MASTERS PROGRAM IN

GEOSPATIAL TECHNOLOGIES

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

UNIVERSITAT

WESTFÄLISCHE

gvSIGDroid

An open source GIS solution for the Android platform

Dissertation supervised by

PhD . Joaquin Huerta

Co-supervised by

Prof. Antonio Krüger (WWU)

Prof. Miguel Neto (ISEGI)

March, 2009

MASTERS PROGRAM IN

GEOSPATIAL TECHNOLOGIES

gvSIGDroid

An open source GIS solution for the Android platform

Cristian Martín Reinhold

Dissertation submitted in partial fulfilment of the requirements
for the Degree of *Master of Science in Geospatial Technologies*

gvSIGDroid

An open source GIS solution for the Android platform

Dissertation supervised by

PhD . Joaquin Huerta

Co-supervised by

Prof. Antonio Krüger (WWU)

Prof. Miguel Neto (ISEGI)

March, 2009

gvSIGDroid

An open source GIS solution for the Android platform

INDEX

Page

ACKNOWLEDGMENTS

I would like to thank professor Joaquín Huerta Guijarro and Michael Gould for the time and attention spent on this master project; additionally, many thanks to the Erasmus Mundus Program for making this Master possible.

Further special thanks go to Miguel Monteninos Lajara, Juan Lucas Domínguez Rubio and Carlos Sánchez Periñán from the *Prodevelop* Company for providing me with numerous helpful hints during the realization of this project. Finally, I would like to express my gratitude to my family, my best friend, and friends, who always trusted in me and were a big support.

To my mother.

gvSIGDroid

An open source GIS solution for the Android platform

ABSTRACT

This project aims to cope with the lack of GIS programs in mobile devices that enable both visualization and editing of almost any kind of geographic information. More specifically, it attempts to reach the new potential users emerged from the Android platform. With gvSIGDroid, which is based on gvSIGMobile, users will be able to retrieve, visualize, navigate and modify Geographic Information (GI), all this in a more suitable, user-friendly interface, especially designed for the Android platform. Based on this prototype, future extensions can be deployed to provide missing functionalities such as Location Based Services and data sharing.

KEYWORDS

Mobile GIS

User Interface Design

Open source

gvSIG

GIS application

ACRONYMS

I. GENERAL ACRONYMS

API – Application Programming Interface

ASL – Apache Software License

BOM – Bill Of Materials

BREW – Binary Runtime Environment

ESRI – Environmental Systems Research Institute

FOMA – Freedom of Mobile Multimedia Access

GI – Geographic Information

GIS – Geographic Information System

GNU – "GNU's Not Unix" (Operating System)

GPL – General Public License

GPS – Global Positioning System

GUI – Graphic User Interface

HDL – Hardware Description Language

J2ME / JME / JavaME – Java 2 Micro Edition

LINUX – Virtual Machine

MDI – Multiple Document Interface

MIT/X11 – Massachusetts Institute of Technology

MVC – Model-View-Controller

NEC – Nippon Electric Company

OGC – Open Geospatial Consortium

PCB – Printed Circuit Board

PDA – Personal Digital Assistant

QWERTY – Acronym referring to the English keyboard layout

RIM – Research In Motion

SDK – Software Development Kit

UID – User Interface Design

UNIX – Virtual Machine

VM – Virtual Machine

WMS – Web Map Service

II. FILE EXTENSIONS ACRONYMS:

.GVM – gvSIG for Mobile

.DEX – Dalvik EXecutable (Dalvik VM files)

.XML – eXtended Markup Language

.KML – Keyhole Markup Language

.KMZ – Keyhole Markup Language, Zipped

.GPX – Graphics

.GML – Geographic Markup Language

.JP(E)G – Joint Photographic (Experts) Group

INDEX OF TABLES

INDEX OF FIGURES

1. INTRODUCTION

1.1. Mobile applications and Geographic Information Systems (GIS)

As Goodchild already postulated in 2004 (Goodchild, et al.), due to new mobile devices, mostly in the form of PDAs *(*Personal Digitalized Assistant) and mobile phones, access to information and services nowadays has changed from desktop devices to ubiquitous computing. This new environment has led to another market of mobile users to which companies are addressing their new software releases.

Therefore, software providers are now migrating their traditional desktop software applications to the mobile realm, readapting their applications so as to obtain the best mobile users' experience on an user-centered scenario (Boll, Breunig, et al., 2004), and offering their solutions either as open or private source software.

These new mobile applications aim to offer a variety of services to the final user, which can be re-grouped based on two different factors: professional/recreational applications and expert/non-expert applications (see section 2.2).

For GIS users there is a need as well to develop *Just-In-Time* applications (Kjeldskov, 2002) for retrieving, viewing, exploring and modifying spatial data at any place at any moment in time. That is the reason why there is also a proliferation of mobile GIS applications, which can be generically classified into the following types:

A. Field-based GIS:

Since the earliest centuries humans have used maps to collect and visualize geographic information (Brown, L.A., 1949). With the digitalization age, these maps were usually translated into digital format at the computer. Today, mobile devices can also perform this task in the field. To accomplish this endeavor, however, it is imperative to offer full enhancement for visualizing, exploring and editing geographic information and its attributes. Hence, editing capabilities are imperative in this context.

B. Location Based Services (LBS) application:

Their aim is to offer location-based functionality, such as navigation, street finding, route tracking, and so on. These services do not necessarily fall within the scope of current gvSIGDroid functionalities since *Google Maps* applications already offer this possibility on Android. Nevertheless, integration of a GPS tracking system into future releases may prove to be fruitful as it would enable the user to access real-time locations from within the application.

1.2. Objectives

Based on the mobile GIS context explained in section 1.1, the main objectives of the present work are:

- To review the current state of the art of mobile devices and applications, stressing the advantages of open source solutions.
- To emphasize the importance of the User Interface Design (UID) phase against the implementing phase. Developers should generally invest more time in designing rather than implementing since *"the decisions made during the design process have a great effect on the cost of a product but cost very little"* (Ullman, 1997).
- To present the main characteristics and architecture of gvSIG, from which gvSIGDroid was derived, to provide background to the application developed.
- To offer an open source GIS solution for a newly developed platform, Android, so that users of that platform can also benefit from GIS functionalities.

1.3. Assumptions

- The present application provides a starting point for future GIS mobile device frameworks built upon the Android platform.
- GIS users need an application on Android to do field-based data gatherings, allowing users to navigate, analyze, and edit geographic information.
- Most of the users of the Android platform will be:
    a) Non-GIS experts that possess an Android device
    b) GIS experts with no previous knowledge of gvSIG
    c) A minority will be experienced gvSIG users

1.4. Dissertation Organization

The present paper is divided in the following chapters: Chapter 2 introduces some theoretical concepts in order to understand the background of the project, such as the different types of mobile applications and the importance of mobile User Interface Design. In Chapter 3, a brief introduction of currently available mobile GIS applications and systems is given. After that, the Android platform is presented along with gvSIG and gvSIGMobile in order to understand integral parts of the architecture from which the gvSIGDroid application is derived. In Chapter 4, a more in-depth description of gvSIGDroid prototype's structure is detailed via use case diagrams, sketches, package diagrams, and class diagrams. Finally, Chapter 5 discusses conclusions of the research undertaken with an emphasis on possible future modifications.

2. THEORETICAL CONCEPTS

2.1. Chapter Objectives

The present chapter explains the core theoretical keys for a better understanding of the project background. First, in section 2.2, a distinction between the different types of mobile applications and their use of GIS is explained. Exemplary applications range from travel support to geotagging functionalities. Then, in section 2.3, some interface design issues are pointed out to describe essential considerations for the design of mobile platform software.

2.2. Mobile (GIS) applications

When developing mobile applications, it is important to identify required functionalities and target audiences. Thus, the future application should be evaluated in terms of two main factors: 1) whether it is a serious application or not, and 2) whether it should be used for specialized and/or non-specialized users.

Although information and services provided by mobile applications vary depending on the type of application being used (Lee, Kim, et al., 2004), they can be classified as follows:

A. Communications:

Examples of communications applications are e-mail clients, instant messaging clients, news/information clients, web browsers, on-device portals (like java portals), social network clients, and so on.

Regarding the metric factors, all communications applications share a high component of seriousness and are targeted to entertainment purposes so generally non-expert users utilize them. In the specific case of social network clients, this changes since there is no need of expert users as it contains a bigger component of fun for the user.

B. Games:

Games, or recreational applications, can be divided into:

- Puzzle/strategy (e.g. Tetris, sudoku, chess, board games)
- Cards/Casino (e.g. Solitaire, Blackjack, Roulette, Poker)
- Action/Adventure (e.g., Doom, Role-Playing games)
- Sports (e.g. Football, Soccer, Tennis)
- Leisure Sports (Bowling, Pool, Darts)
- Learning games (Trivial pursuit, flight and driving simulators)
- Others

When developing game applications, allowing *immersive multimedia* (Lopez-Gulliver et al., 2002) is also a plus to make users feel like they are in a more realistic environment and to increase their level of enjoyment; together with other factors such as surround sound, interactive user-input, simplicity and functionality.

As for the metric variables, all game applications are largely recreational (very playful) and are targeted to the entertainment users community (generally non-expert users).

C. Multimedia – Players and Viewers

They are essentially graphics/image viewers, presentation viewers, and video/audio/streaming players. Most of these applications are mainly tools for entertainment activities.

D. Productivity

Examples of this type of programs are calendars, calculators, diaries, notepad/memo/word processors, spreadsheets, directory services (e.g. yellow pages), banking/finance, and so on.

Productivity applications are professional, non-recreational applications, and commonly addressed to a specific, expert group of users.

E. Travel

GIS applications (both LBS based and field-based) are traditionally part of this category, since they generally are used to handle spatial data information for traveling purposes.

Travel applications can be divided into: city guides, currency converters, translators, navigators (LBS app), Global Positioning Systems (GPS), Map viewers (GIS apps), itineraries/schedules (can be GIS-based), weather (also GIS-based), and so on. They are generally professional tools for targeted groups of specific users, so they could be considered comparable to productivity applications.

F. Utilities

Utilities applications are used to facilitate the daily mobile device usage and usually provide specific, brief and simple information within a minimal interface framework. Examples of these are profile managers, idle screens / screen savers, address books, task managers, call managers, file managers, color pickers, and so on. They are somehow neutral to the professional/recreational and expert/non-expert user factors, though slightly shifting from the center point. For example, it could be considered that selecting a color is subjective to the user's taste, as there is somehow a bit of playfulness when choosing a preferred color.

But in what kind of applications is GIS applied to today? Initially, mobile GIS-based applications were used in travel applications, but now they are spreading into almost every application type. For example, we can have a geocaching game that uses GPS services, a camera utility that geotags image location, and so on.

Therefore, when summarizing all these different application types in relation to the outlined metric factors, the following graph is produced:

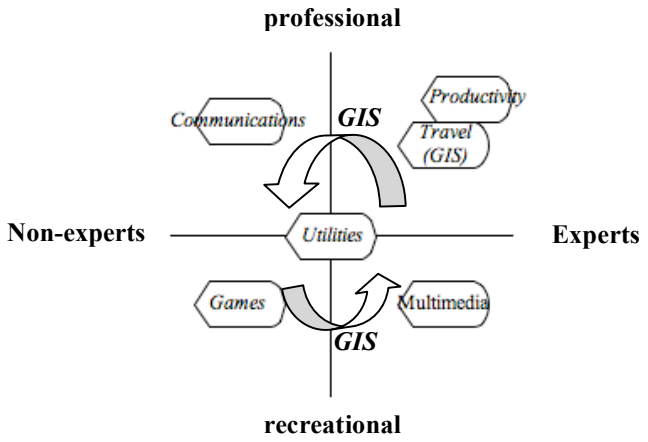


Figure 1. Application types graphic based on their metrics.

There are several companies that are working on the development of mobile GIS applications, emphasizing various combinations of these metric groups. Within the proprietary GIS sector, ESRI (Environmental Systems Research Institute, 1969) is considerably prominent. Meanwhile, open source GIS communities are trying to join efforts in developing common libraries in order to share knowledge and reuse code. More specifically, *Open Source Geospatial Foundation* (OSGeo, 2006) is a clear example of this, since it encompasses several GIS projects beneath the same umbrella, which are not only addressed to desktop developers but also to the mobile community. They offer a specialized wiki for mobile GIS within their *MapTools* project. Unfortunately, this subproject is still at the initial planning phase. Yet, their main objectives have already been defined (MapTools Mobile, 2008). These goals include:

- Fully Open Source
- Compatibility with a large variety of mobile devices
- High portability across different operating systems:

  Linux, Palm OS, Symbian OS, Windows CE/XP Embedded
- Multi-language support

- Numerous data types support:

  Vector, raster, database formats, …

- Integration of GPS

A,more detailed description of open source GIS applications will be offered in section 3.3.

2.3. User Interface Design (UID) for Mobile Applications

2.3.1. The importance of User Interface Design

Interaction design with users is crucial since a bad design can lead to anger or frustration of the final user, and vice-versa (Klein, Moon & Picard, 2002). Moreover, expressive interfaces can be used for making the *look and feel* appealing, by correcting the use of color, icons, text, buttons and sounds for conveying a specific emotional state (e.g., happy, sad). So, having an attractive set of interfaces can convince users to overlook negative aspects of an interface (e.g. low loading/download rate), improving the overall impression of the program. This research field, in which emotions are highlighted when enhancing the value of products, is known as *Hedonomics* (Khalid, 2006).

To achieve this, developers should be aware of the different possible scenarios in which users would show frustration, which are:

- An application crashes or doesn't work as expected.
- The user's expectations are not fulfilled.
- The appearance of the interface results are noisy, annoying or garish.
- Not enough information is provided to inform the user what to do at any time.
- Insufficient information is given to explain the cause of an error or communicate why an action cannot be performed.

- Vague, condemning or unclear error messages are shown, which generally increases the user's frustration. Possible countermeasures are:
  - Avoid words like FATAL ERROR, INVALID, BAD.
  - Word ERROR only if necessary, it is better to write INFO.
  - Avoid uppercase in error messages and long code numbers.
  - Avoid vague messages like "error of type 0". Explanations should be precise and concise.
  - Provide context-sensitive help: possible reasons or solutions to do the right action.

On the other hand, in order to achieve a good interface design, it is important to follow three basic principles:

1. Focus on the user as early as possible, especially in the design and evaluation phases of software development. Users are the final targets; therefore it is essential to consider their needs.
2. Identify and document specific usability and user experience goals to help the program fulfill the end user's expectations.
3. Be aware that redesign is inevitable, so developers will have to create iterative versions to optimize their design.

The very first phase of the application design is to determine to whom an application is addressed, as well as which functionalities are to be provided. This phase is known as *requirements analysis*. When starting this analysis, developers can be faced with two different scenarios:

1. Starting from scratch:

   There is no pre-implemented version of the applications available. Then developers should brainstorm the features to be supported, and then contrast them with other related applications to see how these features are handled.

2. Starting with a desktop version:

   In this case, developers need to define a new solution for this new environment. Since it is indeed a different context, mobile UID principles differ from desktop ones. For the mobile version, most of the already defined *desktop requirements* end up being of no interest. Therefore, there is a need to focus on selecting just the crucial functionalities so that the application does not result overloaded.

In any of both cases, in order to be successful in the requirements analysis, developers need to adhere to the following steps:

- Summarize the application's purpose.
- Identify the application's end user, e.g. non-expert, expert, youth, seniors, men, women, and so on. This helps reveal what the focus of the application should be and supports the production of the first sketches.
- List the functionalities you want for your application.
- From all the functionalities, identify the most important ones, then review each in turn to refine an ideal featur. In the case of uncertainty, it is preferable to reject a functionality to retain simplicity. Additional functionalities can always be implemented at a later iteration.
- *Less is more*. Therefore, it is better to say "No" most of the time during software development. That is good because a study says that 80% of the program users generally use a 20% of its functionality.
- Define the user's conceptual/mental model. For that, identifying objects, tasks, roles and relationships is needed.

Then, once the conceptual model is clear, it is important to devote plenty of time to designing and redesigning the application's user interface. One possible option is sketching, or manually depicting the different layouts of the application's screens to refine final interface designs. Investing this time to achieve an optimal interface application is a low-cost way to strengthen consumer satisfaction.

