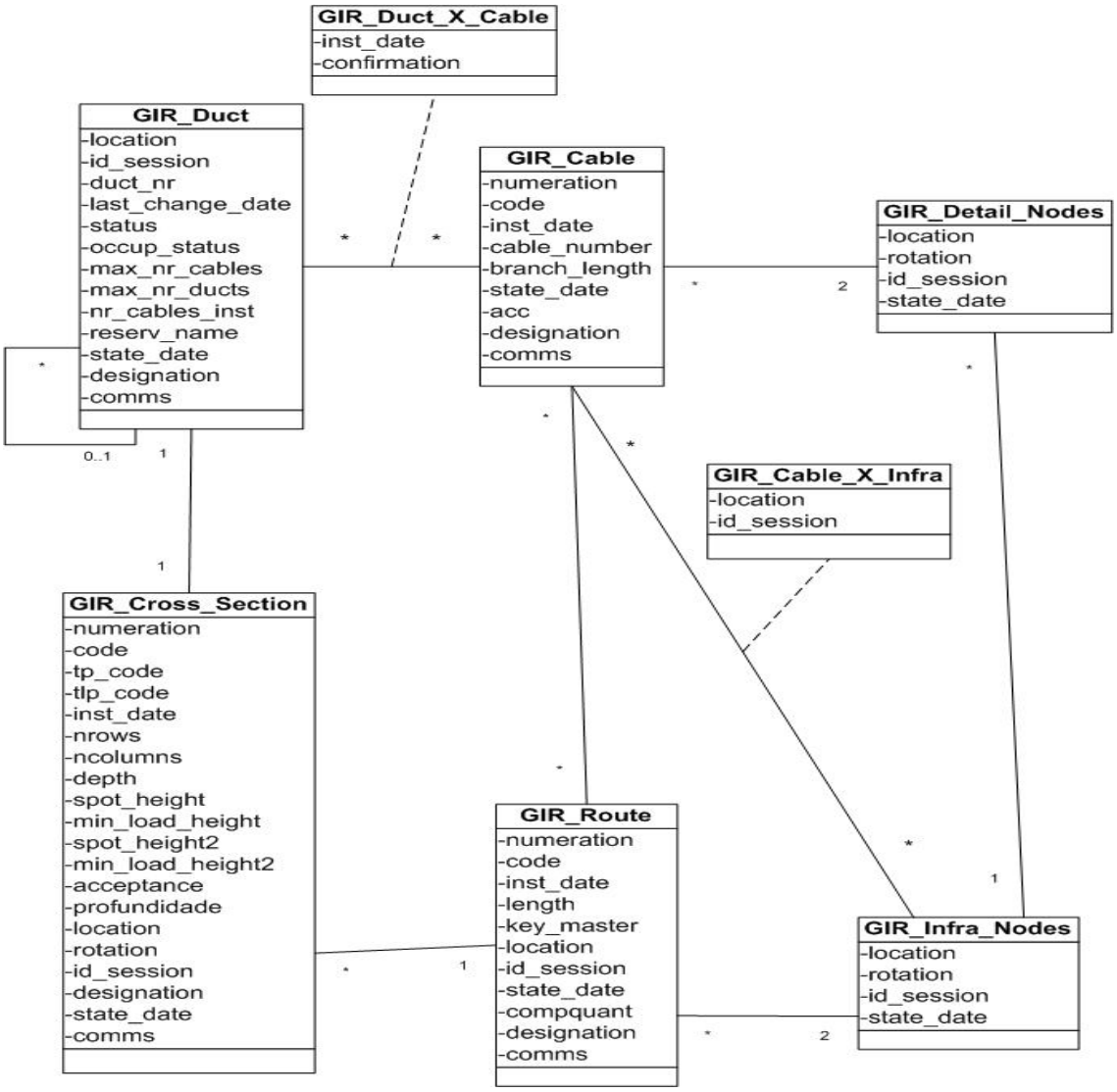

Figure 1. Conceptual model for a telecommunications company

Just to get an overview of the conceptual model, there are six major entities, as depicted in the UML notation in Figure 1. GIR_Route stands for the directional path of a sequence of cables, aerial or through the ground. GIR_Infra_Nodes and GIR_Detail_Nodes stands for equipments, structures or equipments inside structures, along the cable's route. GIR_Cross_Section is the transversal cut of a route, in which one or more GIR_Duct can exist. Inside each GIR_Duct, mostly in PVC, there are one or more GIR_Cable, protecting them.

