2008:025 HIP

BACHELOR'S THESIS

GIS: using open source or commercial products

Jakob Normark

Luleå University of Technology

BSc Programmes in Engineering
Computer Engineering
Department of Skellefteå Campus
Division of Mobile Networking and Computing

2008:025 HIP - ISSN: 1404-5494 - ISRN: LTU-HIP-EX--08/025--SE

Jakob Normark
2008-08-27
v0.40

GIS – Using open source or commercial products

Jakob Normark

Preface

The work presented in this report has been carried out at Explizit AB, Skellefteå. The supervisor from Explizit was Torbjörn Lundmark, and the examinator from LTU was Robert Brännström.

This thesis project was part of a project at Explizit AB, with me and the following project group members:
Torbjörn Lundmark
Robert Brännström
Fredrik Bergqvist
Fredrik Krekula
Fredrik Lundgren
Ulf Jonsson
Östen Lundahl
Mikael Holmstrand
Robert Persson

I would like to thank everyone in the project group for supporting me during my thesis project, especially Torbjörn Lundmark from Explizit, and Robert Brännström from LTU.

Skellefteå 2008-08-27

_____
Jakob Normark

Abstract

When health care is moved from the hospital into patients homes it is often desirable to be able to map information geographically. GIS is a system used to accomplish this, and there are both proprietary and open source GIS softwares available.

Explizit AB is a company in Skellefteå that has a product called CheckUp, which basically is a suitcase with different medical instruments connected to it.

Explizit AB wants to see if it is possible to use a open source GIS solution to build a system that for example can be integrated with CheckUp.

This report contains a investigation on the different open source licenses, what difference the licenses has and how they affect a commercial product, and describes the solution I built using only open source software.

Jakob Normark

Table of Contents

1 - Introduction

Explizit is a company based in Skellefteå. Explizit is a IT business that builds software solutions and outsources IT consultants. Their sphere of activities include everything from developing systems for Svensk Bilprovning, Boliden and Ericsson to Skatteverket. Explizit is owned by Argentum AB and is a part of the Argentum group.

One of Explizits products is a system with wireless data capturing, which can collect and store data from a number of wireless devices. This system is called CheckUp, and the main purpose with CheckUp is to move different medical tests out of the hospital into the patients home. Using bluetooth technology the medical instruments can communicate wireless and transfer the data to the hospital, thus eliminating the need of the patients having to travel to the hospital every time a routine check up is needed.

2 - Description

2.1 - Overview

In wireless data capturing it is often desirable to map information to a given geographical location. One way to accomplish this is by using GIS – Geospatial Information System.

GIS is a system for managing data and referencing this geographically, and therefore GIS can be used for a variety of things ranging from urban planning to scientific investigations. Since mapping data geographically and presenting this graphically is what is desired, GIS suits that need perfectly.

When working with GIS, there is both commercial and open source solutions, and among the commercial both Mapinfo and Carmenta is interesting alternatives. In the open source area, GRASS GIS is a strong candidate, and it is available for both GNU/Linux, Mac OS X and Microsoft Windows.

Explizit wants to see if a open source solution can be used to fullfill the companies demands regarding GIS, and investigate what impact the many open source licenses(GPL, LGPL etc,) makes in a commercial product.

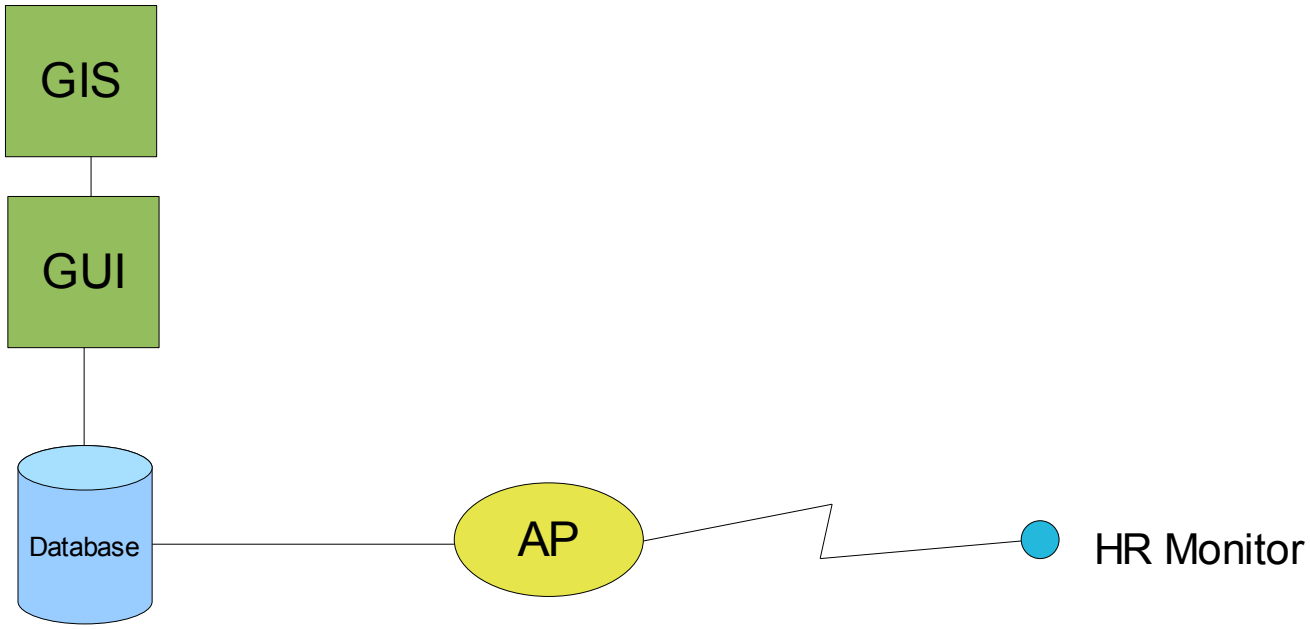
For example, if a pulse watch and a GPS is used when going for a walk, one would later want to see pulse and location at a given time, presented graphically on a map, and for this GIS can be used.

Should a open source GIS or commercial GIS be used?

How does open source licenses affect a commercial product?

2.2 – The concrete problem

A software prototype for mapping data geographically built on a open source GIS is needed.



3 – Purpose

I did my thesis project at Explizit for 10 weeks, which is 15HP, and completed the following tasks.