In order to draw good sketches it is important to be familiar with the target device. For this, one should have various and different programs installed in the device/emulator and use them for a substantial period of time. In this way, one can observe from the user's perspective how buttons and other GUIs are implemented on a specific device before sketching (see section 3.8).

2.3.2. Roots for a better mobile application's design:

The following hints are based on the iPhone Guideline (iPhone HIG, 2008) and some notes taken from some iPhone presentations:

1. As said, it is important to become familiar with the device first.
2. The finger is not a mouse.

   Enabling people to use the finger instead of any intermediate device (such as a mouse) provides them a sense of control. In order to enhance this direct manipulation, it is important that:

   - Objects on the screen remain visible while performing actions on them.
   - The result of the user's action should be immediately apparent.
3. Make it obvious:

   Buttons, images and other objects that are clickable for the user should be big enough to cover the finger size or, in case of being smaller, its touch area should cover the mentioned size - e.g. 44px.
4. It is not easy to type:

   Even providing a built-in keyboard, it is preferable to use text fields only when strictly necessary. If not, use other options such as lists or spinners (combo boxes) for an option to be chosen between a restricted set of possibilities.

5. In general, a mobile device is much more constrained in terms of capacity, memory, processing power, battery duration, screen size, and so on. The screen is much smaller than that of a computer. Therefore, an optimal placement of graphical items is a must. Do not overload the screen with items. In terms of energy efficiency, it can be significantly improved even without any changes to hardware by using software solutions that reduce overheads and improve processor utilization (Silven et Jyrkkä, 2007). Large savings can be expected from applying architectural approaches that reduce the volume of instructions fetched and decoded. Obviously, compiler technology is the key enabler for improvements.
6. While the application is running, one should be aware that specific scenarios must be handled differently than they would on desktop applications:
    a. The user is probably mobile and desires easy content access.
    b. Users may need to jump in and out of an application when a phone call comes in, an email message arrives, or they decide to listen to a song.
    c. Users could also try to initiate a phone call, open their email, or view a *Google Map* while using the application.

    Therefore, it is important to handle those specific scenarios on each layout of your application.
7. About Aesthetics
    a. Do not overload with items

Figure 2. View of a single Web page on a mobile phone (from Kaasinen, 2008)

b. Content information by importance (e.g. use grouped tables)

c. Use pleasant colors and do not overload with different colors.

d. Add control buttons that manage view items (or filtering)

e. Avoid defining home, quit and back buttons (back button is provided)

f. Do not show too many buttons, makes confusion and irritates.

g. Allow customization of your application (order by options, change color, change settings, and so on).

h. Use standard, built-in controls. Don't reinvent the wheel. Extend from them.

i. Iterate the paper sketches at the design phase for a better solution.

3. Related Work

3.1. Chapter objectives

This chapter provides with the general information about related work done on the mobile realm. First, in section 3.2, a short introduction about current mobile operating systems is explained. Then, in sections 3.3 and 3.4, the reader will be able to obtain general knowledge of the current state of the art of mobile GIS applications available as well as the importance of open source projects. Lastly, in section 3.5 and 3.6, a brief introduction to the Android platform together with gvSIG platform is described.

3.2. Mobile Operating Systems (OS)

Mobile devices are generally referred to as smartphones, mobile phones, pocket PCs and PDAs (Personal Digital Assistants), but do not only include these; they are any kind of device that is pocket-sized and can be used while being held. These mobile devices, from a technical point of view, can also be divided based on the runtime environment in which they are being executed:

1. Native platforms and operating systems: Symbian, Windows Mobile and Linux (Android, OpenMoko, …).
2. Mobile browser runtimes: Webkit, Mozilla/Firefox, Opera Mini and RIM (Research In Motion).
3. Other managed platforms and Virtual Machines (VM), such as Java/J2ME/JME (which stands for Java 2 Micro Edition), BREW (Binary Runtime Environment for Wireless), Flash Lite and Silverlight).

There are plenty of different operating systems for mobile devices; the most important ones are listed as follows:

A. Symbian OS (Symbian, 1998):

Nokia, Samsung, Sony Eriksson, Motorola, Samsung, Panasonic, FOMA (Freedom of Mobile Multimedia Access).

Used mostly for smartphones, this proprietary operating system has its own SDK (Symbian OS Software Development Kit), which is based on C++ language and supports runtime environments such as Java, Python and others. It also supports different runtimes, included Java ME (Symbian, Runtime Environments, 2008).

B. iPhone OS (Apple, 2008):

Developed for the iPhone and iPod Touch, it uses its own development kit, called iPhone SDK, providing a specific development environment known as XCode. Cocoa is its programming environment, which uses Objective-C language. It does not support Java VM or Flash, since it uses SVG.

C. Blackberry:

Research In Motion company (RIM, 1984) developed the first Blackberry in 1999. Blackberries are characterized by having a built-in QWERTY keyboard, optimized by the use of thumbs. It uses a modified Java VM.

D. Windows Mobile

Microsoft also has its mobile version of the Windows operating system, which combines a suite of basic applications for mobile devices based on the Microsoft Win32 API (Application Programming Interface). Devices that run Windows Mobile include Pocket PCs, Smartphones, Portable Media Centers, and on-board computers for certain automobiles.

Thanks to ArcPad and ArcGIS WebServer for Windows Mobile, developers that use ArcGIS software of ESRI can easily benefit from these frameworks to extend their own GIS applications (ArcGIS Mobile, 2008)

E. Linux/Unix

Some mobile companies also use this open source operating system for some mobile devices, Google, NEC (Nippon Electric Company), Motorola, Panasonic, and PalPalm among others. This is a generic term used for operating systems that are based on the Linux kernel and therefore Unix-like. Its main characteristic relies on its open source, since it can be freely modified.

Therefore, it has been a big competitor to Symbian OS for smartphones and a real alternative to the proprietary Windows CE and PalmOS for other mobile devices. Some of the operating systems for mobile devices based on a Linux kernel are:

- **Android:**

Android (Google Android, 2008) is the new operating system of Google for mobile Phones. Its kernel is also Linux and on the top of that that some C++ libraries have been added, but mostly Java code, ending up with a complete Java SDK for Android developers.

- **Openmoko:**

The Openmoko project (Openmoko, 2006) has the peculiarity of being divided into two subprojects: one for the open source software and the other for the open source hardware. Hardware as an open source means free release of hardware information in the form of schematics or even of the Hardware Description Language code (HDL) used. It can also contain information about the Bill of Materials (BOM) needed for manufacturing the product and also about the Printed Circuit Board (PCB) layout data. Even more, due to the fact that both operating systems share the same kernel, it has been possible to apply portability to run the Android platform on an Openmoko device.

gvSIGDroid, our current open source project, has been entirely implemented in Java for the Android platform.

3.3. Open source vs. Proprietary source

For smaller companies and projects, open source solutions are usually enough. On the other hand, larger companies often require something more enterprise-ready. This means they need a robust, high-quality product with high levels of both service and support, as well as reliable assistance and support for their projects.

The table below stresses some of the advantages and disadvantages of both proprietary and open source systems (based on Van der Vliet, 2007):

	Private/proprietary source	Open source
Advantages:		
	• Reliable, professional support and training	• Low cost: no license fees
	• Out-of-the-box, comprehensive, modular	• Freedom: open standards facilitate integration
	• Regularly and easily updated	• Customizable
	• Good for large enterprises, bad for small ones	• Especially good for small enterprises and developing countries
Disadvantages:		
	• Expensive	• Lack of professional support or documentation
	• Closed standards, difficult integration with other systems	• Unstable developer communities
		• Lack of release co-ordination
		• Lots of license types that may confuse users about their rights
		• Erratic updates

Table 1. Advantages and disadvantages between proprietary and open source (Van der Vliet, 2007)

Regarding **open source licenses** used in open source projects (see Appendix IV), it is important to really distinguish between *non-permissive* and *permissive* licenses. The first one is represented by the GNU/GPL license, and is mainly characterized by being a *strong-copyleft* license in which all of the source parts of the project (e.g. code, own libraries, third-party libraries) have to encompass one or many open source licenses. The latter one is represented by the GNU/LGPL license, and it is not that strict. It allows third-party code to also encompass libraries from private companies, in which that piece of code is not shared. LGPL can always de changed to GPL since the latter is more restrictive, but not vice-versa.

So far, Android makes use of *Apache license* (AndroidC, 2008), which is permissive. This means that it is not compulsory for private enterprises to share their code when publishing an android project. With this license, anyone can customize the android platform and then keep it as an open source or proprietary, as the source does not have to be published.

3.4. Open Source applications

A. Location-Based Services (GPS):

A.1. BlueMapia[1]:

It's a web and mobile application (Windows Mobile) with social, multi-map GPS functionalities to share videos, photos, comments and maps about places that can be visited by boat. It supports Google, Microsoft, Open Street Map, NOAA/BSB charts and self-calibrated raster.

[1] http://www.bluemapia.com/

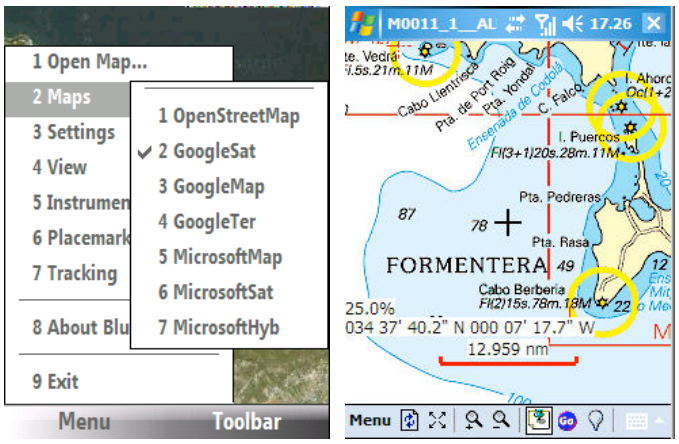


Figure 3. Bluemapia software.

A.2. Gpsd[2]:

This back end daemon reads data from a NMEA or binary protocol GPS and multicasts this information on a TCP port for a variety of client applications. Included in the package is a sample client called *xgps*. It connects to a gpsd at any host inputted, and requests raw data to display the current location of all visible GPS satellites with the current receiver *sees*.

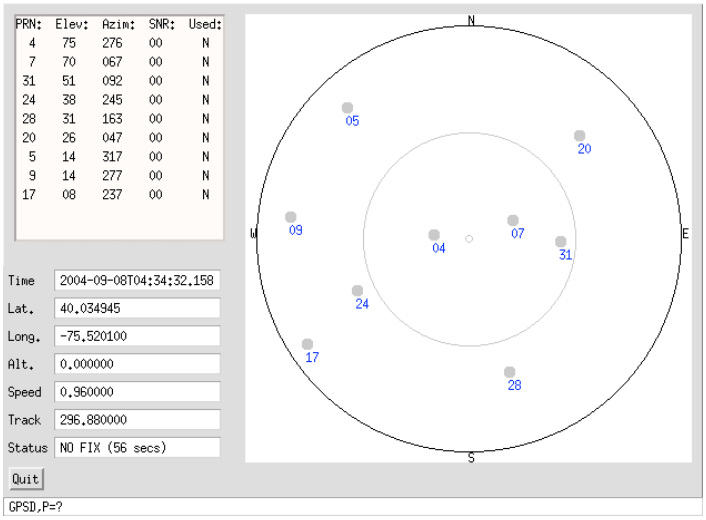


Figure 4. Xgps using Gpsd daemon.

[2] http://gpsd.berlios.de/

A.3. GpsDrive[3]:

GpsDrive is a real-time car (bike, ship, plane) navigation system that displays positions provided from a given GPS receiver on a zoomable map. These maps are autoselected for best resolution depending on current position and can be downloaded from the Internet. It can also allow speech output.

GpsDrive is written in C with use of the GTK+ toolkit under the GPL license, and runs with Linux, Mac OSX, and FreeBSD. It supports the creation of waypoints and tracks in a SQLite DB, and import/export of GPX files. OpenStreetMap data can automatically be rendered into map, and NASA Landsat data can automatically be downloaded for the current position.

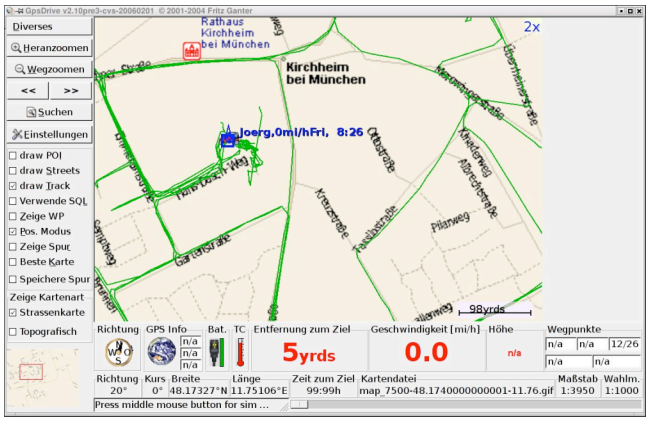


Figure 5. GpsDrive software

A.3. OSMtracker:

The *OpenStreetmap Mobile Tracker* generates tracks and waypoints (NMEA raw and GPX formats), and also offers audio recording. This application is designed for the Windows Mobile PDA/Pocket PC.

[3] http://www.gpsdrive.de/

B. Location-Based Services (WebGIS and WMS Viewers):

B.1. GRASS Server:

WebGIS service enables not only interactive viewing and querying, but also GIS analysis. It offers speech recognition support and provides access to geographic information from different kinds of devices like PCs, PDAs and mobile phones.

C. Location-Based Services (GPS and WebGIS):

C.1. TangoGPS[4]:

TangoPS is an easy to use, fast and lightweight mapping application that runs on any Linux platform from the desktop over *eeePC* down to phones like the *Openmoko Neo*. By default *TangoGPS* uses map data from the *OpenStreetmap* project. Additionally a variety of other repositories can be easily added.

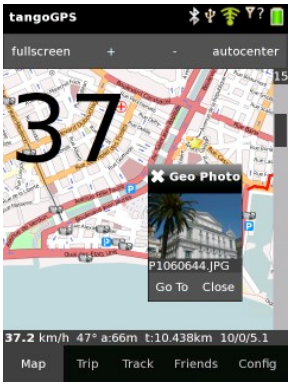


Figure 6. TangoGPS software.

D. Field-based applications:

Unfortunately, the only software applications for mobile phones that enable the edition of points, lines, polygons and the change of their attributes are available mostly

[4] **http://www.tangogps.org/gps/cat/About**

on proprietary format. In terms of open source: *gvSIGDroid* using Java and *Enebro* using C++ are the only options.

D.1. Enebro

Enebro is a field-based application that enables spatial data gathering on mobile devices. It is possible to visualize and edit vectorial and raster layers, and navigate with the available cartography with the help of GPS. It is written in C++ and uses European Unit Public License.