2.2 From the Abstract Model to the Physical Model

The full conceptual model was transformed into a logical one, accordingly to well known rules. This logical model is still independent of the specific RDBMS used. The physical model and the corresponding DDL specification were generated afterwards, according to the specific characteristics of the RDBMS used.

For the transformation of these models, a trial version of the Rational Software Architect was used to assist the process (IBM RSA, v7.0).

Two different physical models were generated, for two different RDBMS SFS compliant: Postgresql/PostGIS and Oracle 10g XE. The second one, although a legacy one, is free of charge, with the known limitations. This approach allowed us to keep both databases to fully evaluate both.

2.2.1 RDBMS Selection

We had an initial objective to compare the support for topology both in PostGIS and in Oracle Locator (Spatial is not available in the XE edition) in the prototype. Unfortunately, we were unable to develop a proper comparative in the short term. Without sound and sustainable arguments, the Oracle XE was selected by the contactor (the cost, in this comparative, was not relevant). So, the prototype was developed on top of an Oracle XE database (the prototype data was less than the 4Gb limit of XE).

2.3 Discussion

Instead of focusing on maps, we focused on models. Models better represent the real world phenomena. Fortunately, in our GI Science and Systems field, we are moving from maps to models. But models are more difficult to share and to develop in the OSS fashion. For example, ESRI developed and share data models for a variety of domains “to simplify the process of implementing projects, and to promote and support standards that exist in our user communities” (ESRI Data Models, 2008). So, our first thought is that FOSS4G is still more focused on technologies rather in solutions for specific domains.

Our second thought goes to the advantage of a well known database producer like Oracle against one very mature and standards complaint Postgresql/PostGIS. Although we knew that was a decision just for the prototype (while the 4Gb limitation of Oracle XE was not a problem), many code was written in the meanwhile in PL/SQL that would make difficult to maintain Postgresql/PostGIS as an alternative solution for the final application.

3. Import Data from Former Legacy Siemens SICAD Based Application

Importing SICAD data required a lot of tenacity. Initially, it was not easy to understand the poorly documented SQD file format, and the meaning of every bit on it. It was a laborious task, only possible, with the help of some telco collaborators who did know the former application details. It was necessary to have the application running to see which data was read and written to understanding how things were stored. Also, the data was spread over the SQD data files and one Oracle database. But the real difficulty was created by inconsistencies of the SQD data. Data integrity was enforced with data edits, but this method did not cope with the evolution and complexity of applications.

The new data model, as usual nowadays, incorporates all possible rules to ensure that data is of high quality, correct, consistent and accessible. So, in fact, what would be a normal import became an extensive set of processing steps to derive a consistent data set, from an inconsistent one.

Fortunately, the spatial data was much more consistent, and was used in fact to detect and, where possible, correct inconsistencies. From the users' point of view, the depicted data looked correct, but the relations and attributes behind were not so consistent.

3.1 Importing in Three Steps

The import process should be flexible enough to allow constant tuning of the validation rules. It was divided in three steps.

The first is the SQD parsing, and it is supported by one perl script. Basically, it interprets the SQD file, and for each recognized feature (cable, manhole, etc), a new xml element is generated, with all attributes beneath it. This first step transforms all data represented in a SQD file in one XML file. The rest of the process gets independent of the SQD file format.

The second step gets the previous XML file and generates a SQLLoader .dat file for each kind of element. It is also written in perl, and for each element, all related data is extracted and written in the corresponding .dat file. All the .dat files are loaded in a temporary Oracle database (an additional corresponding .clt is also necessary). Each .dat fills its corresponding table.

The third step runs on the database. This includes the majority of the validation rules, divided in five packages. Each package guaranties the consistent import of one or more type of element, according to the previous elements already validated.

Additional logging tables are generated for each element failed to import.

3.2 Discussion

Data integrity cannot be guaranteed by the editing operations. Data integrity should be constrained at the database level. So, as this case study shows, what would be a normal import process became the major validation effort of all former data.

4. The Middle Tier

Between the presentation tier (the web interface) and the data tier (Oracle), the middle tier was assembled using well known FOSS4G tools: UMN Mapserver, the WMS used to render all raster maps; Geoserver, used as the WFS and Transactional WFS; and MetaCarta's TileCache, the corner stone component for caching WMS requests. All of these tools are well known in the FOSS4G community, and are well documented.