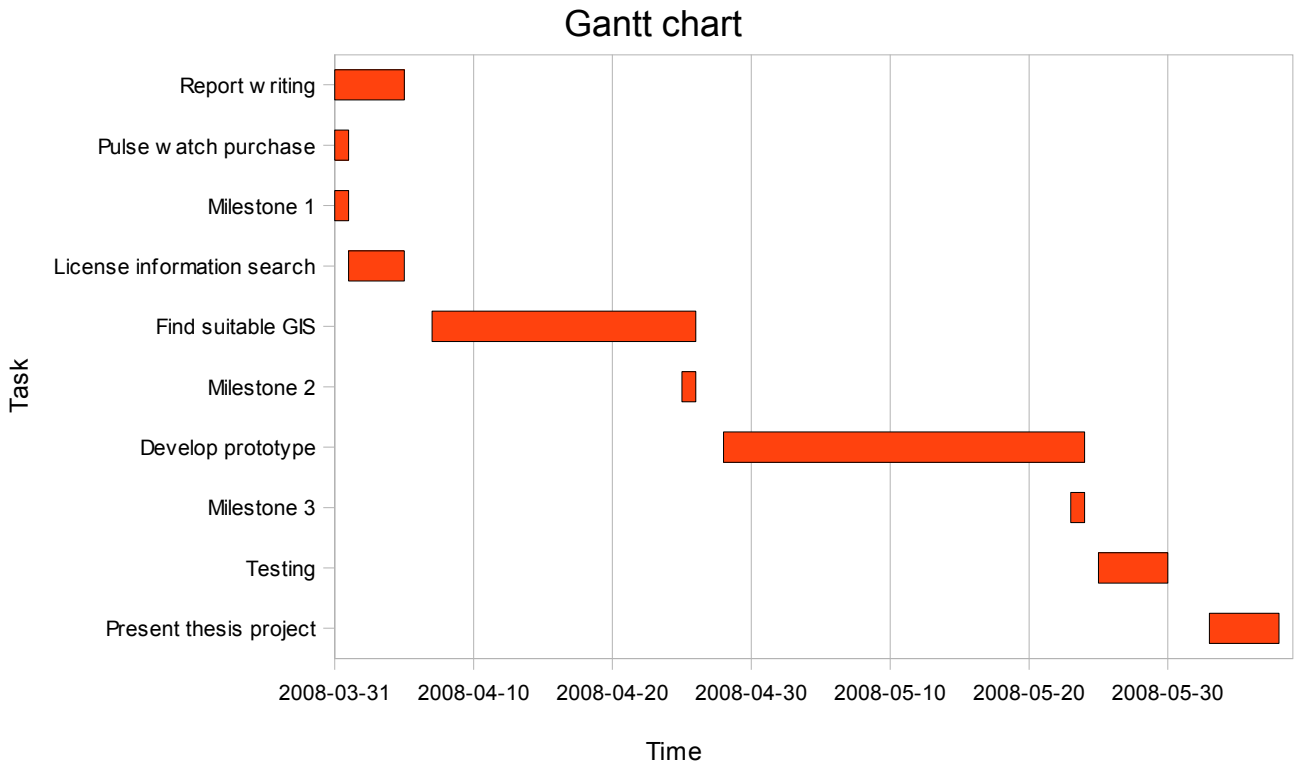
- Find a heart rate monitor with GPS and bluetooth
- Investigate GIS solutions to see what GIS suits Explizits demands best.
- Look into the different open source licenses, how they affect a commercial product.
- Develop a prototype using a open source GIS solution
- Write a report on the thesis project

4 - Time plan

4.1 - The plan

The project is planned to take 10 weeks.

- Week no. 1 – Decide which pulse watch/GPS to use
- Week no. 1 – Gather information about the different licenses used in open GIS software and put together a report of this
- Week no. 2-4 – Get a understanding of what differs in the open source GIS versions and also in the commercial GIS products, and find a suitable open GIS for the prototype
- Week no. 5-8 – Develop a prototype that can gather data from the pulse watch/GPS, and present the gathered data geographically on a map
- Week no. 9 – Prototype testing
- Week no. 1-10 – Write the final report of the thesis project . This will be an ongoing task through the whole project.



4.2 – The risks

Time allocated for a task might not be enough!

To prevent this, follow the time plan as long as possible.If this happens, see if the time plan can be altered without risking the outcome of the project, if not maybe the projects goal has to be reviewed or more people has to be brought in on the project.

Hardware might fail!

Be sure to have some redundancy in the hardware used, and never rely on one single piece of hardware In case of hardware failure, talk to appropriate person at Explizit about replacement hardware.

The project might be impossible to complete!

Set realistic goals to prevent this.

If the project is impossible the complete, talk to Robert Brännström at LTU and Torbjörn Lundmark at Explizit for advice.

5 - Project milestones

Milestone 1 – 2008-03-31

Which pulse watch with GPS and Bluetooth to use should be decided and the watch ordered.

Milestone 2 – 2008-04-25

Which GIS to use is decided.

Milestone 3 – 2008-05-23

The prototype should have all desired functionality and the final testing phase should start.

6 – Output

The output from my thesis project will be a prototype that can take data from a pulse watch with GPS, and using GIS present the gathered information to the user. Also, a report and a presentation of the project will be produced.

7 – Pulse watch

7.1 – Overview

The first task I was assigned to was finding a pulse watch with integrated GPS, which the other two who did their thesis project at Explizit also was assigned.

The pulse watch was supposed to have the following features:

- Monitor the heart rate of the person wearing it
- Track position by GPS
- Have a Bluetooth connection

7.2 – Execution

We started looking into the different pulse watches available the first day, and quickly realized that there was not that many watches to choose between. In the end, the FRWD B600 was the best option, seeing that it had both GPS, heart rate monitoring, altimeter, a thermometer and Bluetooth connection.

The FRWD B600

The FRWD B600 was ordered on the first day and arrived four days later.

8 – Open source investigation

8.1 – Background

I am in my thesis project going to develop a GIS solution for the company Explizit, and also look into the different open source licenses and what difference they make in a commercial product.

GIS is a system which takes different types of data and is able to map this to given geographic locations.

One part in developing the solution for Explizit is choosing an open source GIS, and by using open source software take into account what the license of the chosen GIS means both for the project and for a product that will be commercially available. The two most widely used open licenses, both regarding GIS and open source software in common, is the GNU General Public License and GNU Lesser General Public License.

8.2 – Brief explanation

GNU GPL and GNU LGPL is both open software licenses, maintained by the Free Software Foundation(FSF). Basically, they are both licenses with the main purpose of protecting everybodys right to read, freely modify and distribute the software which is GPL/LGPL licensed.

When the word ”free” is used when discussing the GNU licenses, one should think of free as in ”free speech” and not ”free beer”.

8.3 – The licenses

This is a quick summarization of a few of the open licenses.

8.3.1 – Copyleft

Copyleft means making a program free, and ensuring that all modified and/or extended versions of the program stays free, and that copylefted code never may be relicensed to a proprietary license, thus making it closed source.

8.3.2 – GNU General Public License

The GNU General Public License(GPL) is the most used open license. It is a copyleft license designed to ensure that software licensed under GPL will stay free to anyone. Free then means the freedom to use the software for any purpose, to share the software, to change the software and to share the changed software. Despite changes and sharing, the program will stay GPL, and this is what differs GPL from LGPL.

If, for example, a programmer writes a library and releases this under the GPL license, all programs that use the given library must also be free software, thus ensuring that all usage of the library will be done by open source software.

8.3.3 – GNU Lesser General Public License

The GNU Lesser General Public License(LGPL) is also a open license that is maintained by the Free Software Foundation. The main difference between LGPL and GPL is that the LGPL license is not a copyleft license. This means that any software/libraries written and licensed under LGPL is permitted for usage in proprietary programs.

8.3.4 – BSD Licenses

The Berkeley Software Distribution(BSD) license is a license that is considered very permissive, because it is a non-copyleft license with lesser restrictions than for example GNU GPL. BSD licensing permits using the BSD licensed software in commercial proprietary software. There is two major versions of the BSD license, the original BSD license and the modified BSD license, where the modified version is the most widely used. The biggest difference between these two is that the original license required a statement when marketing products that include BSD licensed software. One statement had to be included for every license with a different name, and this led to big marketing issues. The statement requirement is not included in the modified BSD license.