3.5. The Android Platform

The Android platform (AndroidB, 2008) is not only a framework, but also consists of a complete, open source operating system developed by Google for mobile devices. As previously mentioned, it is based on a Linux kernel upon which several tiers have been developed, and can be identified in the following architecture:

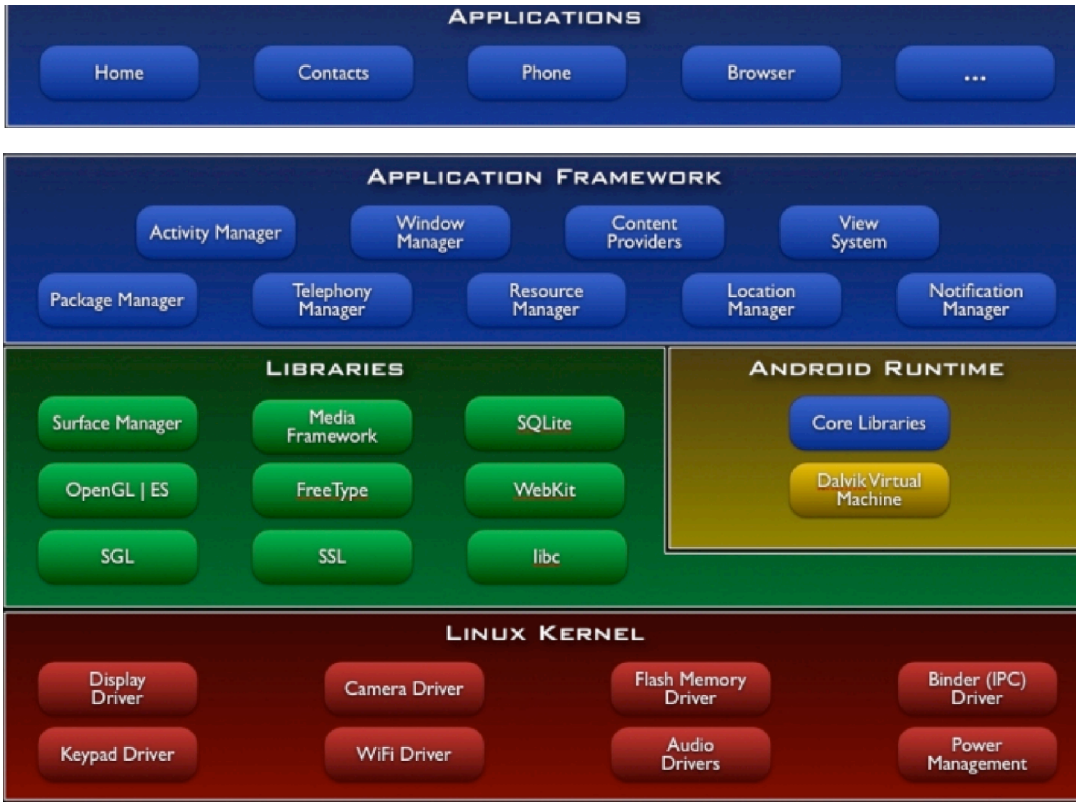


Figure 7. Android Architecture

Based on the structure above, some libraries that are written in Java (blue colored) can be distinguished. These libraries are distributed between the application framework, the developers-implemented applications and the core libraries. The latter ones also have functions that enable accessing both the libraries (C++, in green) and the Linux kernel (Unix, in red) tiers. All this java code encompasses the framework API that the developer uses when implementing an Android application. Finally, the implemented code is transformed into *.dex* files (Dalvik EXecutable files, in yellow), which are interpreted and executed by the Dalvik Virtual Machine (VM).

Hence, developers have access to the same framework APIs used by the core applications. In order for gvSIGDroid to work on the Android platform, only Java code will be implemented by using the API provided by the Android SDK at an application tier level.

So, what is important to know about Android? At a very basic level, Android is just about managing different components. An Android APK (Android PacKage) consists of a collection of components. Actually, the *.apk* file is just a zipped file that is created and installed in the device, and acts as a *.jar* file in java. To understand the main components of this zipped file, Android framework can be seen as a big *"sea of components"* in which these APK would be the "islands". So, each component on this *"island"* (APK) cannot communicate to other components of other islands, but it looks as if they can.

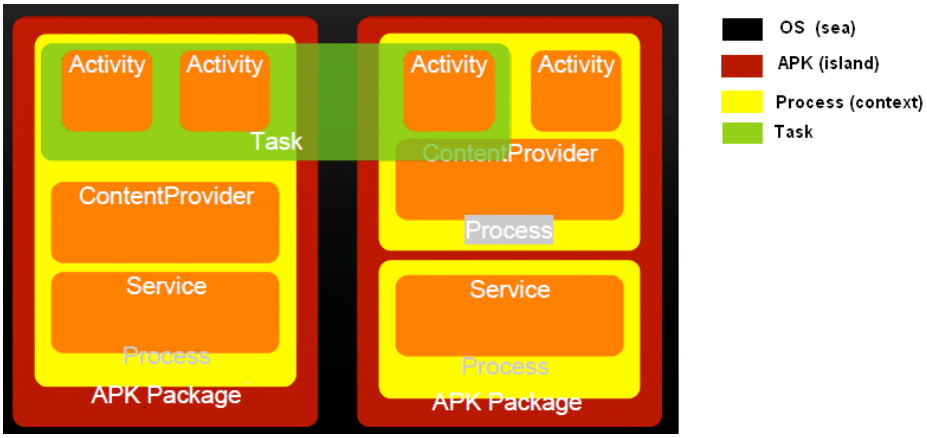


Figure 8. APK general structure.

As shown in Figure 8, the main components are: activities, content providers, services and tasks. Each APK is associated with a processor thread (by default 1, see left APK package) but it is also possible to execute the components on a different process (see right APK package).

- **Activity:**

An "Activity" belongs to a specific class in the java API (*java.app.Activity*). It is called "activity" and not "application" because they do not need to be related to a User Interface (UI), they can just "do something" since they are an execution context on their own.

- **Task:**

A "Task" is a collection of activities that are not that well defined. It is more of a "notion" than a concrete entity of the API. It is just a runtime record of a sequence of activities, a collection of related activities. Each task is what the user would consider as an *application*. For developers coming from a desktop metaphor, they may think of clicking a button and starting a program. In Android, activities are shared, and are more distributed, so that an application can be built and activities of other APKs can be used, taking into account that the activities that don't belong to the APK will run on a different process, in the context of the APK which installed them.

- **Process:**

About "Processes", they simply consist of Linux processes, with no additional semantics. By default, each APK is related to a Linux process and one thread per process, so (almost) all the APK components run within the same thread.

Any process in Android is started whenever it is required, that is, any time the user or some other system component (activity, content providers or services) requests the code for the APK to be executed, the system will run the process needed if it is not running

yet, and will run it until it is automatically killed when needed. The system will continue running the process even if all components in the application are concluded and shut down (not too long, but has this option).

In most of the cases, Android developers will be mainly developing activities and tasks (sequence of activities). These activities have a managed activity life cycle, which developers should be aware of (AndroidA, 2008).

In gvSIGDroid, in order to implement the Graphical User Interface (GUI) and execute its correspondent actions, it was crucial to deal with different activities and get them to communicate with each other. Activities always have a View associated, which can be created programmatically or by XML (extended Markup Language) inflation. Therefore, a thorough understanding of the different activity and view types, intents and resource managements was needed for successfully implementing the solution.

3.6. gvSIG

gvSIG is a desktop Geographic Information System (GIS) that enables capturing, storing, handling, analyzing and deploying referenced GI to solve complex management and planning problems.

Its main characteristics rely on:

- Open source software, under GNU/GPL license, which allows its free use, distribution, study and improvement.
- Developed using Java, is available for Linux, Windows and Mac OS X
- Easily extendable, allowing:
    - Continuous application enhancement
    - Tailor-made solutions.
- Integration of local and remote data through Open Geospatial Consortium (OGC) standards in the same view.
- Support of most of common vector and raster formats

- Wide range of geographic tools (query, geoprocessing, networks, etc.)
- In several languages: (Spanish, English, German, French, Italian ...)
- Invested development areas: Metadata, 3D Visualization, Time representation, among others.

3.6.1. gvSIG general Architecture

As can be seen in Figure 9, gvSIG's architecture can be divided into the following components:

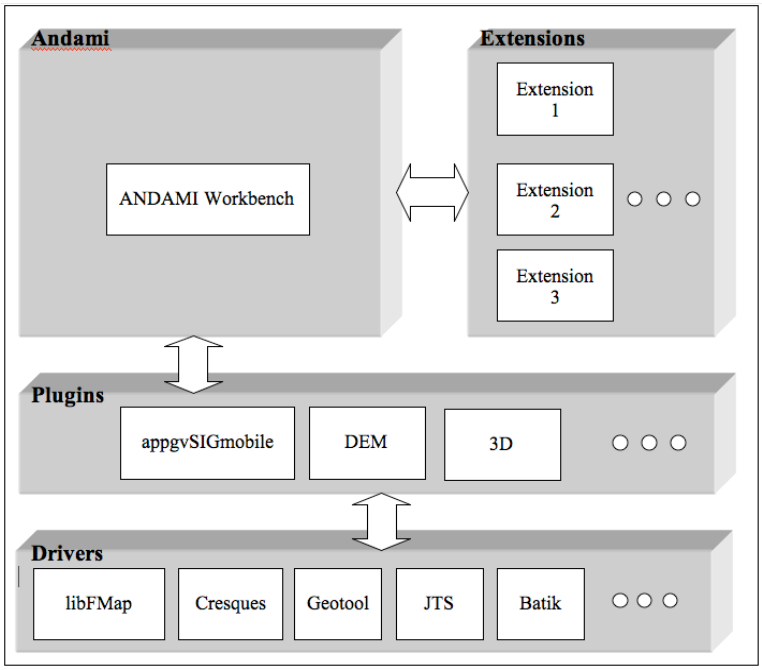


Figure 9. gvSIG Architecture.

1. ANDAMI:

Base Application is extensible by means of *plugins*. It is responsible for the creation of windows, loading and managing the extensions, selecting the suitable *look & feel*, enabling the beginning of the application by means of *Java Web Start*, initializing the

language of the application, etc. This application is completely generic, and it serves as a seed for any MDI (Multiple Document Interface) application to be created.

2. FMAP:

Library of classes that allows creating custom GIS applications. It includes an internal *core* with the low level objects necessary for its operation like modified JTS (Java Topology Suite) and Java2D entities, and the appropriate conversors and a number of objects to work with those entities. Within this library we find classes to read and write the supported formats, draw maps at the adequate scales, assign legends, define symbols, perform searches, queries, analysis, etc. The "drivers" (read/write) of formats are included in this section.

3. GUI:

This module consists of extensions to the base application that can interact with the user. In this library of classes we mostly find dialogue boxes used in the final application, as well as the classes that support those dialogue boxes (e.g. forms to assign legends, create maps, define scales, etc).

3.6.2. gvSIG Mobile

gvSIG Mobile has migrated the gvSIG Desktop to the mobile context. In this way some code has been rejected or shortened in order to contain just the main basic functionalities and for reusing code. Since no extensions are possible in gvSIG mobile, the whole Andami module has been rejected, and appGvSIG was renamed to appGvsigMobile, including just very few classes from Andami needed for launching the application into the environment.

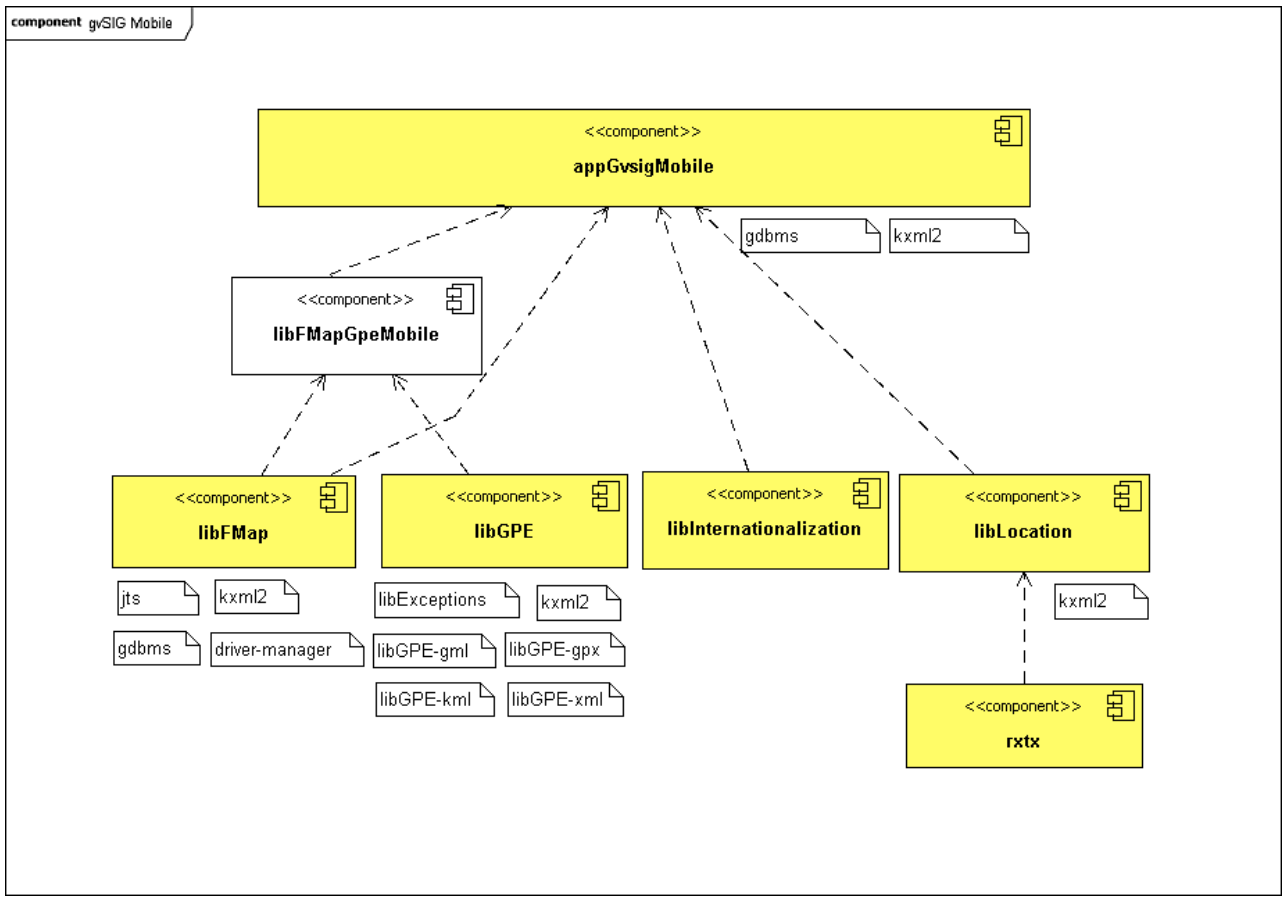


Figure 10. gvSIGMobile Architecture

The main differences between gvSIG and gvSIGMobile are:

	gvSIG Desktop	**gvSIGMobile**
Andami module	Used for managing plugins and extensions	No extensions, so no Andami
libFMap library	Complete	Reduced
Views per project	One project, many views	Only one project and one view at a time
GUI	Use of swing library (based on Java.awt)	Use of Java.awt

Table 2. Main differences between gvSIG and gvSIGMobile

4. gvSIGDroid

4.1. Chapter Objectives

gvSIGDroid is the result of migrating gvSIGMobile application into the Android platform. For accomplishing this endeavor, some previous steps before the implementation phase were needed, which are stressed in the following sections. In section 4.2, a set of requirements that the application needs to offer are listed, to get an initial idea of the future functionalities the application will have. After that, in section 4.3, a use case scenario was drawn in order to visualize the main core modules of the application. Then, in section 4.4, some preliminary sketches were drawn (see also appendix III) to get an overall insight of the final result and concretize what the application should be able to do and how should it look like. In section 4.5, the final architecture of gvSIGDroid was explained so as to understand the design and implementation phases that follow in section 4.6. Finally, some internationalization issues regarding the Android platform and gvSIGMobile were emphasized.

4.2. Requirements analysis

The main purpose of gvSIGDroid is to exploit Android SDK to provide a user-friendlier application, which results more in keeping with the public's expectation. Due to the five month time constraint, some functionalities of gvSIGMobile will not be available on the first prototype of gvSIGDroid, e.g. edition capabilities and GPS positioning. Therefore, the initial requirements settled for the application to be fulfilled are listed as follows:

Graphic view:

- Add and delete layers.
- Enable navigation options (zoom/pan modes).
- Symbology options.
- Opening and saving projects

Supported file formats (see acronyms section):

- .SHP, .KML/.KMZ, .GPX / .GML
- .PNG / .JPG / .JPEG

Web services (OGC Standards):

- WMS Service (Web Mapping Service).

Tools:

- Information (“Identify”)
- Measuring areas and surfaces.
- Selection tools.

