Rey Juan Carlos University – Igalia
Master on Free Software
*2009 – 2010 Edition*

# Master Thesis

*Author:* Simón Pena Placer
A Coruña, November 11, 2010

# Contents

# Chapter 1

# Introduction

This report tries to cover, in a structured way, all the details of the work done during the master, as well as the conclusions reached. In this first chapter, the motivations which lead to the enrolment on the master will be discussed, the objectives intended to be accomplished will be exposed, and the expectations set on this course will be presented.

In the second chapter, the teaching work taken during the master will be introduced. The master modules, which group the main areas of knowledge to be taught, will be shown, followed by the actual subjects in which the modules were divided. Those subjects will be described, showing both the theoretical and practical work done within them.

In the third chapter, the work done in Grilo at Igalia, as the master's *practicum*, will be presented. Grilo project will be introduced, the tasks to accomplish will be exposed, the results obtained will be shown, and the future work and conclusions will be discussed.

In the fourth and last chapter, general conclusions for the whole master will be presented. The work done in the different sections will be related and linked and the initial objectives will be revisited, to check their completion status.

## 1.1 Background and motivations

I consciously know and use Libre Software[1] since my first year of studies at the Computer Science Faculty in the A Coruña University. In the five-year period that followed, I learnt about systems administration at a domestic level –local networking, firewalling, scripting for tasks automation or even security up to some degree. Besides, the degree itself made me know and use several programming languages, doing developments from small to medium-big. Since the second year, my Operating System of choice was

[1]Instead of using the terms *Free Software*, *Open Source Software* or *Free and Open Source Software*, *Libre Software* will be used mainly throughout this report

a GNU/Linux distribution: Mandrake at first, Debian followed to be later replaced by Ubuntu. In the same way, I tried to use *Libre Software* alternatives when possible, and also learn about Free Software philosophy, getting to attend to one of R. M. Stallman's conference held in A Coruña.

While my knowledge about Libre Software was enough for my daily needs, I had the feeling that I should somehow *formalize* and consolidate all that knowledge. The Free Software Master provided me with that opportunity.

## 1.2 Objectives

In order to being able to better evaluate my progress in the master, I set myself a number of objectives, distributed across several areas.

**System Administration:** My knowledge of this field was limited to common, domestic management. From setting up a local network to configuring a ssh server, I made use of tutorials or HOWTOs from the Internet, but had not enjoyed a real training on the matter: I was sure I was *reinventing the wheel* sometimes, or getting suboptimal solutions in many other occasions. The objective would be to formalize as much knowledge as possible, replacing the *raw* one with well formed one. At the same time, it would be important to get the ability to acquire new knowledge in an optimal way.

**Mobile Development:** I have been interested in development for mobile devices for some time, specially since I bought a Nokia N800. While I tried to start developing for it many times, never succeeded. I wanted to acquire good practices for mobile development, as well as some basic understanding of the most important APIs used in it. It would be interesting not to limit that learning to Maemo-based devices, but also try to target Android ones, or even mobile development in a generic way.

**Desktop Development:** While there are specific parts of mobile development which make it very different from desktop one, there should be many points in common, from sharing technologies to just making use of architectonic patterns, so that only the User Interface would need to be changed. I wanted to know the best way to make a new development so that making a port to the mobile world was plain easy. Related to that, I was interested in getting the most from existing Desktops, such as GNOME and KDE, understanding the balance between collaboration and competition. Which was the role *freedesktop*[3][2] was playing?

[2]It was a relief to know that GNOME and KDE are enjoying a great collaboration

**Licensing:** How can several projects collaborate? When is it possible to combine code coming under different licenses? Which are the main licenses *out there*? I wanted to know about that: since my Computer Science's end of degree, I firmly believe that reusing existing code is a key in the success of a development. But when reusing, you have to take into account lots of things...

**Role played:** While I had some experience using and enjoying *Libre Software*, my role had always been a passive one: I had just benefited from *Libre Software*, never contributing to it. That was a really important objective of mine: get to contribute to a *Libre Software* project, or at least, learn how to do it, if it were the case.

## 1.3 Expectations

By looking at the master program, it was quite clear that all the topics I was interested in were covered. However, I was aware that, in order to accomplish these objectives, I would need to work hard. The most obvious situation would be about contributing back to a *Libre Software* project: while I could learn the mechanisms to do it, I would still need to put time and effort to actually having something valuable to collaborate with.