8.3.5 – Apache license

The Apache license is like BSD a non-copyleft license. The license says that all Apache licensed software has to keep notices from earlier works intact, but does not forbid using the free sofware in proprietary software or relicensing the free software. The most important thing is keeping intact notices saying that Apache licensed software has been used.

8.3.6 – Common Public License

Common Public License(CPL) is a copyleft free software license published by IBM. CPL is a license with the main goal of encouraging open source development, and it tries to accomplish that by restricting use of free software in proprietary software. CPL does have one exception for use in proprietary software, and that is, just like LGPL, not requiring the code to be free when using CPL licensed libraries.

8.3.7 – Creative Commons licenses

Unlike the other licenses discussed here, Creative Commons licenses is mainly supposed to be a license covering information(art, images, music etc), instead of software and source code. Originally there was four different Creative Commons licenses, where only one restricts using the material in commercial software, and that is the NonCommercial license. Most of the time, the single most important thing in the licenses is to give the creator of the used material credit for his/her work.

8.4 – What happens when free software is sold?

Let us imagine that a company that sells software decides to include free software in their code. Depending on what license the free software is licensed under, the software that is going to be sold will be affected differently.

If the license is GPL, the company also has to release their own code in GPL – or a GPL compatible license.

If the free software is licensed under LGPL or CPL the company does not have to release their own code, just make sure that the LGPL/CPL-code can be obtained by customers. The same rules apply to BSD licensed code as to LGPL licensed code, with the exception that if the original BSD license is used, the marketing rules has to be applied to the companies product.

The Apache license is as permittive as the modified BSD license, where the only restriction is that the original notices has to be redistributed.

Selling software including free software is often misunderstood as not permitted, but that is not true. If the license of the free software is a copyleft license(eg. GPL) it can be sold, as long as the source code is made available for the people or companies paying for the software. If a company makes software derivated from GPL software, the derivated software also has to be released as a GPL-compatible license. This also means that if a company sells free software which is copylefted, no one can stop anyone who bought the source code to redistribute it to a third party.

If the license of the free software used in a product is a non-copyleft license(eg. LGPL, BSD) it can also be sold, but then the companies own source code of the program that uses the free software does not have to be open source, thus ensuring that the own source code can not be obtained by competing companies.

Using information(eg. maps in a GIS solution) licensed with Creative Commons in a commercial product is permitted, as long as it is not a NonCommercial version of the license, and often the original author of the information must be credited.

8.5 – Conclusion

When using open source software in a commercial product, and it is desirable to keep the code in the commercial product closed, the free software used must either be licensed under LGPL license, the modified BSD license, or any other non-copyleft license as these licenses allows proprietary software to be used with it. This means that when the proprietary software is sold the only code that has to be made available to the customers is the software/libraries used.

If the only option is to use open source software which is licensed under the GPL license in the commercial product, then the product itself also has to be free software.

A common misconception is that open source licensing does not allow companies to charge money for their software if it is GPL software, but the truth is that charging money for free software is not a problem. The source code does not have to be shipped with the program either, but if free software has been sold, the source code for the program has to be available for the ones who bought the program. Also, the ones that bought the software can not be stopped from redistributing the code, as it is a right given to them by the GPL license.

8.6 – References

http://www.gnu.org/ Home of the GPL and LGPL licenses. The GPL and LGPL version referenced to in this document is version 3, dated June 29 2007

http://en.wikipedia.org/wiki/BSD_license – Source of information regarding the BSD license.

http://www.apache.org/licenses/LICENSE-2.0 – Home of the Apache license. Version referenced to in this document is version 2.0, dated January 2004

http://www.opensource.org/licenses/cpl1.0.php – The Common Public License Version 1.0

http://en.wikipedia.org/wiki/Creative_Commons_licenses – Creative Common licenses

9 – GIS investigation

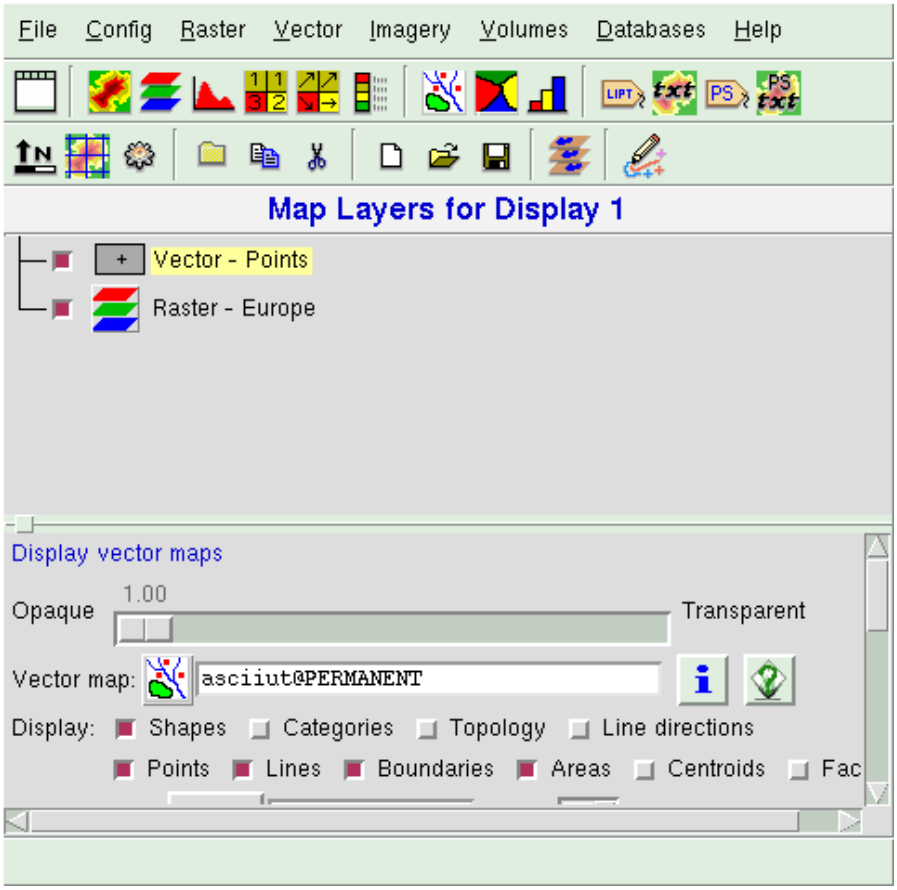
9.1 – Background

As my thesis project involves presenting data geographically using a GIS solution, which GIS to use has to be investigated and decided.

Among the numerous open source GIS solutions available for the Linux platform, the ones that seemed to be worth looking into was GRASS GIS and Quantum GIS. A way of presenting GIS data on a web server would also be appreciated. Only Linux compatible GIS solutions was looked into since Explizit wanted the project to be as open source oriented as possible.

9.2 – The alternatives

9.2.1 – Grass GIS



GRASS GIS 6.3.0

GRASS GIS is kind of the mainstream open source GIS solution. GRASS supports almost every standard map format (both raster and vector) through the use of GDAL/OGR libraries. GRASS is able to render maps and export them to various image formats, and can perform many different calculations on maps, both vector and raster maps.