4.3. Use Case Diagram

The following figure shows the general use case of gvSIGDroid application:

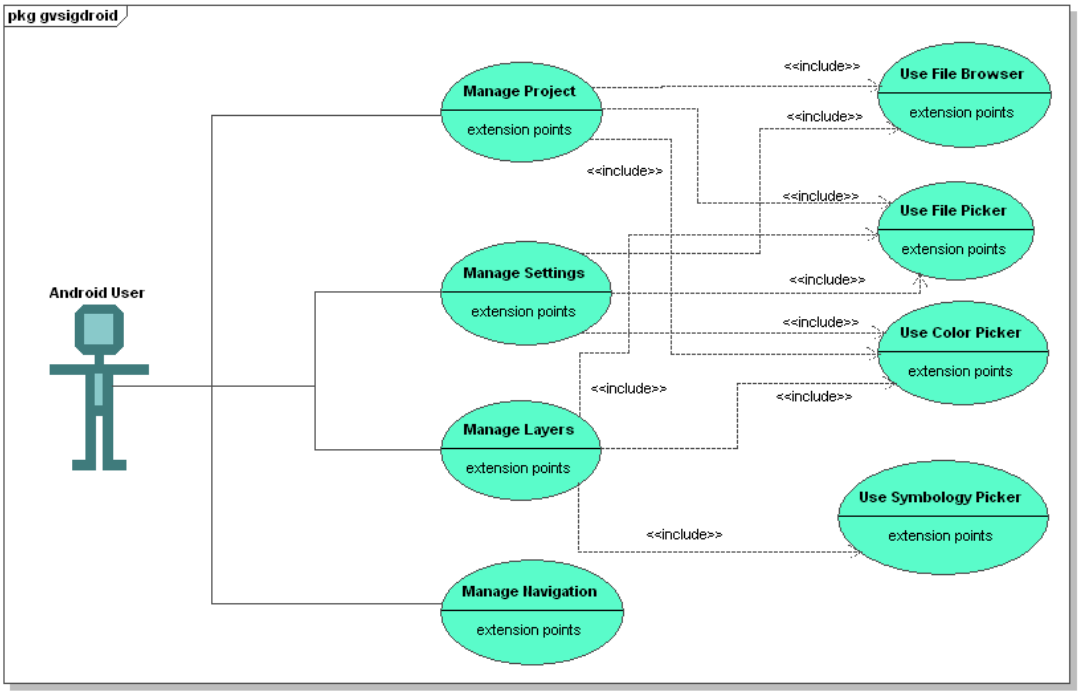


Figure 11. gvSIGDroid's Use Case Diagram

Here, the structure of the program has been divided in four main functionalities:

A. Manage project:

This module allows creating a new *.gvm* project based on the default parameters, or loading a previously stored one, saving the current project and looking at its properties. For this, a filtered file picker for selecting a *.gvm* file, a color picker for changing the background color as well as a file browser for selecting the final path to save the project are needed.

B. Manage settings:

gvSIGDroid also provides a module that enables users to change some default parameters such as language, default *.gvm* path, default SRS, default background color, and so on. Therefore, a file browser for selecting the root path, a color picker

for choosing a default background color and a file picker for selecting a default *.gvm* project to be loaded at the initialization, are needed.

C. Manage layers:

Logically, gvSIGDroid requires a module that manages the various loading options of layers to a project. Therefore, a filtered file picker is needed to select between different types of layers, which are stored locally on the device, as well as WMS layers. Once the layer is added, users can change its symbology, for which a symbology picker with a color picker in it is needed.

D. Manage navigation:

Users also need to navigate through the layer, being able to perform some actions on it, like zooming in, zooming out, re-centering the layer by point, measuring paths and areas, and so on. For this no pickers nor browsers are needed, just a context dialogue to let the user select between these different options.

4.4. Preliminary sketches

Being aware that the implemented GUI should be user-friendly to Android users, it was imperative to explore already installed applications on the device before sketching (see appendix III), which led to significant and helpful observations during the whole design process.

For example, in applications like *GMail* or *Google Maps*, options were shown when clicking the menu button rather than showing a set of buttons on the screen, and also options to be applied on an item in a list were shown when long-clicking that item. Also, in order to get to the previous screen, as well as for exiting the application, no extra button was needed since both situations were already handled by the system via the "back" button,

Then, some sketches were drawn on behalf of those explored interfaces so that the results are similar. The first depicted sketch was a hierarchy of complex layouts extended from the android *LinearLayout* class, in order to reuse code.

As can be observed in figure 12, from top to bottom, we have the most complex view types which are BaseListView and BaseTabView, which add a footer to the view with the Ok and Cancel buttons. In the TabView, the elements that are contained within them are SingleViews. Both OkCancelView and SingleView share its superclass, which is an abstract View that manages the right allocation of the inner layout at any case.

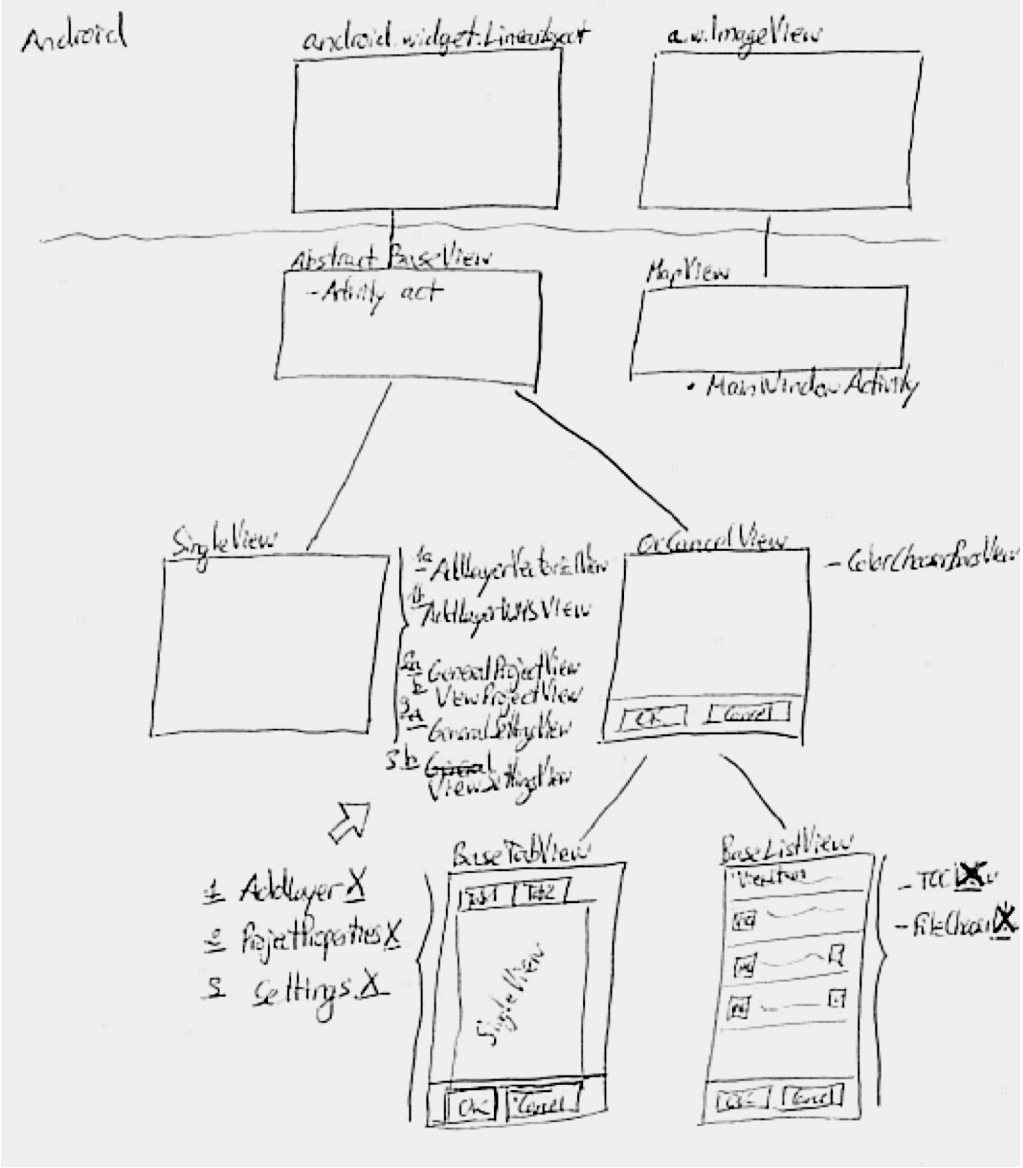


Figure 12. Layout hierarchy sketched

Once aware of the different possible layouts, the whole interface set was sketched, distinguishing the different modules of the use case, as can be seen in Appendix III.

4.5. gvSIGDroid Architecture

In gvSIGDroid, both Android graphic utilities for the presentation layer and gvSIGDroid inner model have to be used, among other resources. For this, a loosely coupled architecture has been designed based on the *View-Control-Model* paradigm (VCM) (see figure 13).

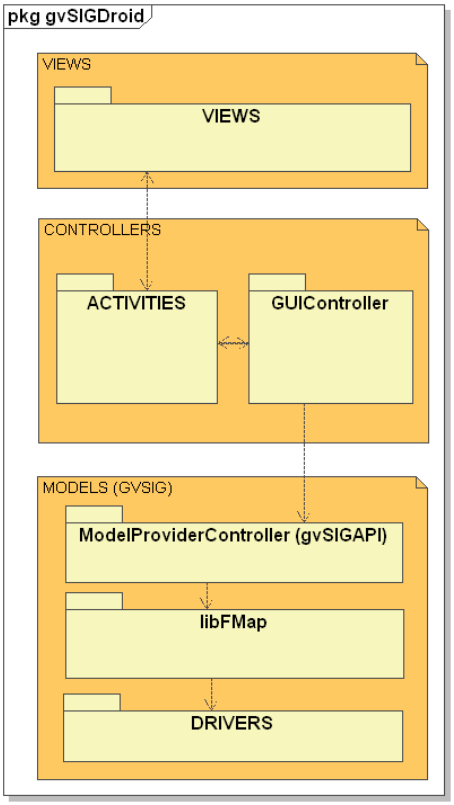


Figure 13. gvSIGDroid's Architecture.

In this architecture, the following modules can be distinguished:

A. Views:

Java class used as base for the graphical objects in the Android platform; all graphical objects extend from this class. Therefore, methods for inputting information as well as drawing capabilities are offered by the system.

B. Activities:

Activities are the core view controllers in the Android platform. They are in charge of handling graphical events like *onClick()* for images and buttons or *onListItemLongClick()* for items of a *ListView*. Apart from this, activities also handle system events, like the *onPause()* event fired when another application of the system is going to gain the focus (an incoming call for example). All these special cases must be correctly handled on each module.

C. Other controllers:

Apart from the events mentioned on the previous section, there are several actions that need to be handled on a View, which are:

1) Calling other activities: Showing a new screen implies calling to a new activity, which will be handled by the *GUIController* class, which still handles GUI information.

2) To provide the information needed from the internal gvSIG data model, which should not require any graphical control, a different controller will be used. This controller acts as a model provider, as an API of gvSIG, being an abstraction of the current model arquitecture of gvSIGMobile (*libFMap*) to ease the retrieval and storing of information to this module. This is managed by the *ModelProviderController* class, which will link to the *libFMap* module of gvSIG.

Both controllers must be somehow connected to the Activity class related to the view. Generally speaking, a *GUIController* will not only manage the triggering of new activities, but will also extend from a *ModelProviderController*, acquiring all its methods for accessing the model. An activity just has to contain a *GUIController* instance to access the gvSIG model and retrieve the information in the form of a GUI.

Distinguishing between graphical controllers and model controllers, it is possible to apply the bottom part of this architecture, starting from the *ModelProviderController* to other GUI frameworks and share code among applications.

The main differences between *gvSIGMobile* and *gvSIGDroid* architectures are:

	gvSIGMobile	**gvSIGDroid**
GUI	Use of *java.awt.** library	Use of Android views
VMC	Views and controllers difficult to be distinguished	Activities as controllers, View classes as views.
Internationalization	Use of *i18n* library	Managed by Android

Table 3. Main differences between gvSIGMobile and gvSIGDroid

4.6. Design and Implementation

Based on the use-case scenario, the following modules are listed:

4.6.1. Project module

This module aims to manage all possible actions that a *gvm* project (gvSIG project) can do, which are: *new project, open project, save project* and *save project as*. At the model level, an abstract *ProjectManager* class has been implemented with the *doNewProject(), doOpenProject()* and *doSaveProject()* methods. Then *DroidProjectManager* has inherited this class and developed its functions *showProjectProperties()*, *newProject(), saveProject()* and *saveProjectAs()*. Each of these functions will be explained next.

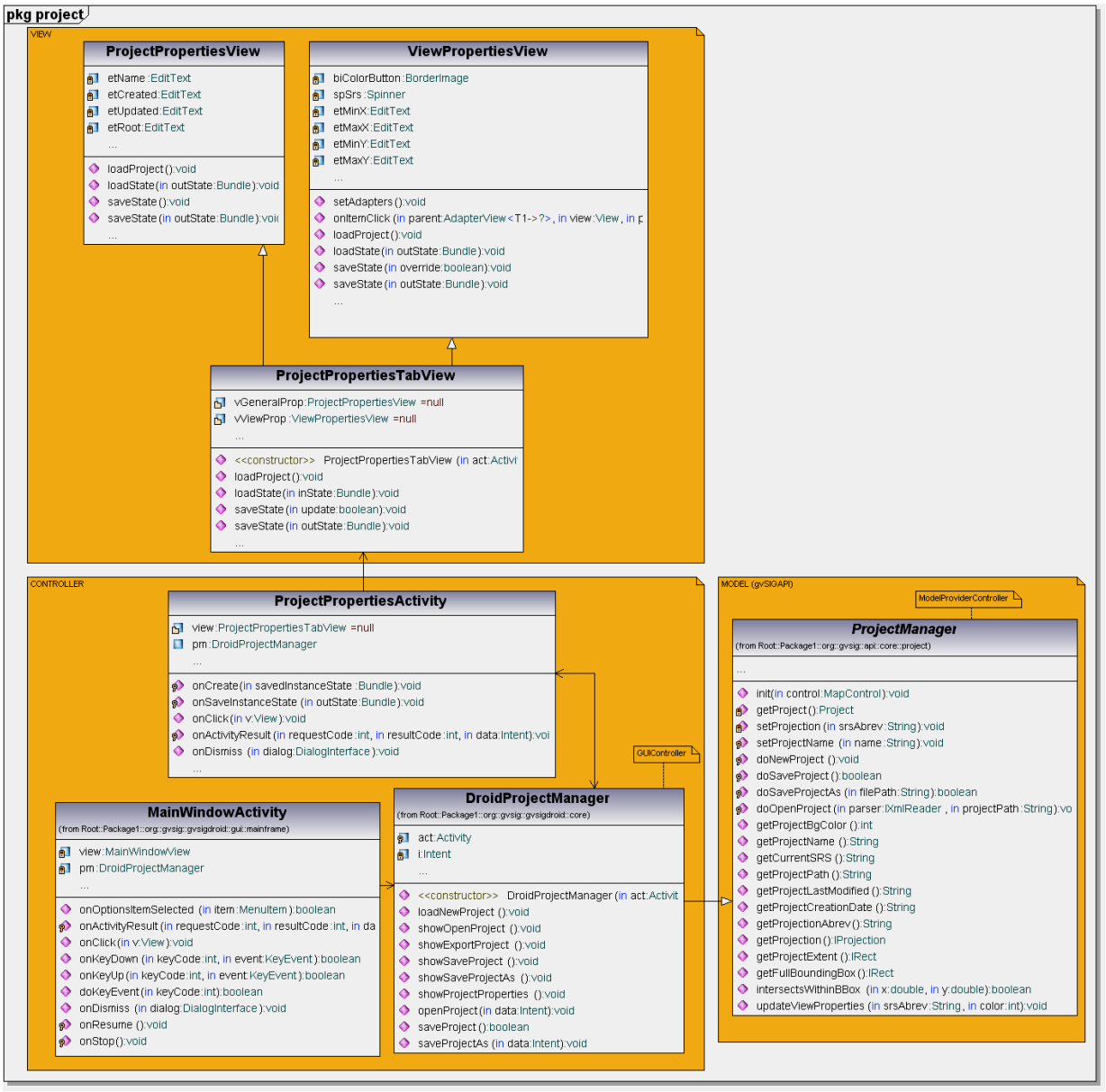
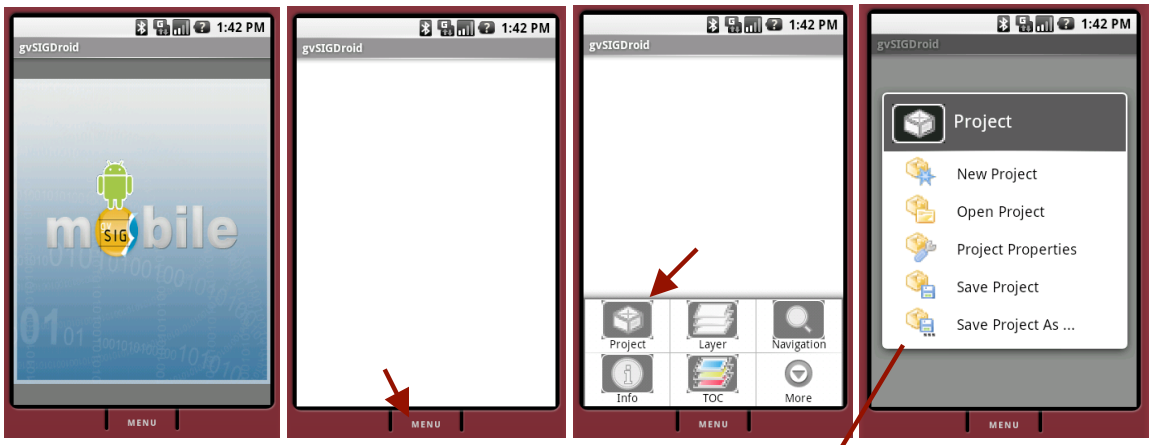


Figure 14. gvSIGDroid's class diagram, Project module.