4.1 Discussion

The usage of these tools was straightforward. Small (but many) adjustments were necessary to comply with some Geoserver requisites. For example, the data model relation between class and subclass should be implemented by two different tables with no redundancy. But Geoserver was unable to join the attributes of the subclass and its parent, so all attributes of the class were replicated in all subclasses, thus introducing (controlled) redundant information.

Many additional tables were necessary just to store and manipulate visual data. For example, to specific position of a crosscut section can be changed by the user, to improve the visualization. So,

besides the information about the crosscut, we need additional information for render it. The additional information (not related with the business, but solely with rendering and layout, is stored in additional tables (with an additional _VIS suffix). Also, some limitations of dynamic SLD were circumvented using additional attributes on the database.

5. Web Interface

From the beginning, we promise a web interface for the Telco FOSS4G based application. Not every interaction would be supported through the web in the short term. For example, advanced editing and management of the base cartography should be made with the existing legacy tools, specifically designed to manage cartography.

The new web interface is based on Open Layers.

The GIS web interface includes all functionalities required to support the everyday operations. These operations were identified, analyzed, and implemented. Some were just visualization and query operations that almost ran ‘out of the box’. Other editing operations demanded a large development effort. For example, editing all connections between cables and junctions must be ‘semantically’ supported. We need to know how many and which cables came in and out. For each cable, we also need to know how many available pairs exists, which were connected, disconnected or connect, but with no continuity (dead pairs). This example and many others require that the presentation layer knows about telco operations. So, Open Layers basic editing capabilities must be adequate to the telco business.

5.1 Schematics

Telco users are trained to reasoning on top of network schematics. Schematics are a well known visual representation of telco (and others) networks. Support for schematics generation and visualization was a major requisite, and it is divided in: network representation in the relational database, and visualization and manipulation on the web.

Schematics preserve the topological properties of the cable network. To represent this complex feature (the topology) in a relational database, we only need support for nodes and edges. Ultimately, this can be done with just one table, where each record describes the connection (edge) between cables or other equipment (nodes). In fact, the connections in the telco business are among cable pairs, or set of pairs. So, the above model gets a little bit more complicated, because we need to identify the cable pairs, and this identification can be local or relative to their source (the numbering can came from the central, for example). To retrieve the schematic starting at some specific point in the network, a recursive procedure must be written if the database engine does not support recursive queries. IBM DB2 does support recursive queries as Oracle (although Oracle uses a non-standard CONNECT BY clause). Postgresql does not support (yet) recursive queries, but the pgRouting project provide the functionality required for this. Because we are using Oracle, a simple recursive query is used to retrieve aany schematics, giving a source.

To layout the schematics, a simple JAVA application is used, which uses the GraphViz API to layout the initial representation. The representation is rendered in SVG and its delivered to the

client. This is done almost instantly. In the former legacy GIS, the schematics were generated in a batch base, where each job could run for several hours or days.

The SVG delivered to the client has additional Javascript included to allow the manipulation of the schematic's layout. Whenever one element (node) is moved, the Javascript routine calculates the dependent connections (edges) and changes them accordingly.