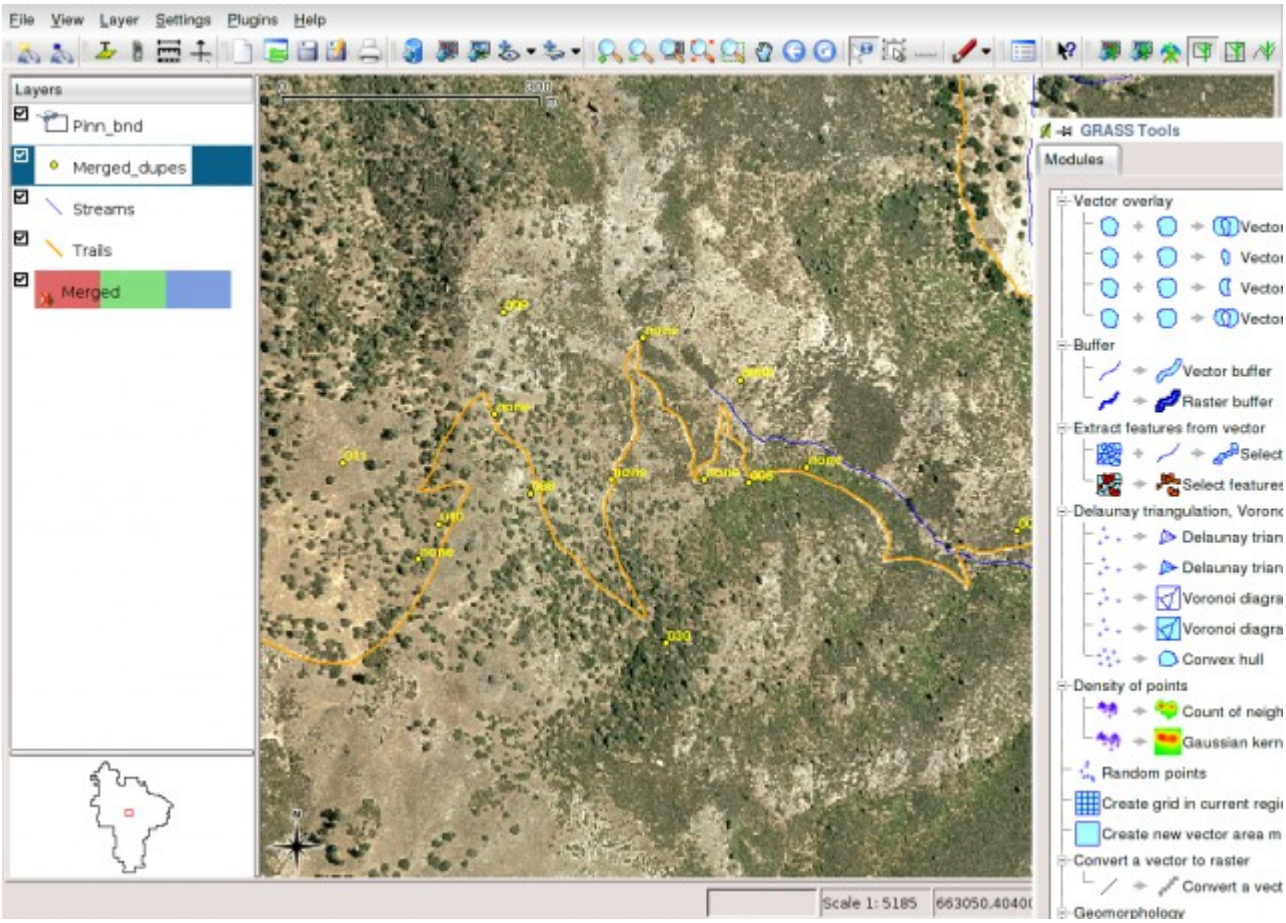
Developing software with GRASS GIS seems to be fairly easy, given that GRASS GIS comes with tons of documentation, and an active community. Binary packages are available for all the major Linux distributions (Ubuntu, Debian, OpenSUSE...), Microsoft Windows (through Cygwin) and MAC OS X.

The pros:
+Large community
+Well documented GIS functions
+Able to import and export to almost all standard GIS map formats

+Easy to use as base for developing GIS software
+Platform independent
+Good scripting capabilities