A. New project:

When initializing gvSIGDroid, a *MainWindowActivity* with its associated *MainView* is launched. This *MainView* shows an initial logo as depected in figure 15.a. In the meantimee *loadNewProject()* method of *DroidPojectManager* is being launched. Then, a new project with a default background color, default Spatial Reference System and bounding box parameters is shown (15.b). When the user presses the menu button,

different options appear, among them the project button (15.c). When pressed, a context dialogue appears showing the project options (15.d and 16).



(a) (b) (c) (d)

Figure 15. Initial gvSIGDroid's views: logo view (a), blank view (b), after menu button pressed (c) and after project menu pressed (d).

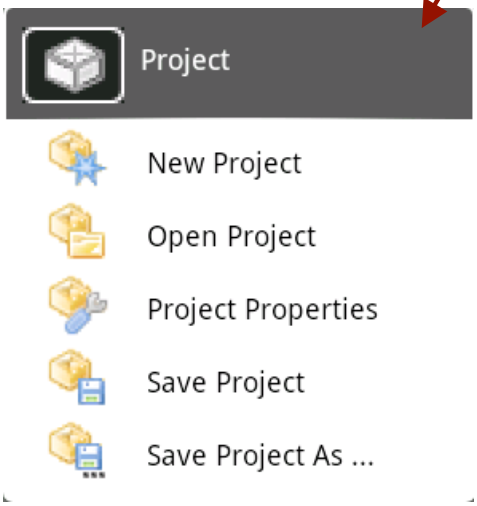


Figure 16. gvSIGDroid's Project Menu

B. Open project:

When selecting the option “open project”, *MainWindowActivity* calls *showOpenProject()* method of *DroidProjectManager* controller, since a new GUI is needed for asking the project to be launched. Thus, the *FilePickerActivity* is launched, being able to select from the existing *.gvm* files of the *sdcard*. When selected, the method *onActivityResult()* of MainWindowActivity receives the file path and executes the *doOpenProject()* method of

DroidProjectManager. This proceeding has been done, for example, in a project that contains several shapefiles from the city of Valencia, Spain (figure 17):

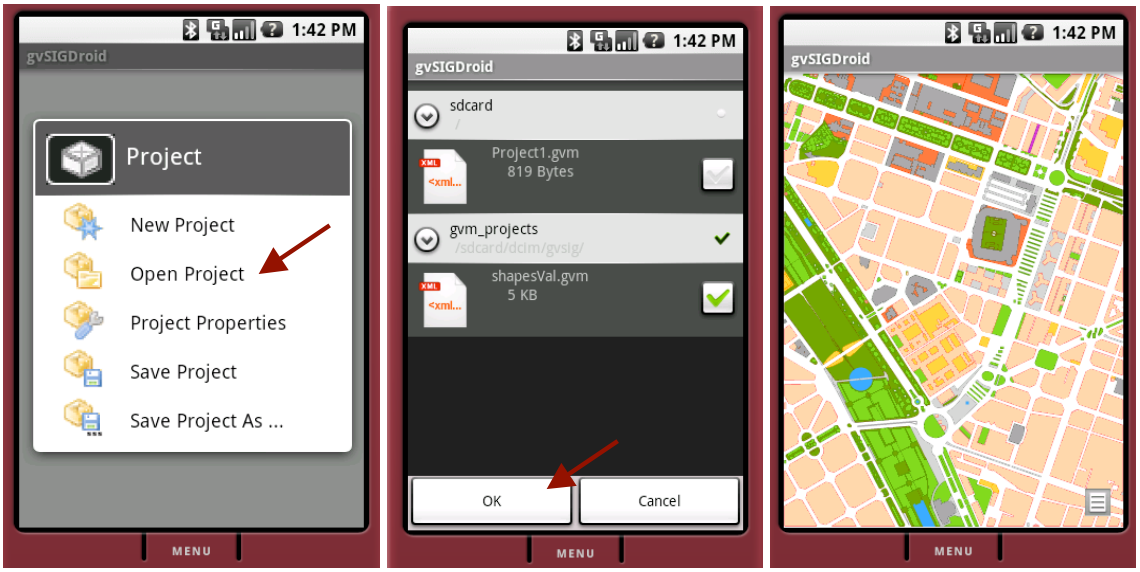


Figure 17. Open Project process.

C. Save project / Save project as:

When selecting the option "save project", *MainWindowActivity* simply calls saveProject() and the *gvm* file is updated with the current properties. If "save project as" is selected, FileBrowserActivity is launched to select the desired folder to be saved at and the chosen name of the *gvm* file (figure 18).

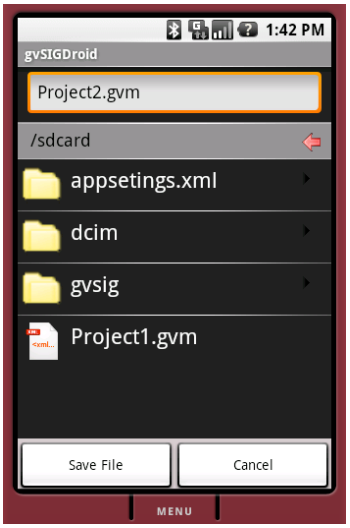
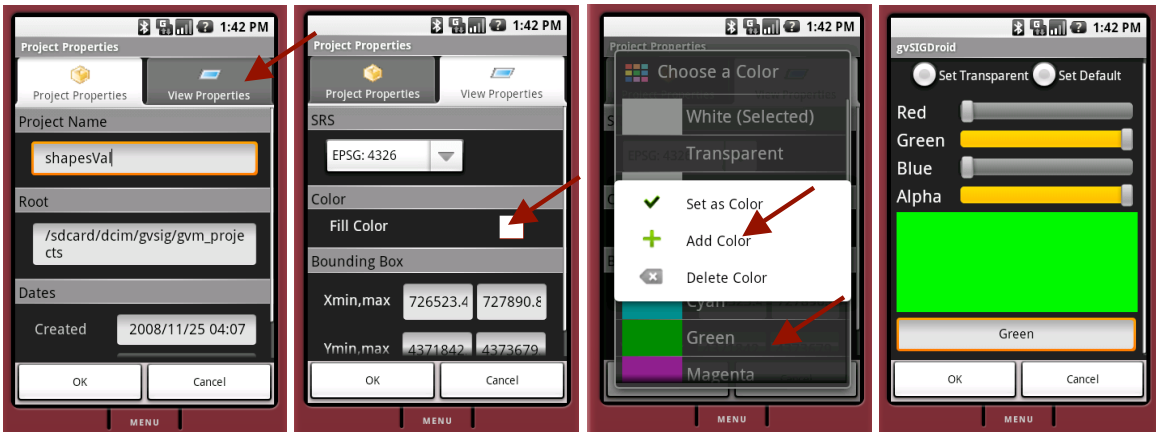


Figure 18. File browser for saving a gvm file.

D. Project Properties

The properties of the current project can be shown. When *ProjectPropertiesActivity* is launched, it uses a *ProjectTabView* containing two different views: data about the project and data about the view. This GUI information is loaded and stored via the getters and setters methods of *DroidProjectManager*,; all of them are extended from the *gvSIGAPI ProjectManager* abstract class. These properties can be shown in figure 19.a and b. The project's background color can be changed by use of the *ColorPickerActivity*. (figure 19.c and d).



(a) (b) (c) (d)

Figure 19.gvSIGDroid Project Properties.

4.6.2. Settings module:

The settings module is divided into three main sections: general settings, project settings and view settings (figure 20). The first one uses specific Android views and activities implemented for settings options. So, *GeneralSettingsActivity* extends from *PreferenceActivity* and contains a *PreferenceScreen* view. The other two use the general gvSIGDroid architecture, with activities, GUI controllers and *ModelProvider* controllers. All these activities use *DroidSettingsController* to gather default values of the project.

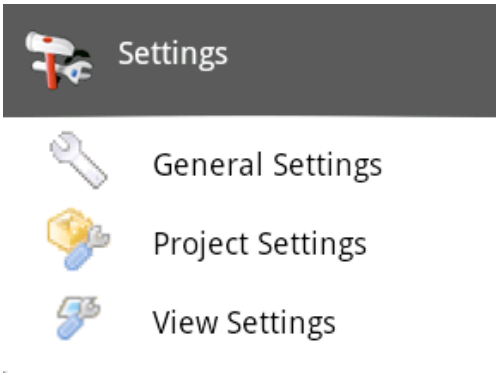


Figure 20. gvSIGDroid's Settings Menu

Therefore, the settings model has been designed in two directions: one Android oriented ("General settings" option), and another, more gvSIG oriented, in which settings are saved on a temporal *appsettings.xml* file, containing values such as default bounding box, default background color, default root file, and so on. The reason for different approaches is that at the moment, Android's preference views do not provide more sophisticated views – for example for handling colors- and also because the distinction between project settings, view settings and other general settings are easy for the user to understand.

When looking at figure 21, we can see that *DroidSettingsManager* uses *Appsettings* to gather the information from the xml file into a *TreeMap* java object, being parsed and then stored back when any change occurs with the aid of the *XmlUtils* class. Once the *TreeMap* is formed, Appsettings has methods to gather the information required, which is used by *DroidSettingManager* for providing the information to the setting's view.

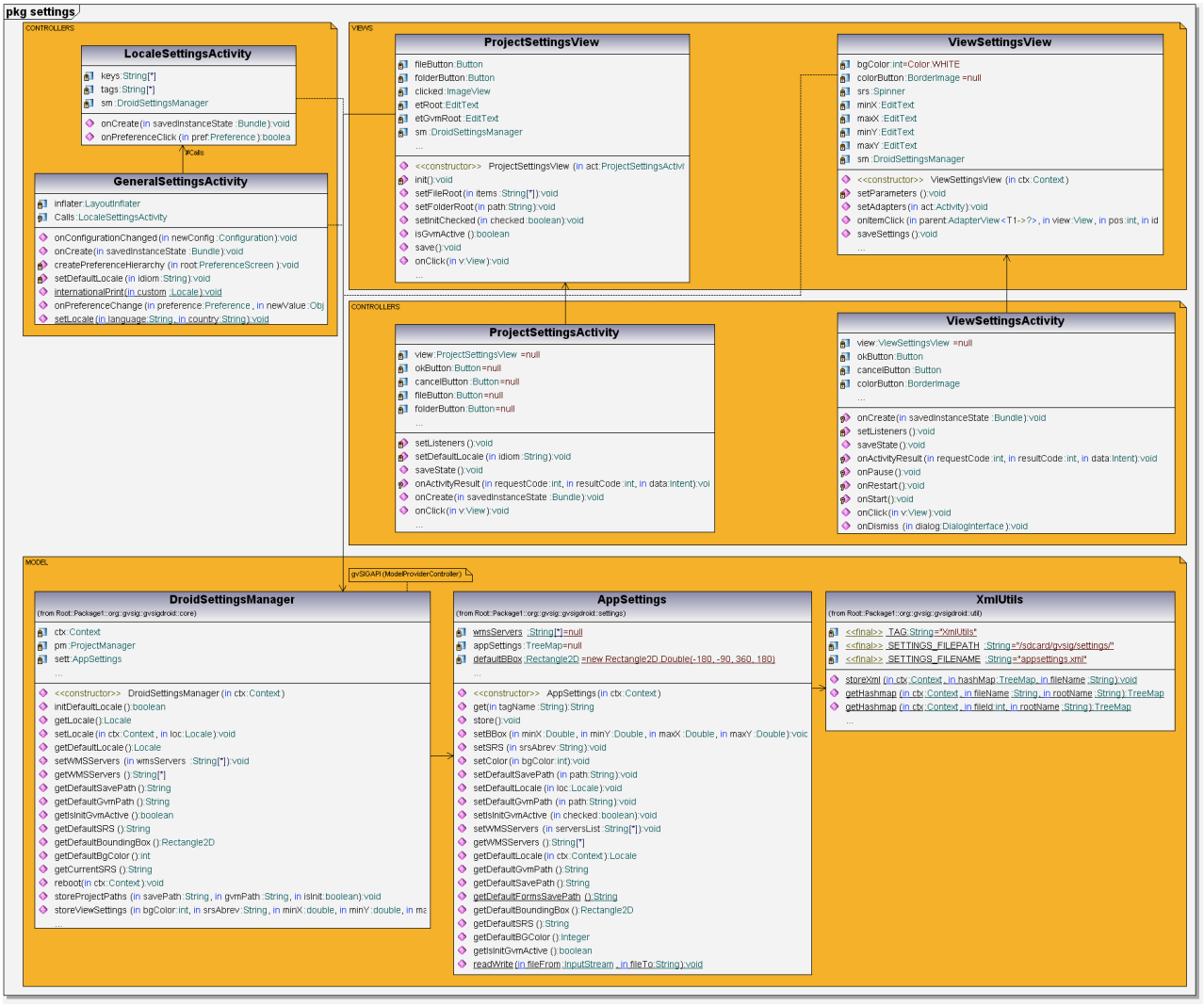


Figure 21. gvSIGDroid's Settings class diagram.

A. General Settings:

General settings are used for handling any kind of options the user wants to customize that are not related to the project nor the view of the project. As of now, the only option available is the language setting.

Since the change of *Locale* requires the program to reboot, all gvSIGDroid activities are closed once the user chooses a different language than the current one. For example, if French is selected, the main logo view will appear again and when pressing the menu button, options will appear on the selected language (figure 22). This is all handled by the *DroidSettingsManager* controller with the *setLocale()* method.

Figure 22. Changing language settings in gvSIGDroid.

B. Project Settings:

For project settings, only two options are possible: setting the default root from which file browsers will start, and setting a *gvm* file to be initialized each time the application re-launches. This is previously disabled. For the first option, a file browser is needed to set the path, and for the latter one a file picker to set the *gvm* file. These changes are stored into the *appsettings.xml* file and into the *TreeMap* object once the "ok" button is pressed.

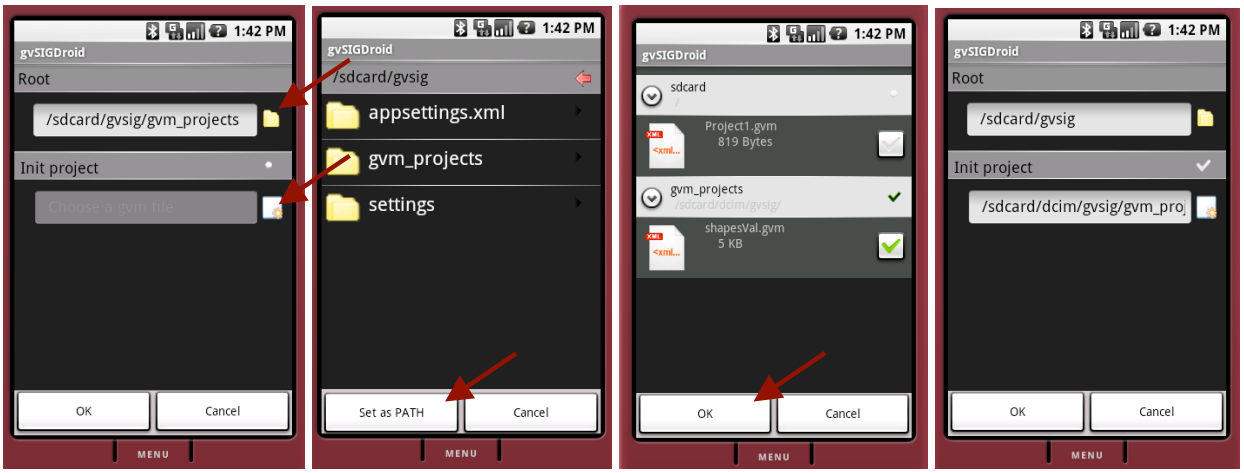


Figure 23. Project settings options.