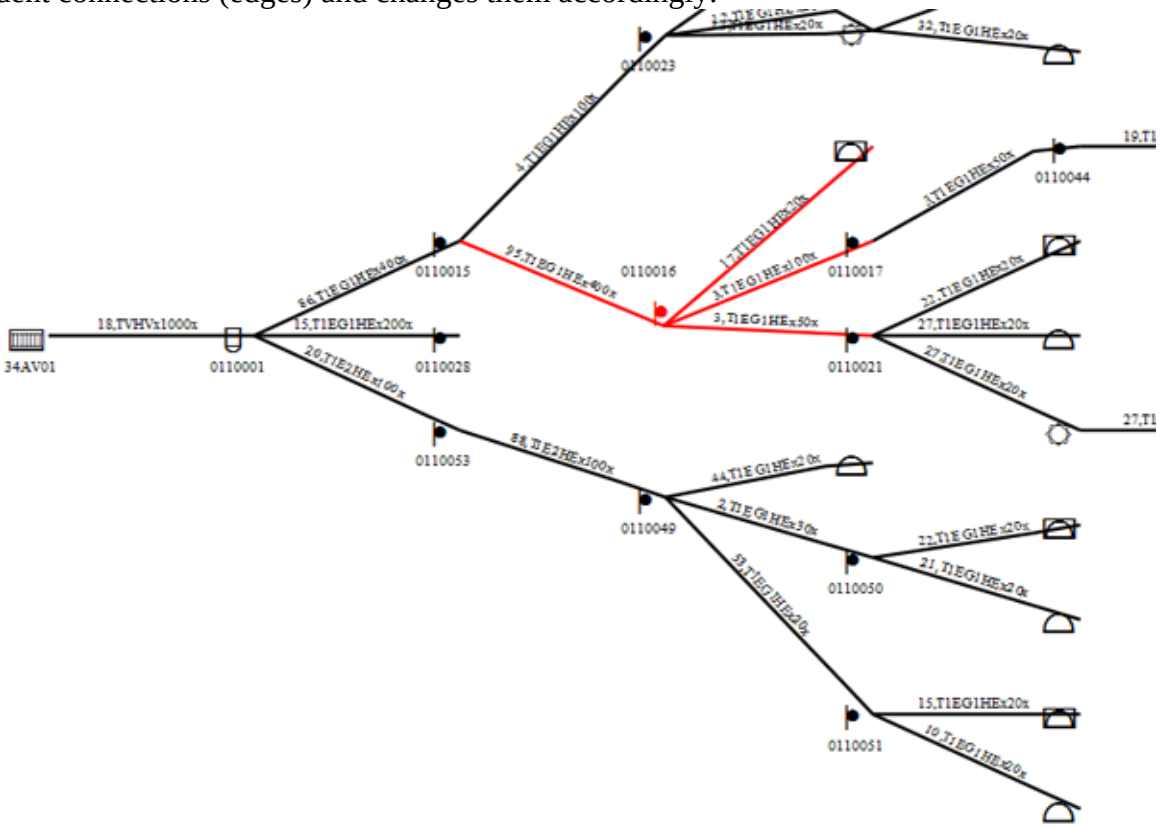

Figure 2. SVG schematics after a small change in one node's position (highlighted in red the incoming and outcome connections)

The user has the option to save the schematic or just to close it and request another one. If the schematic is saved (it is stored on the server side), for each moved node, the deltas in both axes are calculated and stored in the database. Whenever the same schematic is requested, the user has the option to apply automatically all previous manual adjustments.

5.2 Discussion

The presentation layer was the most time consuming tier. Many lines of Javascript code were required and many tests were necessary to have robust editing capabilities. But the major problem is that changes to the other tiers were frequently requested by the programmers of the presentation layer. Some were minor changes, but others required changes in the configuration of Geoserver, in the SLDs, in the data model (new tables just to store layout positions), etc. Also, the pressure to have things up and running, forced the programmers to hack the OpenLayers framework as possible and not as it should be done. The generic editing capabilities of OpenLayers were mixed with the required specific operations of the telco company. When Open Layers 2.6 came out, it was very difficult to isolate the modifications made on top of the previous version.

Regarding the schematic support, the GraphViz open source graph visualization software helped a lot. The output representation in SVG is straightforwardly managed with Javascript in the client side, and because there are no sophisticated graphical primitives in the generated SVG, it is rendered very well in Firefox. The Inkscape OSS tool, although not used, inspired us how to generate the SVG with additional information for better Javascript DOM manipulation.

6. Conclusions

The application built is currently running in the telco company, in three different cities. It is still running in parallel with the previous legacy GIS application.

From our experience, we can highlight some thoughts, besides those already included in each chapter, under the discussion section.

First of all, during the development phase, under the pressure of the contractor, it is very hard to share the code developed and return it to the community. So, only afterwards, we can identify the parts that might be interesting to the community.

Secondly, we found very hard to prove the quality of the built application. Informally, every tier was fully tested, extended logs were generated and analyzed. But some additional effort must be made to certify our FOSS4G based solutions.

References

Ramsey 2007, The State of Open Source GIS, FOSS4G 2007, http://www.foss4g2007.org/presentations/viewattachment.php?attachment_id=8

IBM RSA, v7.0, http://www-01.ibm.com/software/awdtools/architect/swarchitect/index.html

ESRI Data Models, viewed 30 June 2008, http://www.esri.com/datamodels