The cons:
-Complex GUI
-The user manual is not that good

9.2.2 – Quantum GIS

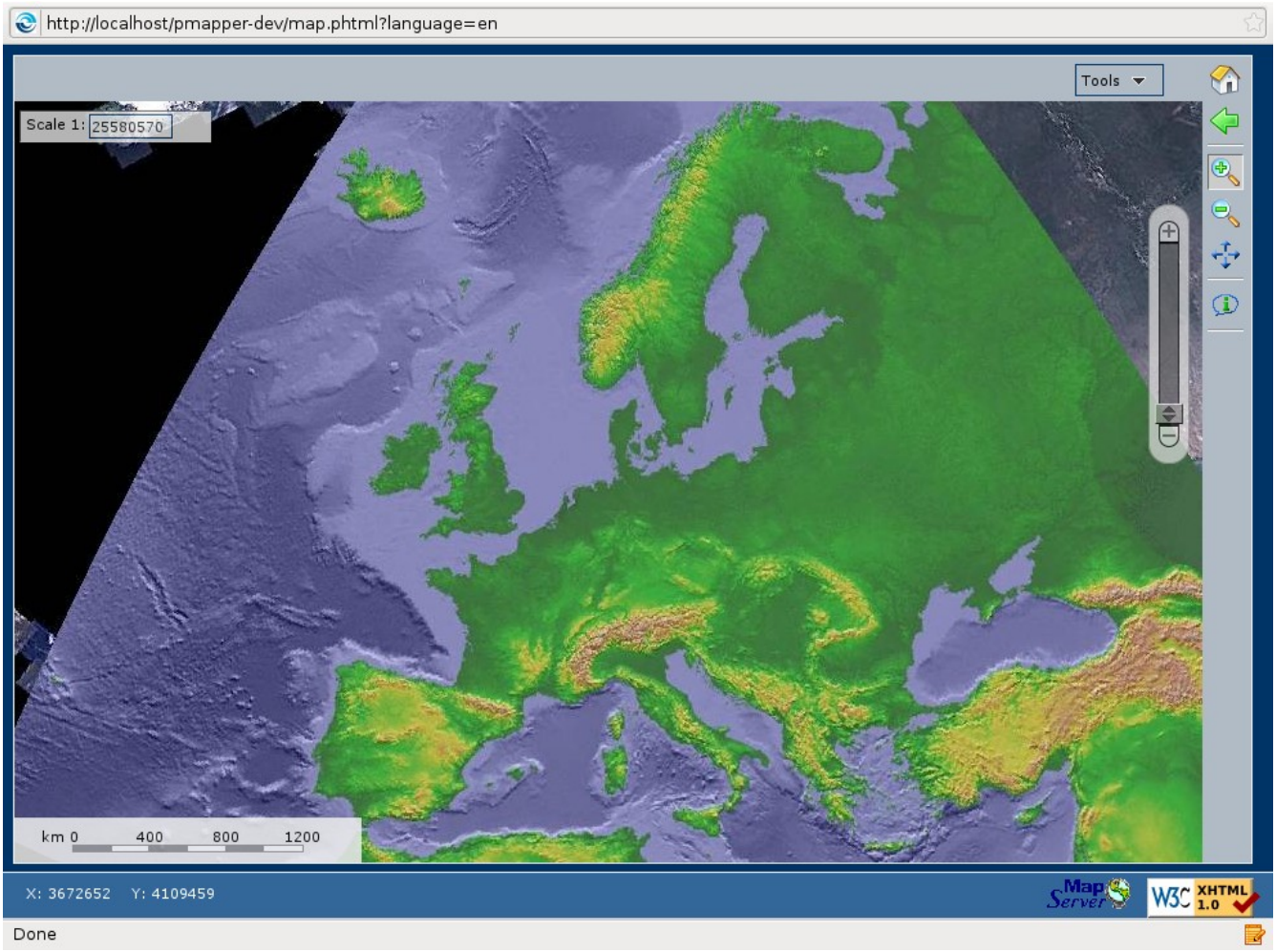


Quantum GIS(QGIS) is really good when it comes to rendering data, and it is also really easy to use. QGIS, just like GRASS GIS, supports most of the standard formats through the GDAL/OGR library. QGIS is by far more easy to use than GRASS GIS, but when it comes to more than just clicking in a GUI QGIS quickly starts loosing its advantage over GRASS GIS. In a poll on the Quantum GIS website, over 80% answered that they use QGIS as a frontend to GRASS GIS. That tells us QGIS is probably best used as just a frontend, and not as a development base.

The pros:
+User friendly
+Nice GUI

The cons:
-Not suitable as development base in my project
-Mostly a frontend for GRASS GIS

9.2.3 – p.mapper



p.mapper is not a standalone GIS solution, instead p.mapper is built upon PHP/MapScript and uses MapServer to render GIS map layers in a map on a web server. The map can easily be zoomed in, panned, queried and manipulated by the use of plugins. This is probably the best way of presenting GIS data in this project. Using a connection to a Web Map Server, maps can easily and at no cost be retrieved.

The pros:
+Platform independent
+Can read ESRI Shapefile-files, which can be exported from GRASS GIS
+Comes preconfigured with a NASA WMS(Web Map Server) that will come in handy

The cons:
-The map might load slow if the connection to the WMS server used is slow

9.3 – Conclusion

The first conclusion that can be done is that using open source software to accomplish the goal of mapping data from the pulse watch is possible.

GRASS GIS has all the functionality needed for the project and is also very good because of its adaptability, but using p.mapper as a complementary tool when presenting the GIS data is recommended.

GRASS GIS should be able to take the coordinates from the GPS/pulse watch, transform them into xy-coordinates understandable by p.mapper after being exported into a ESRI Shapefile. The integration between the two softwares will be coded in C++ in a GNU/Linux environment, and hopefully the communication with the GPS/pulse watch can be fully automated.

10 – The development

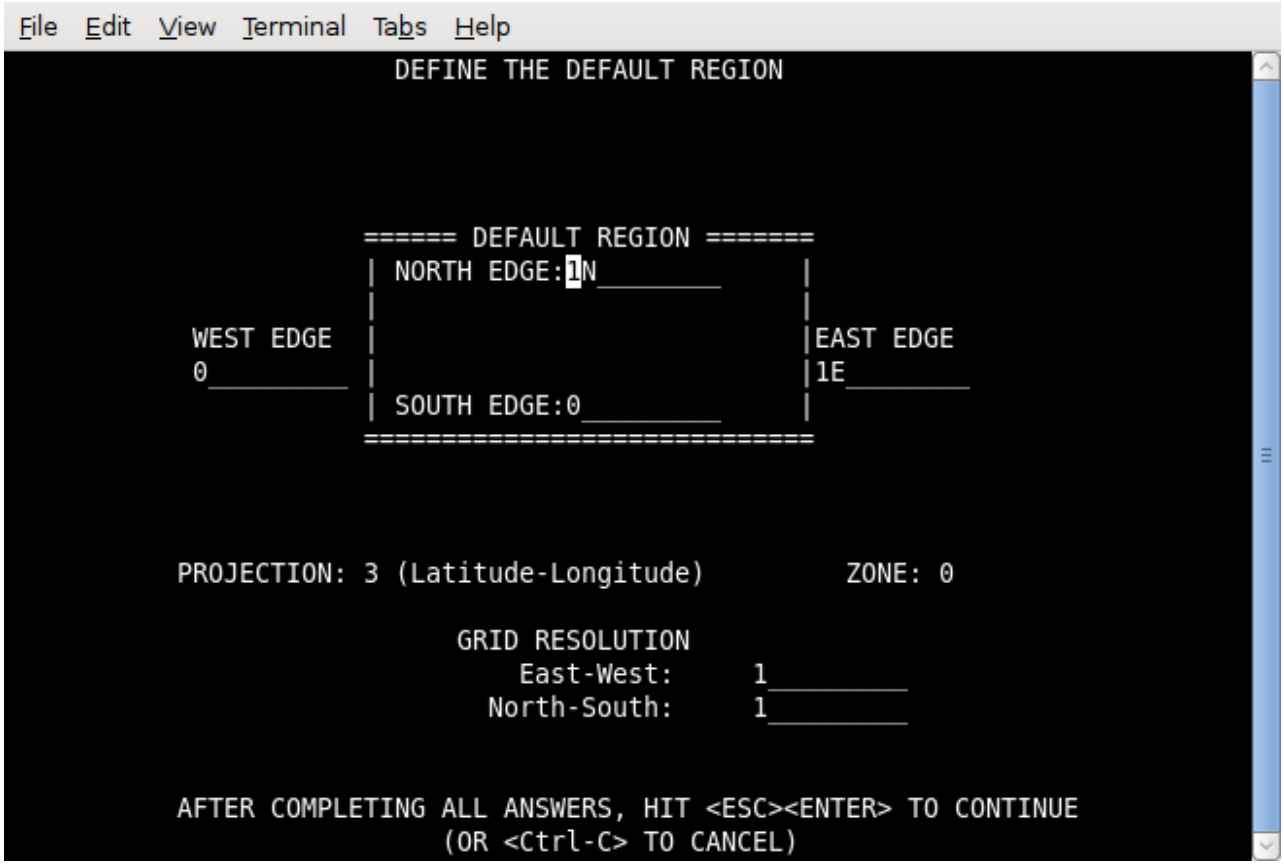
10.1 – Overview

First of all, which programming language to use had to be chosen. Since GRASS GIS is written in C++, it was easy to decide that C++ and wxWidgets was the language to write the FRWD/GRASS/p.mapper integration software in. wxWidgets is a cross-platform GUI toolkit which comes in handy when GRASS GIS also is cross-platform. If porting the program to another platform is needed wxWidgets makes that a whole lot easier. For those not familiar with wxWidgets, it is a framework much like Windows Forms and MFC.

10.2 - Coordinates

The initial problem when dealing with the different softwares was the coordinates. The FRWD software exported a text file containing the coordinates in GPS latitude-longitude. GRASS GIS had a X/Y-coordinate system which did not seem to follow any existing standard, and p.mapper used another X/Y-coordinate system.

GRASS GIS documentation said that when creating locations in GRASS, one could choose to use latitude and longitude coordinates, but not how the user should enter the coordinates when prompted to.



The screenshot shown above was all I had to go on, and with coordinates looking like 64º44'7580”N 20º58'4664”E it was hard to get a accurate region in GIS. Finally, I skipped entering minutes and seconds at the edges, and when only entering degrees I got the region covering the area I wanted.

After that I went on trying to get something out of the FRWD and into GRASS GIS. The software that came with the FRWD had an export function that exports data to a .txt file, which probably could be parsed in some way.