C. View Settings:

The view settings option works identically as the View properties tab. The only difference is, that instead of saving the options into the current project properties, they are stored back to the *appsettings.xml* file and the *TreeMap* object (figure 24).

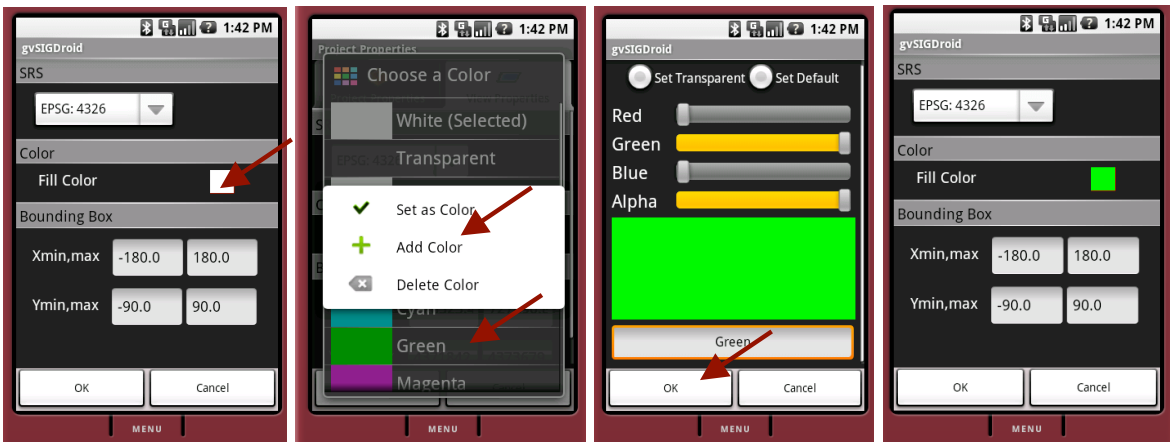


Figure 24. View settings options.

4.6.3. Layer module

This module aims to manage the layers containing the project. Layers can be added to the project, and can have several formats: vectorial (*shapefiles, kml/kmz, gml, gpx*), images (*jpeg, jpg, png*) and WMS (generally *png/jpg* format). The first two are local files stored at the *sdcard*, but the last one remotely accesses a Map Server to gather the information. Once added, we can visualize their order and the symbology used (*TOCActivity*). Also, it is possible to take a snapshot of the view as well.

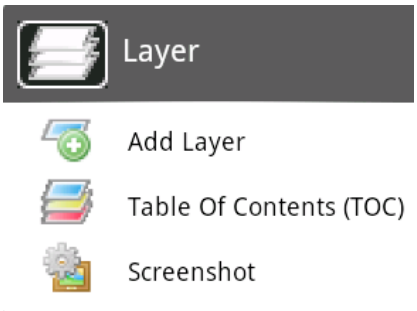


Figure 25. gvSIGDroid's layer menu.

A. Add Local Layer:

The overall general diagram for adding a layer is similar to the one used by project properties, as can be seen in figure 26:

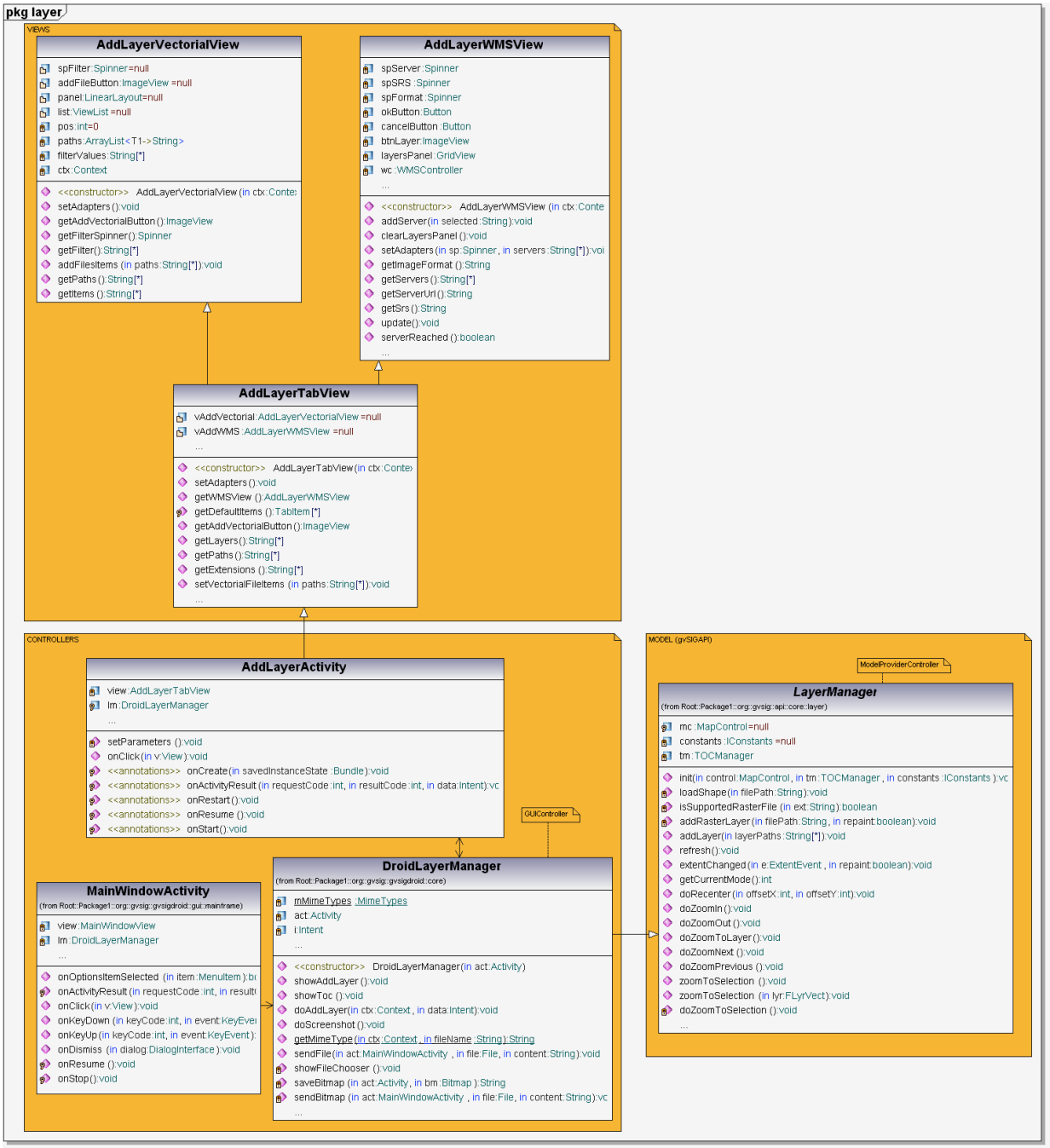


Figure 26. gvSIGDroid's Layer Module class diagram.

Hence, when selecting the "add layer" option, the *MainActivity* calls *AddLayerActivity*, which manages a *TabView* with two tabs. Moreover, there is a *DroidLayerManager* controller associated to the activity that extends from *gvSIGAPI*'s *LayerManager* abstract class, containing the method *addLayer()*.

When selecting tab one (figure 27), the user has to choose a filtering option (a) between *any, images, shapefile, kml/kmz, gpx, gml* and *xml*. Then, a filepicker will appear when the plus button is pressed (b) some files are selected and shown at the selected layers list (c). If some layer has been selected by mistake, it is possible to erase it from the list before starting to draw. Only after pressing the "ok" button will the system start drawing into the MainView (d).

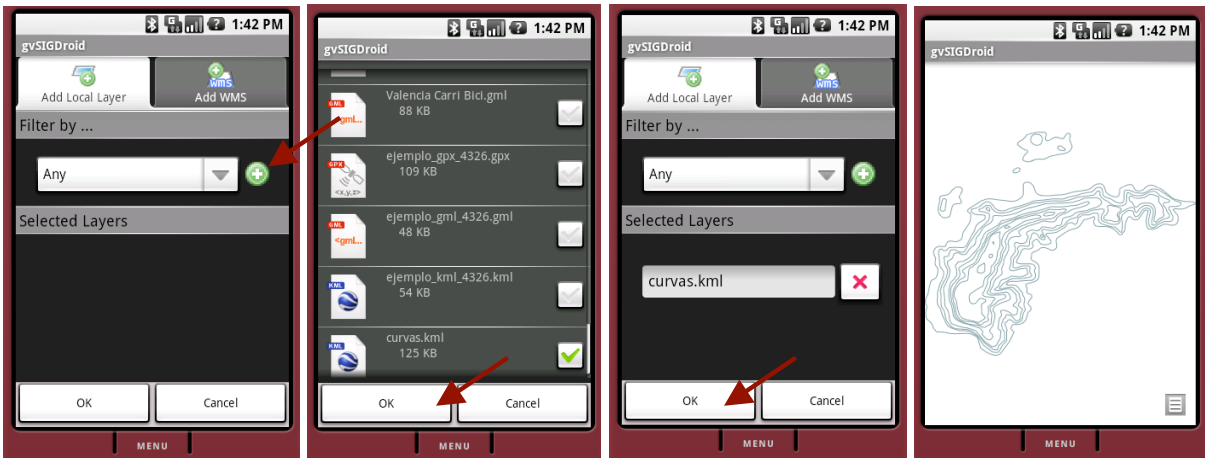


Figure 27. Adding a local layer.

B. Add WMS Layer:

This option is only possible when an Internet connection is available. Then, a server can be selected from a given list, and a dialogue window will appear for selecting the layers of interest to be loaded. Once the dialogue is dismissed by pressing "ok", the Format and SRS can also be selected. Then, when pressing the "ok" button the selected layers will be loaded into the project.

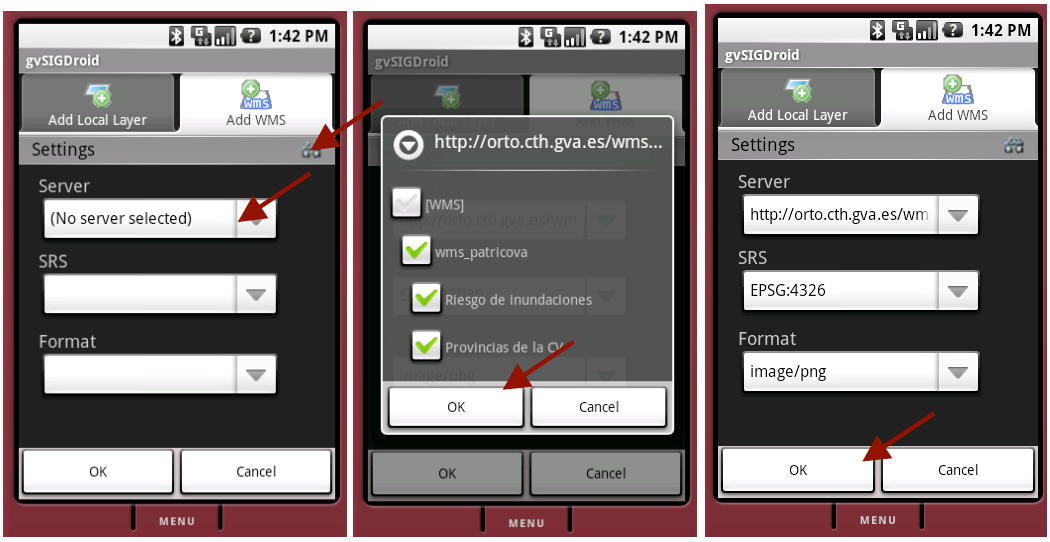


Figure 28. Adding a Web Map Service layer.

The remote connection to the server is done via a WMSController at a very low level, at the libFMap module.

4.6.4. Navigation module

Once a layer has been added to the project, we can navigate through it, or apply some functionalities such as measuring distances or editing new points, lines or polygons on it. The different navigation modes, as depicted in Figure 29, are: Zoom mode, Pan mode, Selection mode, Measurement mode and Edit mode.

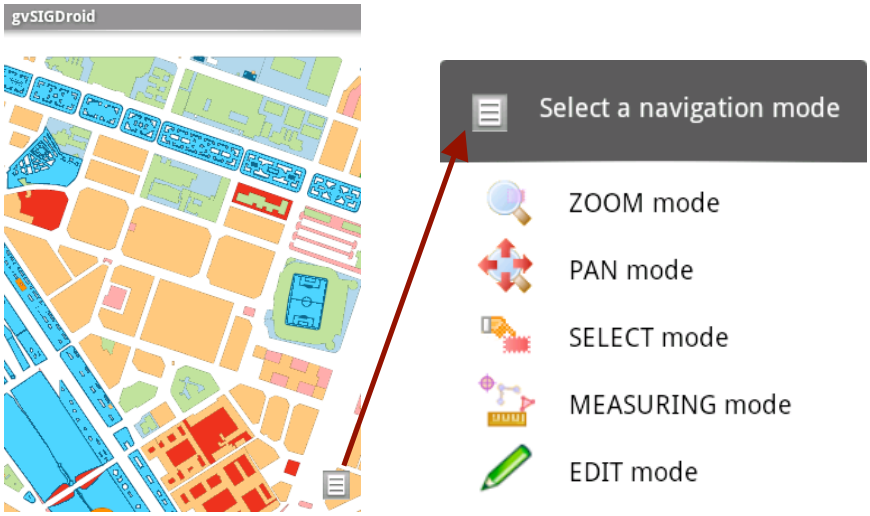


Figure 29. gvSIGDroid's Navigation Button (left) and Menu (right).

A. Zoom mode:

Layers can be zoomed in, zoomed out, zoomed to a selection, zoom to previous, zoom to next, and zoom to layer, among other options (see figure 30).

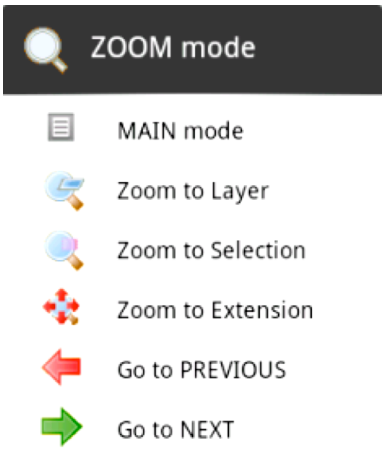


Figure 30. Zoom options.

Users can interact with their fingers. For example, when the user is at zoom mode, they can drop down, drag and drop up to select a specific region of the map to do zoom in (figure 31).

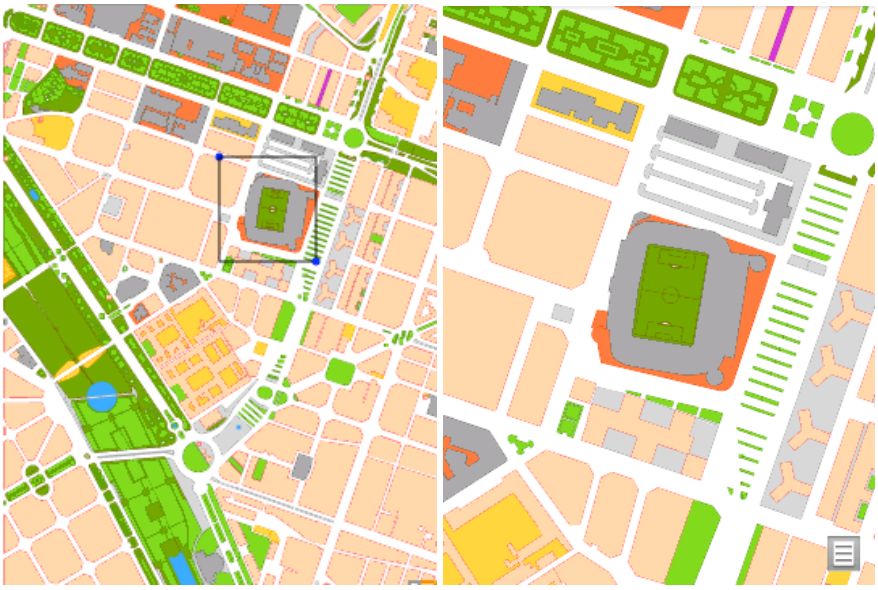


Figure 31. Zoom in to selection.