After 52 lines interesting things started showing up, with row after row looking like this:
20 25 2016 6445,1152 2058,9417 11,3 24,1 105 140 22,5 1002,7

The first column was marked “Time(s)”, the second “Laptime” and so on. The two really interesting columns was number 5 and 6, which was marked “N/S latitude (ddmm.mmmm)” and “E/W longitude (dddmm.mmmm)”. The letters “ddmm.mmmm” indicated that those numbers were actually WGS84 coordinates, which is standard GPS coordinates, and that they were written in decimal degrees. They weren't.

6445,1152 actually has to be interpreted 64 45 1152, where 64 is degrees, 45 is minutes and 1152 is seconds. When I found this out I tried getting these points to be understood by GRASS GIS, when GRASS GIS can import points from standard ASCII files.

According to the GRASS GIS manual, the points in the ASCII files had to be written in one of the following forms:

Acceptable formats:
key: D=Degrees; M=Minutes; S=Seconds; h=Hemisphere (N,S,E,W)

- *(+/-)DDD.DDDDD*
- *DDDh*
- *DDD:MMh*
- *DDD:MM.MMMMMh*
- *DDD:MM:SSh*
- *DDD:MM:SS.SSSSSh*

DDD:MM:SSh did the trick, but with four decimal seconds. When a coordinate was parsed from the FRWD file, and rewritten on this form GRASS GIS could import the point, and map this correctly in a region. The X/Y-coordinates I initially thought I had to work with could be ignored.

The integration between GRASS GIS and p.mapper also got easier than expected. At first, a SQL solution was attempted, but after having trouble getting a PostgreSQL database GIS enabled with PostGIS and then getting p.mapper to connect to the database, another approach was needed. When reading the documentation for p.mapper/ MapServer I found out that ESRI Shapefiles, which GRASS GIS could export vector maps to, could be read by MapServer. After exporting a point map from GRASS GIS, and configuring p.mapper I finally got it working without having to “manually” convert any coordinates at all.

10.3 – Parsing the FRWD text file

The text files exported from the software starts with 52 rows of data that can be ignored, and on row 53 the interesting data starts. From row 53 and onward the data captured(pulse, coordinates etc.) is presented, and in a consequent way. Each row has 15 attributes, in the following order:

Label:	**Sample value:**
Time(s):	4
Laptime:	Yes
North-South(m):	433
East-West(m):	0
N/S latitude(ddmm.mmmm):	6445,4679
E/W longitude(dddmm.mmmm):	2053,1959
Altitude(m):	23,5
Speed(km/h):	14,6
Distance(m):	20
Heart Rate(bpm):	134
Temperature(celsius):	24,4
Pressure(mbar):	1003,9
Pitch Angle(deg):	0,2
Training Effect:	0
EPOC:	0

With one sample a row and the above values always separated by a TAB character(0x09) the file was fairly easy to parse. One big while loop, ensuring that data was available, containing code for parsing and writing the data to a ASCII file is the main function of the C++ class I wrote called TextParser. The ASCII file written to is a file called lines.dat which is placed under /tmp in the Linux environment the program is running on. /tmp is a good place for such a file, because all files in that directory is automatically deleted when the system is shut down. That means no paying attention to deleting the temporary files for the program, and it is not a problem because lines.dat will never be needed when not created by TextParser.

Sample output lines.dat:

1|20:53.1828E|64:45.4600N|130|8,2km/h|0m

2|20:53.1825E|64:45.4614N|131|8,3km/h|3m

3|20:53.1828E|64:45.4629N|131|9,1km/h|5m

4|20:53.1846E|64:45.4647N|132|10,4km/h|9m

5|20:53.1888E|64:45.4666N|133|12,4km/h|14m

What the parser prints in lines.dat is one row per sample, and six columns per row. The columns are as follows:

1. Category – The sample number, starts at 1 and increments by one each row
2. Longitude – Coordinate in E/W (Degrees:minutes.seconds)
3. Latitude – Coordinate in N/S (Degrees:minutes.seconds)
4. Heart rate (bpm) – Heart rate in beats per minute
5. Speed – The speed at which the FRWD was moving at the time of the sample
6. Distance traveled – Distance traveled since sample number one.

Each column is separated by the character | which is the standard field separator in GRASS GIS ASCII files.

10.4 – Getting FRWD data to and from GRASS GIS

Importing data in GRASS GIS from the file created by TextParser was fairly easy. GRASS has a built in CLI, and one of the commands is called v.in.ascii. v.in.ascii builds a vector map, which is a map consisting of points whom contain data. When importing ASCII files, one sets the name for the resulting map, and also defines the different columns names. Example import command:

v.in.ascii --overwrite input=/tmp/lines.dat x=2 y=3 cat=1 output=frwd format=point fs='|' 'columns=cat int, x varchar(20), y varchar(20), pulse int, speed varchar(16), distance varchar(16)'

Exporting data from GRASS GIS to a standard GIS format was easier than importing it. Again, the CLI could be used with the command v.out.ogr. OGR is an open source library for reading and writing many different vector file formats. With p.mapper, the ESRI Shapefile format was needed, and this was easily achieved with v.out.ogr by entering Shapefile as output format. Example export command:

v.out.ogr -e input=static type=point dsn=/tmp/static/ olayer=static layer=1 format=ESRI_Shapefile

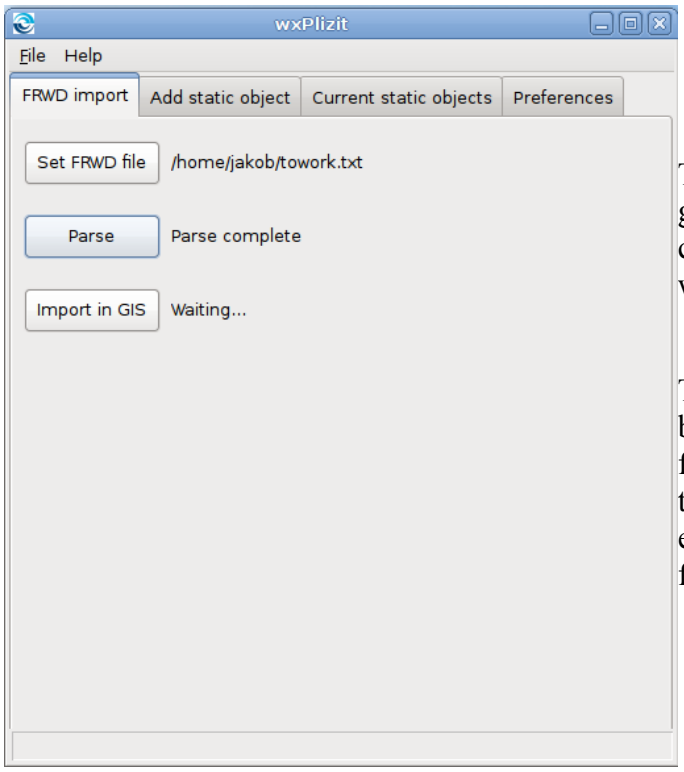
10.5 – p.mapper integration

Since p.mapper reads ESRI Shapefiles only two things had to be done. The vector files had to be copied somewhere where p.mapper could easily access them, and a layer with the shapefile had to be created in the p.mapper configuration file. Finding a good spot for the vector files wasn't hard, since p.mapper already had some data in a subdirectory to /var/www/. The directory was renamed to pmapperdata and all the files necessary was put in it.

The additional configuration that had to be done in p.mapper was in the so called map file, which defines all the different layers to be displayed in p.mapper. I simply added a layer called “FRWD” which takes data from a ESRI Shapefile called “frwd”.
Example layer in the map file:

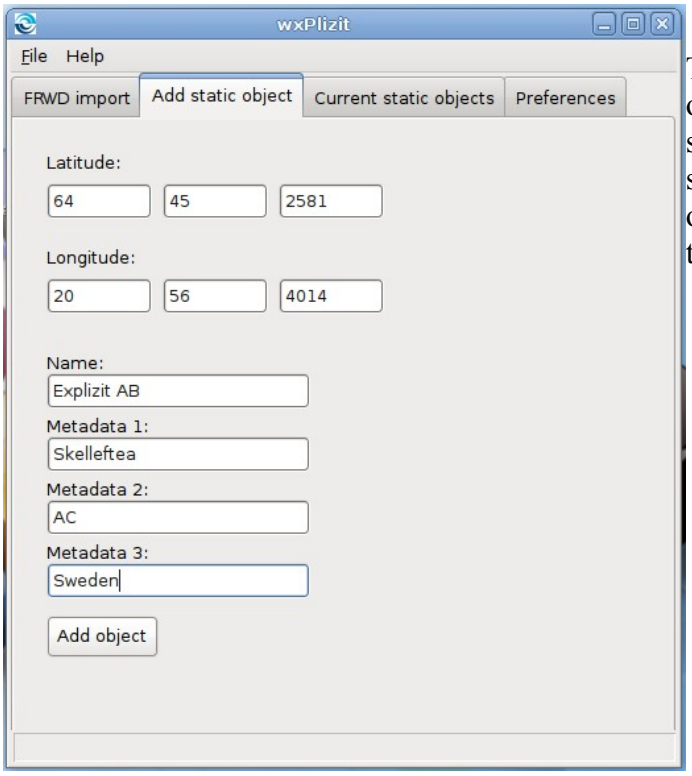
```
LAYER
  NAME "frwd"
  TYPE point
  DATA "frwd"
  PROJECTION
    "proj=latlong"
    "ellps=WGS84"
    "datum=WGS84"
  END
  TOLERANCE 4
  TOLERANCEUNITS pixels
  STATUS ON
  METADATA
    "DESCRIPTION" "FRWD"
  END  # Metadata
  CLASS
    Name 'FRWD sample'
    COLOR 255 0 0
    SYMBOL 'square'
    SIZE 4
    MINSIZE 4
    MAXSIZE 4
    TEMPLATE void
  END  # Class
END  # Layer
```

10.6 – wxPlizit

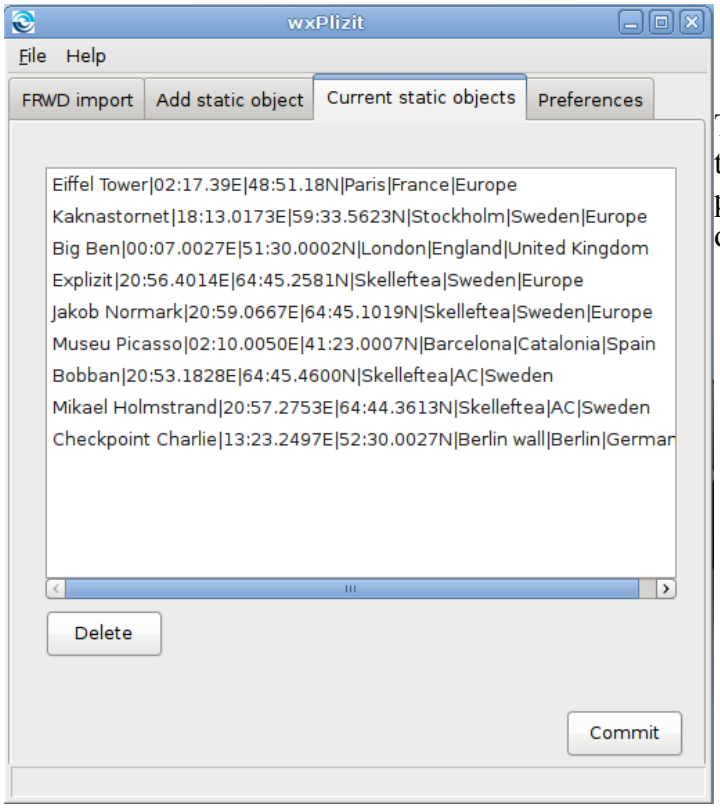


The GUI for importing FRWD text files, parsing them, getting the data to GRASS and then to p.mapper is called wxPlizit. This is a screen shot of the first of wxPlizits four tabs.

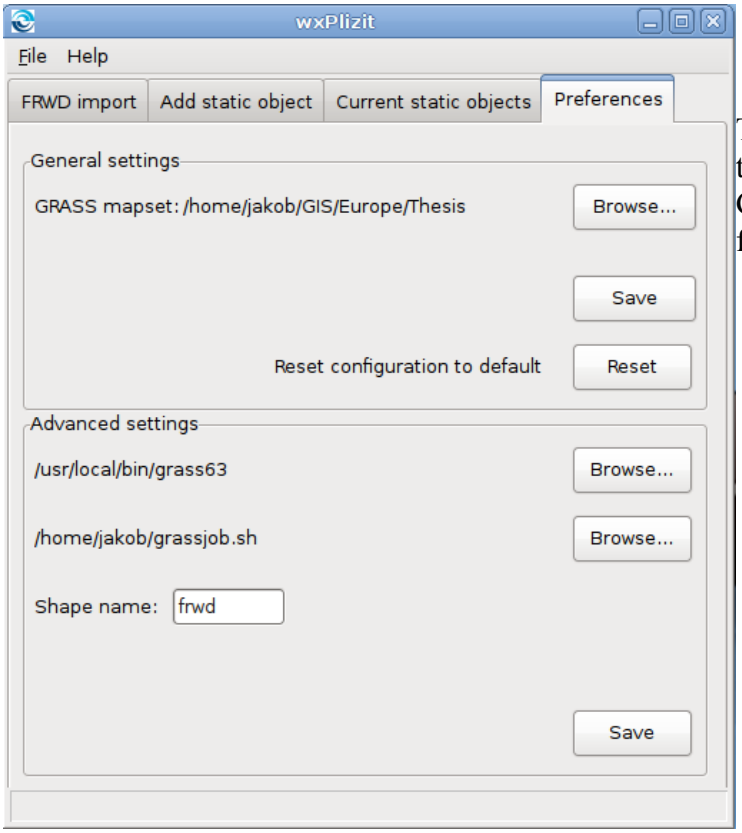
The “FRWD import” tab has three buttons. The first button lets the user browse and select the FRWD text file, the second button parses the selected file and the third button imports the parsed data in GRASS GIS, exports it to a ESRI Shapefile and copies the vector files to p.mappers data directory



The second tab in the program is called “Add static object”. The supervisor at Explizit wanted the software to be able to take coordinates and some simple metadata and map this in GIS. The static objects require lat-long coordinates, a name and three metadata tags.



The third tab - “Current static objects” - manages the static objects that wxPlizit, GRASS GIS and p.mapper currently share. The user can view and delete points, and commit the changes done.