B. Pan mode:

With the d-pad buttons of the mobile device, users can pan left, down, up and right from the current view. Moreover, one can click on pan mode over the screen and the view will be centered to that point, allowing users an easy way to interact with the application, controlling it, which in turn increases the users' satisfaction.

C. Select mode:

Users can also make selections interactively just by selecting by point or rectangle. In this case, the vector items within the selected region change to the default selected color.

D. Measuring mode:

Measuring capabilities are also available in gvSIGDroid. So, users can measure by point, in which the point coordinates are shown; measure by line, showing the total path length; and measue by area, showing the overall surface of a closed polygon.

Figure 32. Measure by line capability.

4.7. Internationalization issues

As previously mentioned, managing different languages in gvSIG differs from gvSIGMobile and gvSIGDesktop to gvSIGDroid. In Android, languages are managed in different string.xml files located in subfolders of /res/ starting with the name "value" and following its correspondent language code.

Unfortunately, although internationalization and localization are critical, they are not yet completely available in the current SDK of Android (version 1.0). As the SDK matures, this section will contain information on the Internationalization and Localization features of the Android platform. In the meantime, it is a good idea to start by externalizing all strings, and practicing good structure in creating and using resources.

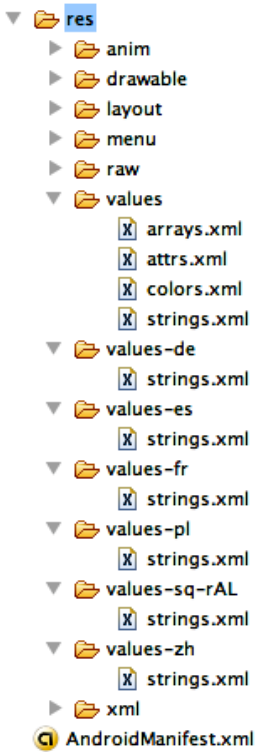


Figure 33. Structure of Android's resources folder.

So, arrays and strings can contain the values of *strings.xml* files to differentiate between languages. Internally, Android uses *java.lang.Locale* class, accessing to its referent id into the xml file, and if not found uses the default xml file of the values folder. More on this topic can be found at (AndroidD, 2008).

Being aware that internationalizing gvSIGDroid is crucial for increasing the market and propagating gvSIG all over the world, language files are available in Spanish, Catalan, Galician, Basque, Chinese, Albanian, Italian, Portuguese, French, Polish, English and German. Unfortunately, Android devices are still not fully supporting different languages. Nonetheless, on the emulators some parameters can be changed so that the emulator runs on a specific Local and our internationalization changes.

Figure 34. Screenshots of gvSIGDroid's layer menu in different languages.

5. CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

The present paper attempted to migrate gvSIGMobile software into the Android platform. After some difficulties integrating both libraries, the implementation of the Graphical User Interface in Android provided the application with a clean, nice and user-friendly interface.

gvSIGDroid offers the users of this first prototype to load almost any kind of geographic data resource. Consequently, users can utilize this program to visualize geographic information about their surroundings from the field, facilitating on-the-fly decision-making.

When comparing gvSIGMobile to gvSIGDroid, both share the same libFMap library, but gvSIGMobile uses java.awt graphics library while gvSIGDroid does not. These libraries are strongly bound to the core module library of libFMap, so it was hard to isolate those bindings for independent utilization in the Android version.

In terms of usability, gvSIGDroid offers an intuitive interface in which users do not need to interact that much with the application in order to understand what each of the buttons is for. Nevertheless, the first release contains several inconveniences. For example, the loading and drawing of the layers take much longer than in gvSIGMobile. This is due to the fact that gvSIGDroid first uses the libFMap objects as they were designed in gvSIGMobile, before adapting them to its unique Android objects, consuming substantial time and resources. In future implementations, it would be preferable to create a core module to reimplement libFMap. This module would retain most of the original code yet include specific Android core objects at an earlier stage.

Another inconvenience for field-based researchers when using gvSIGDroid is that, within the program, there is still no possibility of taking advantage of the GPS hardware of the device, since the GPS module has not been implemented at all.

5.2. Future work

Some future work should be done in order to enhance gvSIGDroid with a wide set of GIS functionalities. On the one hand, it is important to explore new ways of loading big shapefiles more efficiently. As said, libFMap should be re-implemented to provide Android shape inner classes at earlier stages, merging both Android and gvSIG data shape models for drawing.

Another field for extending gvSIGDroid will be adding Location-Based Services in the form of GPS services. GPS is crucial since it helps to real-time field data gathering, as witnessed in (Simonis, 2001). In this context, it is also important to add advanced editing capabilities to the application.

Moreover, the gvSIG extension for sensors presented by Alain Tamayo, a parallel project of the University Jaume I, could also be integrated into gvSIGDroid. Integration of this extension may prove very interesting, as it would provide users with useful terrestrial, submarine and aerial information. Finally, it could also be worthwhile to integrate a mobile software framework for all possible OGC Web Services, as stated in (Bröring, Förster & Simonis, 2006), which could be feasible within a gvSIG realm.

BIBLIOGRAPHY

ANDROIDA, ACTIVITIES AND TASKS, 2008, (URL: http://developer.android.com/guide/topics/fundamentals.html#acttask, Retrieved 26-02-2009).

ANDROIDB, ARCHITECTURE, 2008, (URL: http://code.google.com/intl/es-ES/android/what-is-android.html, Retrieved 26-02-2009).

ANDROIDC, LICENSES, 2008, Site of Google Android about Android Licenses (URL: http://source.android.com/license, Retrieved 26-02-2009).

ANDROIDD, RESOURCES AND INTERNATIONALIZATION (URL: http://developer.android.com/guide/topics/resources/resources-i18n.html , Retrieved 26-02-2009).

APPLE, IPHONE OS, 2008, Official site of Apple for iPhone Developers Ltd. (URL: http://developer.apple.com/iphone/, Retrieved 26-02-2009).

ARCGIS MOBILE, 2008, ArcGIS Mobile: Customer Success Stories (URL: http://www.esri.com/software/arcgis/arcgismobile/success_stories.html, Retrieved 26-02-2009).

BOLL, S., BREUNIG, M., JENSEN, C. S., KÖNIG-RIES, B., MALAKA, R., MATTHES, F., PANAYIOTOU, C., SALTENIS, S. & SCHWARZ, T., 2004, Working Group - Towards a Handbook for User-Centred Mobile Application Design, In *Proceedings of Mobile Information Management*, (Dagstuhl: IBFI), (URL: http://drops.dagstuhl.de/opus/volltexte/2005/166/pdf/04441.SWM3.Paper.166.pdf, Retrieved 26-02-2009).

BRÖRING, A., FÖRSTER, T., SIMONIS, I., 2006, Integrated Software Framework for OGC Web Services. *Free and Open Source Software for Geoinformatics (FOSS4G).* (URL: http://www.ingosimonis.de/_dl/Foss4G06.pdf, Retrieved 26-02-2009).

BROWN, L. B., 1949, The Story of Maps. (New York: Bonanza Books).

ESRI, 1969, Official site of Environmental Systems Research Institute (URL: http://www.esri.com/, Retrieved 26-02-2009).

GOODCHILD, M. F., JOHNSTON, D. M., MAGUIRE, D. J., & NORONHA V. T., 2004, Distributed and mobile computing. In *R. B. McMaster & E. L. Usery, (Ed.), A Research Agenda for Geographic Information Science,* (Boca Raton: CRC Press), pp. 257-286. (URL: http://www.geog.ucsb.edu/%7Egood/papers/400.pdf, Retrieved 26-02-2009).

GOOGLE ANDROID, 2008, Official page for Android Developers (URL: http://code.google.com/intl/en/android/, Retrieved 26-02-2009).

IPHONE GUIDELINE INC, 2008, iPhone Human Interface Guidelines of Apple Inc. (URL:

KHALID H.M., 2006, Embracing diversity in user needs for affective design, *Applied Ergonomics*, 37 (4 SPEC. ISS.), pp. 409-418.

KJELDSKOV J., 2002, Just-In-Place Information for Mobile Device Interfaces. In *Proceedings of MobileHCI 2002, Pisa, Italy. Lecture Notes in Computer Science*, (Berlin: Springer-Verlag).

KLEIN J., MOON Y. & PICARD R.W, 2002, This computer responds to user frustration: Theory, design, and results, *Interacting with Computers*, 14 (2), pp. 119-140.

LEE, E., KIM, M.-J., KIN, M. AND JANG, B.-T., 2004, Network Perspective for Spatial Data Distribution to Wireless Environment, *12th International Conference on Geoinformatics*, Sweden. (URL: http://www.hig.se/~bjg/geoinformatics/files/p107.pdf, Retrieved 26-02-2009).

LOPEZ-GULLIVER, R., SOMMERER, C., MIGNONNEAU, L., 2002, Interfacing the Web: Multi-modal and Immersive Interaction with the Internet. In: *VSMM 2002 Proceedings of the Eighth International Conference on Virtual Systems and MultiMedia*, Gyeongju, Korea, pp. 753-764.

MAPTOOLS MOBILE, 2008, Official site of MapTools project for Mobile GIS (URL: http://mobile.maptools.org/, Retrieved 26-02-2009).

OPENMOKO, 2006, Official page for Openmoko Project (URL: http://wiki.openmoko.org/wiki/Main_Page, Retrieved 26-02-2009).

OSGEO, 2006, Official site of Open Source Geospatial foundation (URL: http://www.osgeo.org/, Retrieved 26-02-2009).

RIM, 1984, Official site of Research In Motion, Ltd. (URL: http://www.rim.com/, Retrieved 26-02-2009).

SILVEN, O. & JYRKKÄ K., 2007, Observations on Power-Efficiency Trends in Mobile Communication Devices, *EURASIP Journal on Embedded Systems*

SIMONIS, I., 2001, Optimization of field data management using mobile GIS, wireless technologies and distributed simulation and visualization systems. *Proceedings of 7th EC-GIS, Potsdam, Germany*, pp. 5

SYMBIAN, 2008, Official site of Symbian Ltd. (URL: http://www.symbian.com/index.asp, Retrieved 26-02-2009).

SYMBIAN, RUNTIME ENVIRONMENTS, 2008, Official site of Symbian Ltd. (URL: http://developer.symbian.com/main/downloads/papers/runtimes_feature_table.pdf, Retrieved 26-02-2009).

ULLMAN, D. G., 1997, The Mechanical Design Process, 2nd Edition (New York: Mc-Graw-Hill).

VAN DER VLIET, S., 2007, Site of *Contactivity* about the Open Source Enterprise Content Management System, *OSMC: Combining Freedom and Reliability*, (URL: http://www.contactivity.com/en/Services/Contactivity-Internet-Solutions/eZ-publish/Enterprise-Content-Management-Combining-freedom-and-reliability, Retrieved 26-02-2009).

APPENDIXES

Appendix I: Hints for designing GUI features:

1) Less is more:

   Few features provide clarity on your layout, and people use it more (lots of features make people desist when trying to find out what feature to press).

2) Navigation

   - Efficient, fast, interacting, the screen should act as a mirror object.
   - They should always know where they are.

3) Lists

   - Use presets as possible.
   - Use controllers/adapters.

4) Pickers and spinners

   - For multiple values.
   - For constrained text.

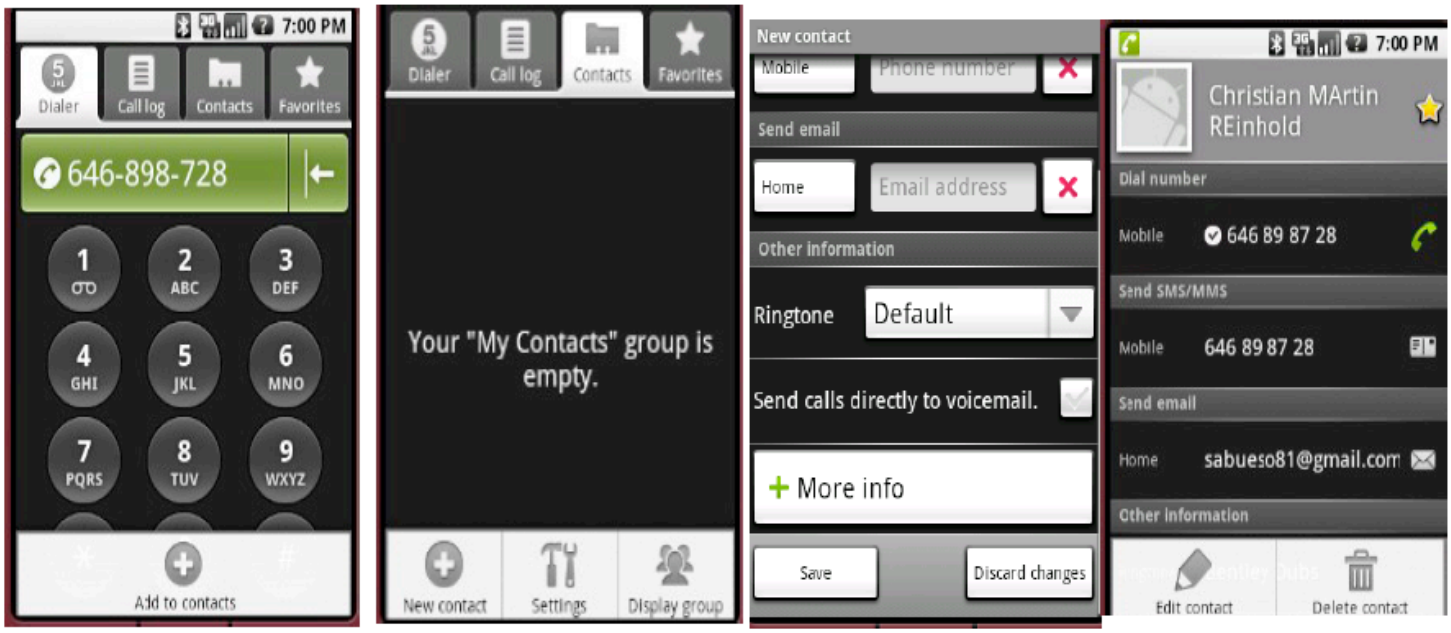
5) Toolbars vs.Tabbars

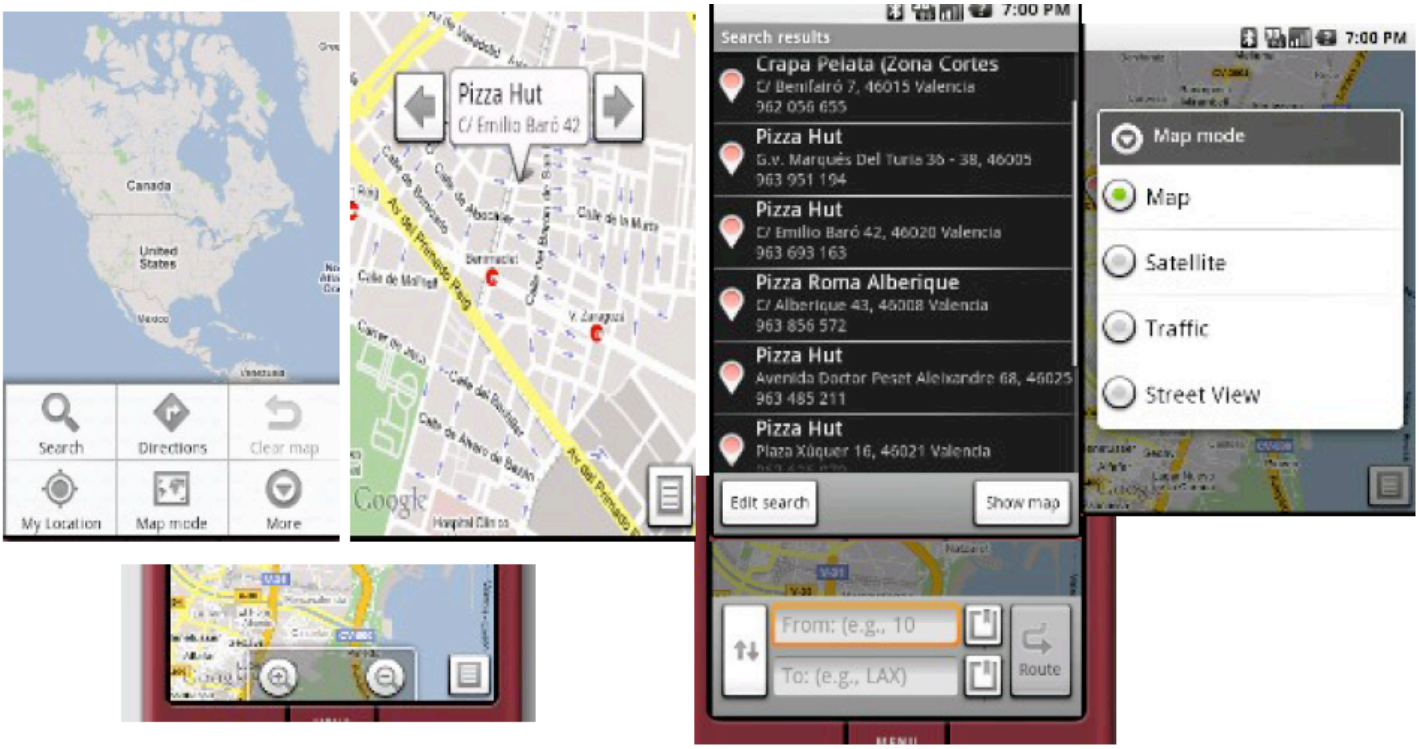
   - Distinguish between toolbars and tabbars. Toolbars do something and tabbars are different ways of seeing things (switch between modes).
   - The more important tab goes from left to right and from top to bottom.