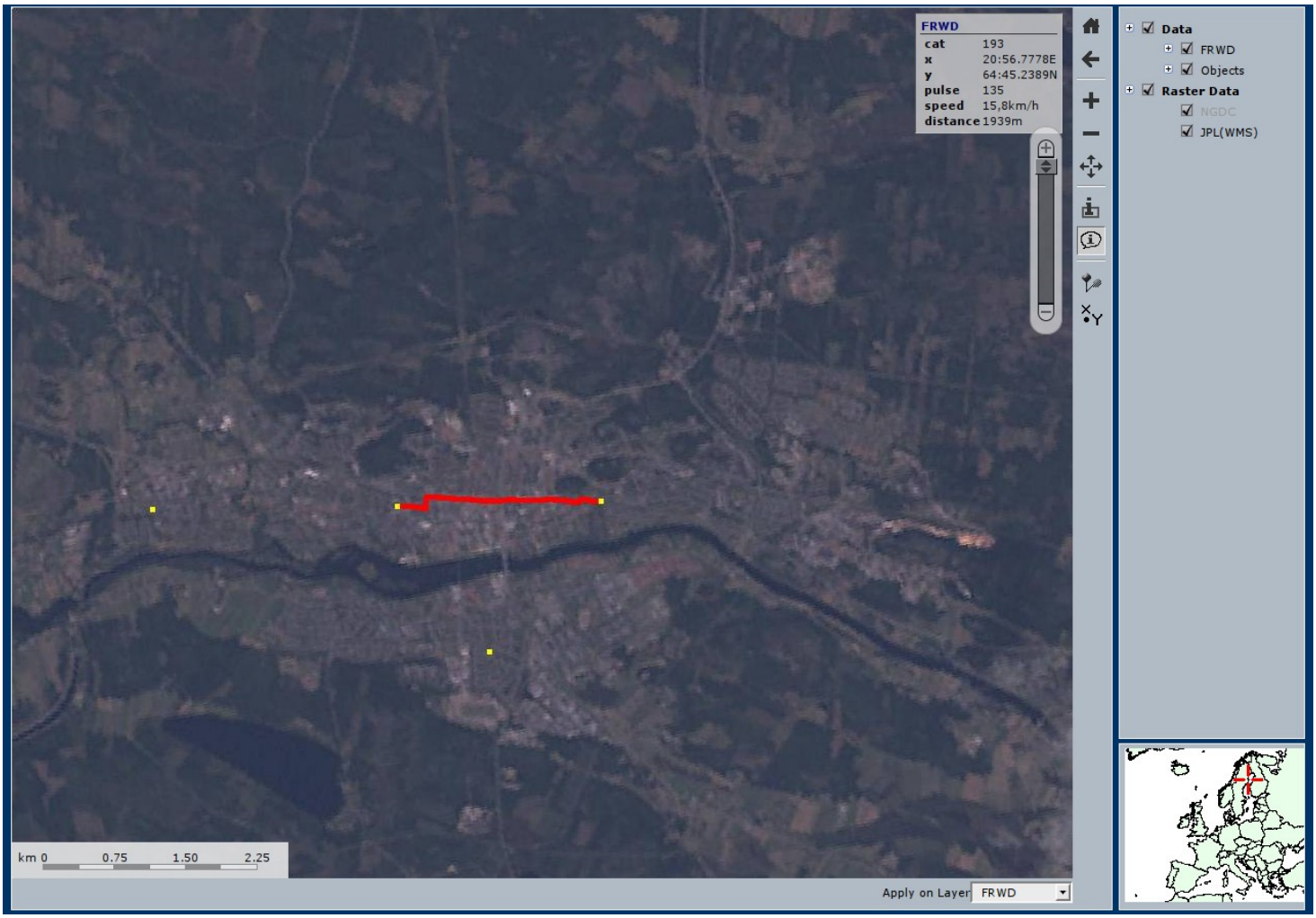
The fourth tab in wxPlizit contains settings for the most important parameters concerning GRASS GIS, the generated vector file and script files.

A Linux environment was used to build this program, all in C++ with the GUI in wxWidgets. The code should be easy to port to other operating systems, since the code does not use that many platform dependent commands. With both wxPlizit, GRASS GIS and p.mapper/MapServer being platform independent, the whole software suite has great adaptability.

Currently, the software is optimized for 32-bit Ubuntu Linux/Debian, with a .deb-package available for installation which depends on wxWidgets 2.8, GRASS GIS 6.3, Apache2 with php5 and php5-mapscript. A custom version of p.mapper is included in the package.

The package can easily be ported to rpm, or any other Linux package because of the simple structure of .deb-packages. The contents of the .deb-package can after they have been extracted simply be copied to the root of a Linux root-partiton to install it. Doing this will have the disadvantage of removing all automatic dependency checking, but they can fairly easy be manually resolved.

After a FRWD file has been parsed, imported to GRASS GIS and then exported to p.mapper the result looks like this:



This is the p.mapper application - running in Firefox web browser - showing data from when I biked from home to Explizit(from east to west on the map, the red line is the path).

Skellefteå has been zoomed in and a random point queried. As seen in the image sample number 193 was queried, I moved at 15.8km/h, had a heart rate of 135 and I had traveled 1939m from the start point. You can also see the coordinates(64:45:2389 north 20:56.7778 east), and if other data from the FRWD is needed, it is not hard to change the code in wxPlizit to accomplish this.

11 – Discussion

As all software, wxPlizit has weaknesses. The program is thoroughly bug tested, and should not contain any major bugs. However, there are a few known bugs. For example, the coordinates that is taken, either from the FRWD data or entered as a static object, has to be a point in the north east hemisphere. This is because of how TextParser parses data from the FRWD, and this must be fixed if the program has to be able to map data elsewhere. Since the FRWD had to physically be in another hemisphere to present coordinates from there example coordinates could not be obtained. Thus no sample coordinates to parse. Other than that, the software can be considered stable.

~~*~*~*~*~*

GIS was a fun thing to learn since it has so many areas of use. It was not a easy thing to learn though, seeing that I barely knew what it was when I started working with it and that the documentation regarding GRASS GIS is anything but self-explanatory. I did get a few bonuses though, because I ultimately had to compile GRASS GIS myself instead of using pre-built binaries, and by doing that I got a even better understanding about Linux and compiling from source. I also learned a few things about SQL databases, when GRASS GIS databases can be synchronized with SQL databases, and especially PostgreSQL. If I had been able to put more time into GIS enabling PostgreSQL I would have got it to work. But the solution with Shapefiles is just as good for wxPlizit.

~~*~*~*~*~*

Working in a project group in my thesis project was a great thing to do. Help was almost always available in some form, and Explizit really gave me good prerequisites to get a good result from my thesis project. Also, the time plan I made at the beginning of the project was perfect. I never ran out of time with a certain part of the project, and the plan could be followed without any adjustments.

The project was not flawless though, as in my opinion too few project meetings was held. One meeting took place in the start, and one meeting in the middle of the project(even then everyone didn't participate). Even if the whole project group couldn't have participated, a few more meetings should have been held. Also, i had no backup FRWD. If the FRWD unit had failed I almost certainly would have to switch to a standard GPS unit.

~~*~*~*~*~*

About the initial problem, “A prototype for mapping data geographically built on a open source GIS is needed.”. It can be done with only open source software, I did it and it can be sold for profit as a commercial product.