6) Tables

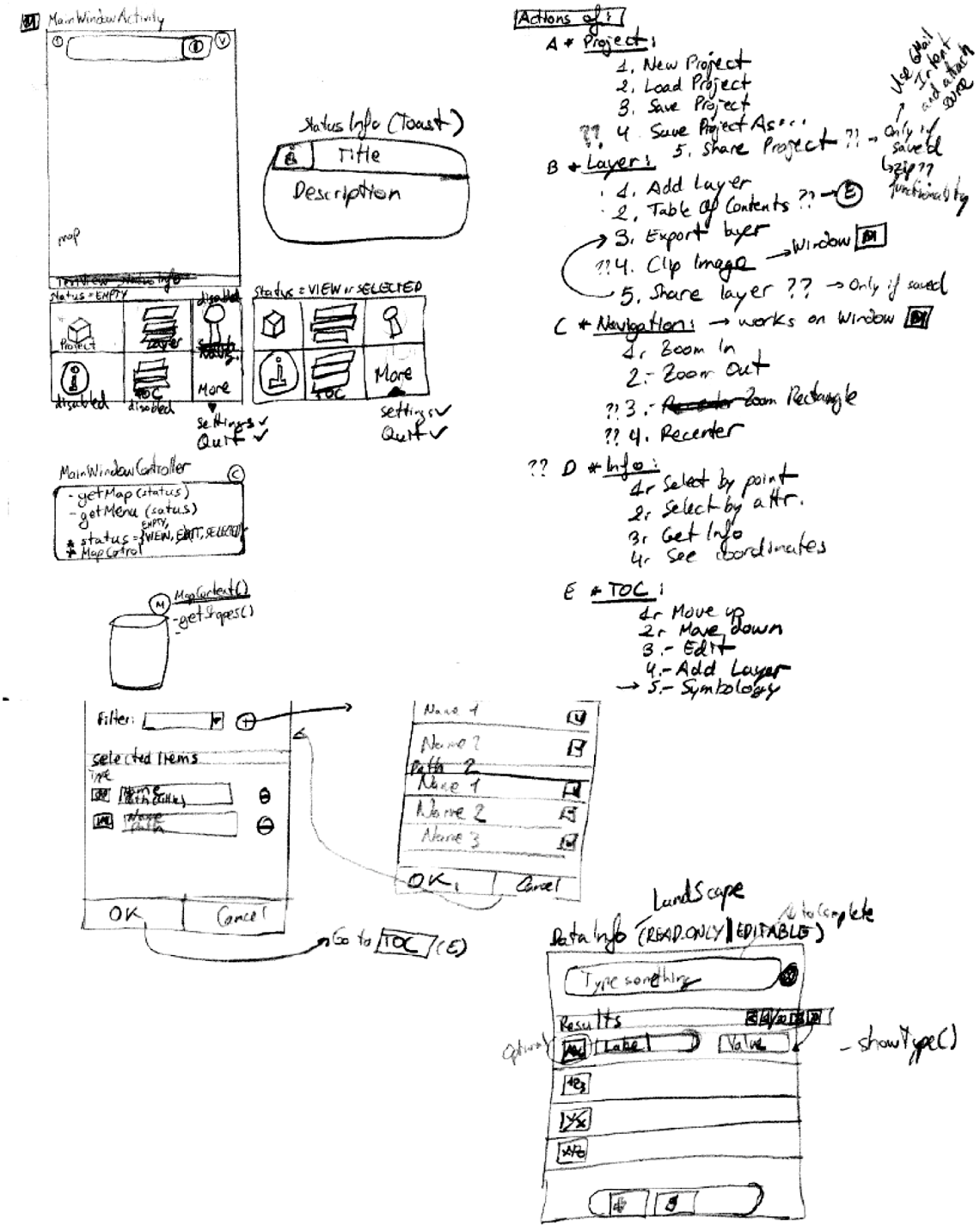
   - Show frequency and order by options.
   - Accessibility and freshness (refresh with the newest info).

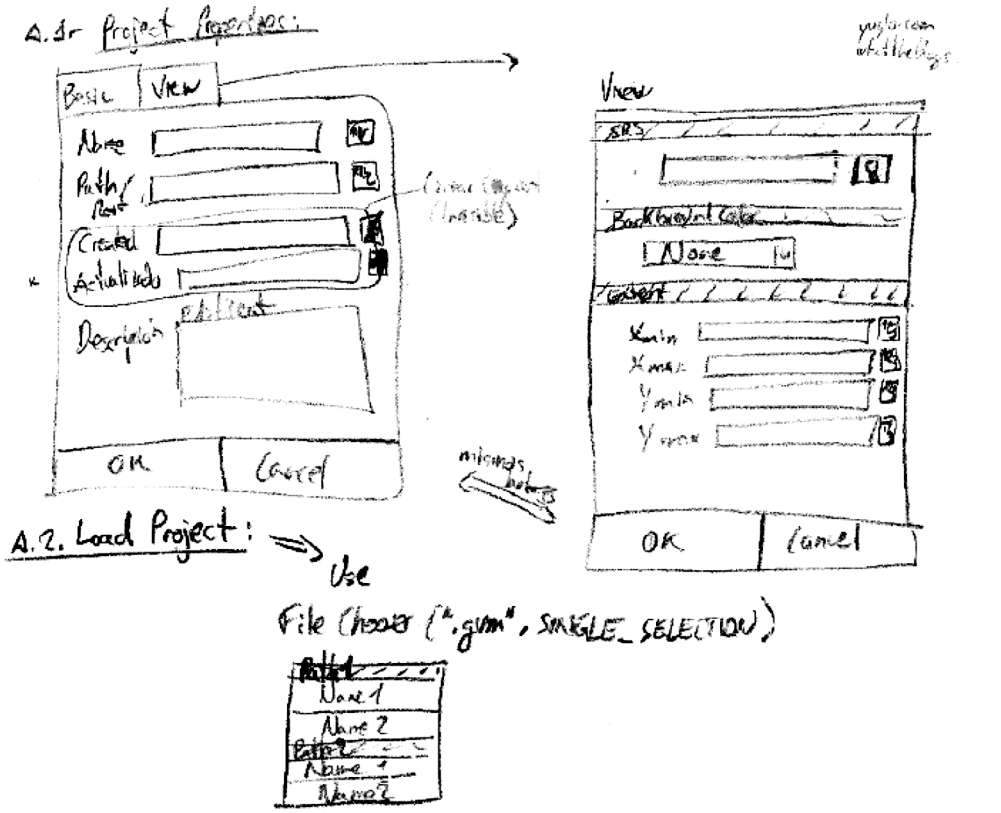
Appendix II: Android Layouts References

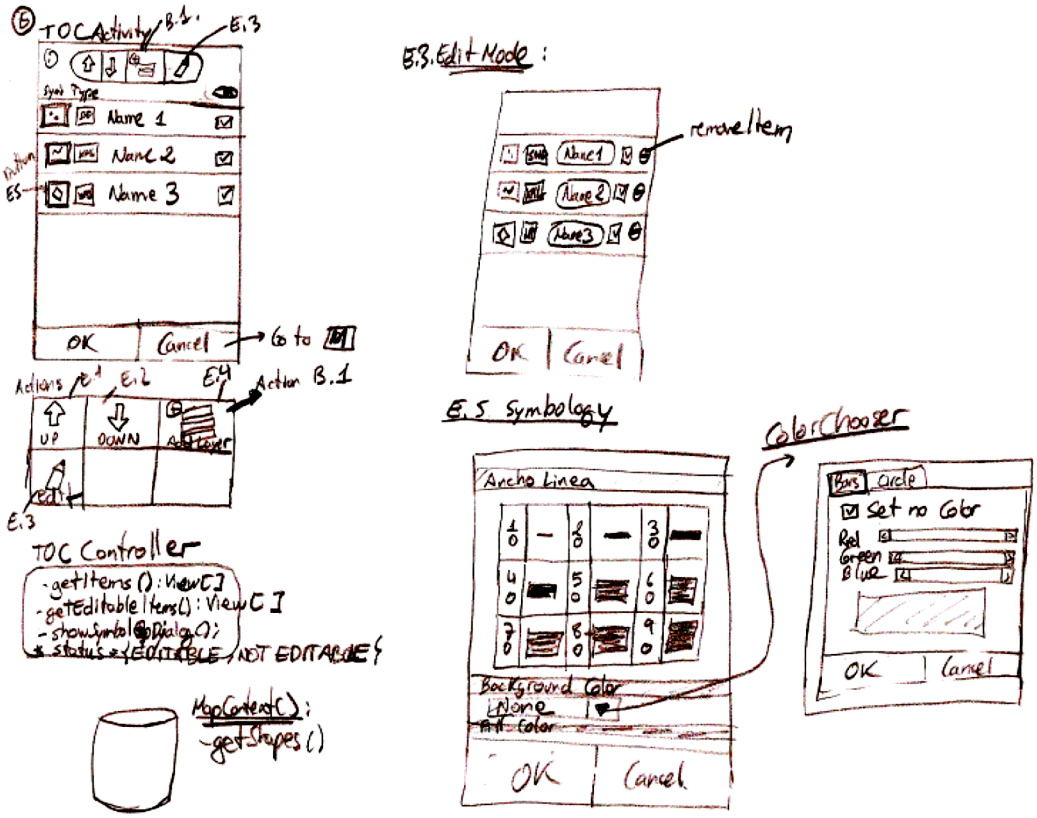


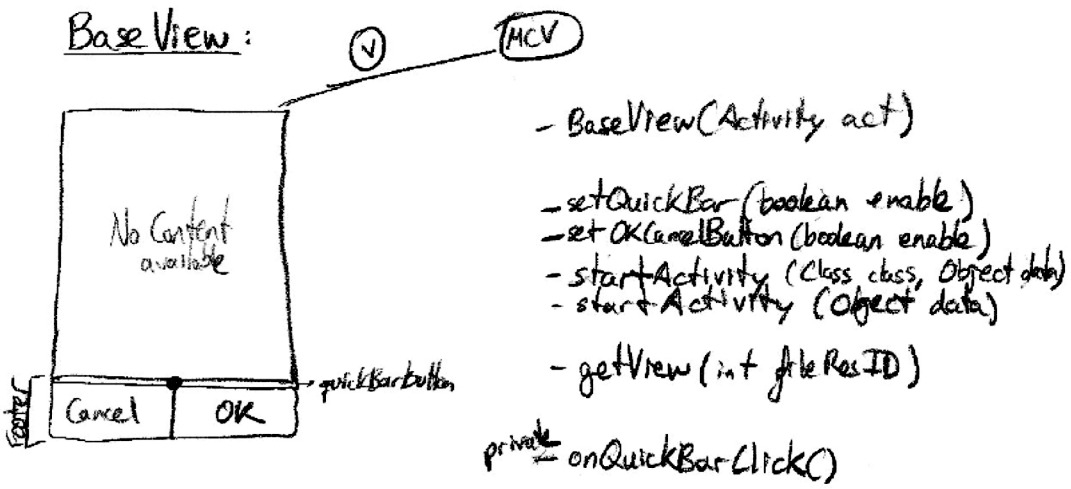


Appendix III: Initial sketches




A.1. Project Properties:
Basic
View
Name
Path
Created
Actualized
Description
OK
Cancel
View
Background Color
None
Xmin
Xmax
Ymin
Ymax
OK
Cancel
A.2. Load Project:
Use
File Chooser (".gvm", SINGLE_SELECTION)
Name 1
Name 2
Name 3


TOCActivity
B.2.
E.3
Name 1
Name 2
Name 3
OK
Cancel
Go to
Actions
E.1
E.2
E.4
Action B.1
UP
DOWN
Add Layer
Edit
E.3
TOC Controller
-getItems(): View[]
-getEditableItems(): View[]
-showSymbologyDialog();
-status = {EDITABLE, NOT EDITABLE}
MapContext():
-getShapes()
E.3. Edit Mode:
removeItem
Name1
Name2
Name3
OK
Cancel
E.5 Symbology
Ancho Linea
Background Color
None
Fill Color
OK
Cancel
ColorChooser
Bars
Circle
Set no Color
Red
Green
Blue
OK
Cancel



Note: this one ended being BaseView and OkCancelView

Appendix IV: General Licenses Description

About license types, since it is not within the scope, just the three licenses related to gvSIGDroid project will be generally contrasted:

1. GNU / GPL (General Public License)

GNU is a open source project based on Linux. Its acronym means GNU's Not Unix. GPL is a **strong-copyleft** license, which states that "everyone is permitted to copy and distribute verbatim copies of this license document**, but changing it is not allowed**".

2. GNU/LGPL, BSD License and MIT/X11

This acronym stands for *GNU Lesser General Public License* (formerly the *GNU Library General Public License*) or commonly *LGPL*. It is much more permissive than GPL, in order to copy with other permissive licenses such as the BSD license (for databases) and the MIT license.

The main difference between the GPL and the LGPL is that the latter can be linked to a non-(L)GPLed program, which may be free software or proprietary software. GPL must be free software.

You can change from LPGL to GPL since the latter one is more restrictive, but not the other way round. This is useful for direct reuse of LGPLed code in GPLed libraries and applications, or if one wants to create a version of the code that software companies cannot use in proprietary software products.

gvSIG projects use this specific license, as well as MIT/X11 for some particular third-party libraries.

3. Apache license:

Although the underlying Linux kernel is licensed under version 2 of the Free Software Foundation's General Public License (GPLv2), much of the user-space

software infrastructure that will make up the Open Handset Alliance's platform will be distributed under version 2 of the Apache Software License (ASL).

Code that is distributed under the ASL and other permissive licenses can be integrated into closed-source proprietary products and redistributed under a broad variety of other terms. Unlike permissive open-source licenses, "copyleft" licenses (such as the GPL) generally impose restrictions on redistribution of code in order to ensure that modifications and derivatives are kept open and distributed under similar terms.

For android projects, you must currently used Apache License (2.0 right now). Just copy and paste the following, changing the year put in brackets “[]”:

```
/*
      Copyright (C) [yyyy] Jeffrey Sharkey, http://jsharkey.org/

      This program is free software: you can redistribute it and/or modify
      it under the terms of the GNU General Public License as published by
      the Free Software Foundation, either version 3 of the License, or
      (at your option) any later version.

      This program is distributed in the hope that it will be useful,
      but WITHOUT ANY WARRANTY; without even the implied warranty of
      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
      GNU General Public License for more details.

      You should have received a copy of the GNU General Public License
      along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
```