The same could happen with desktop or mobile development: while the workshops would be interesting to start covering topics, I knew I would need to use a lot of my own time to consolidate that information.

My expectations were quite clear, however. I expected this master to put a lot of information on the table, to point to a lot of resources, and to serve as a starting point to the *Libre Software* world. I also had some expectations on start creating a professional network of *Libre Software* developers and companies and, luckily, jump professionally into that field.

# Chapter 2

# Subjects taken

This chapter covers the teaching program taken during the master. First, the modules covered will be stated, and the subjects in which they were divided will be presented. After that, an in-depth review of each module will be provided.

The master is organized around several fields or modules:

- Introduction to Libre Software
- Integration and Administration of Libre Software
- Libre Software development on Web environments
- Libre Software development on Destkop/Mobile environments
- Studies of Libre Software communities

While those modules present all the topics to be covered in the master, in order to make them easier to study and evaluate, they are separated into several different subjects:

**Introduction to Libre Software** During this subject, a general overview on Libre Software was introduced. Libre Software as a concept, with its motivations and consequences, was presented and then its history and most important actors were also contextualized.

It was also taught how Software Engineering techniques and Libre Software can benefit from each other, and, for that, licenses, patents and legislation are studied.

**Systems Integration with Libre Software** During this subject, a really wide range of technologies in the Systems Administration realm were presented. From networking, understanding and setting up web servers, email servers and other communication mechanisms to the security concerns most important when managing those systems, this subject made use of practices and seminars to help acquiring a knowledge very valuable for a Systems administrator.

Besides, to better perform some of the common tasks found on the *sysadmin* day, Bash Scripting and Perl were also used, so they could be more easily automated.

**Libre Software Development** During this subject, the main differences between a proprietary project and a Libre one were presented. Tools used to communicate and coordinate projects were introduced, as well as several development platforms, programming languages. To achieve that goal, this subject made use of several practices and workshops, where near-to-real-life projects were developed.

**Quality on Libre Software Development** During this subject, the knowledge acquired in the previous one was improved by introducing other elements very important for a successful project. As a result, documentation, packaging, versioning and distribution of Libre Software were studied. The importance of notifying, detecting and solving errors was stated, and unit testing and continuous integration were introduced.

**Detailed Technical Studies of Libre Software Projects** During this subject, some well known, existing Libre Software projects were studied, applying for that those techniques and knowledge acquired in the previous subject. Among others, there were seminars on Android, GNOME, KDE, Open Solaris and Telefonica I+D projects EzWeb and MyMobileWeb.

**Dynamics of Libre Software Communities** During this subject, the following concepts were introduced

- Business Models based on Libre Software. Migrations to Libre Software. Success stories and costs analysis
- Methodologies, techniques and tools to analyze Libre Software Projects, communities and dynamics. Data Sources and metrics used in Libre Software studies
- Analysis of business view of Libre Software

## 2.1 Introduction to Libre Software

In the course of this subject we learnt about the most important people in this movement, such as Richard Stallman, Eric Raymond, Linus Torvalds, Miguel de Icaza... We also learnt about the most successful FLOSS-based business models[1] such as dual licensing, open core, product specialists... Licenses were an important matter, too: we explored the most important ones, learnt the differences between permissive and *copyleft* ones, and got to know the Open Source Initiative and Free Software Foundation as the main entities to classify and approve different Free and Open Source licenses. Later we analyzed, using the Libresoft tools[4, 9, 2], several important communities and projects.

To better consolidate the acquired information, we have to develop several works:

- About the business models part, I presented a business model focused on service-oriented applications for mobile devices[5]. While accounting numbers are not necessarily accurate, all the process followed to obtain them, as well as sources and references, is perfectly valid should I decide to become an entrepreneur. With regards to the ideas presented, they are built on existing technologies and, in fact, some of them are being exploited right now.

- As an introductory work for evaluate Free and Open Source penetration in the real world, I did a review of the then-current Operating Systems in the mobile devices world[7]. While it was a valid review of the state of the art during Christmas, significant things have changed in such a dynamic field.

- For a practical training in the use of the Libresoft Tools, the Eye of GNOME project's source code repository and mailing list were analyzed[6]. That led to the identification of the most important contributors to the project.

- Once that we had undergone that training, a bigger analysis was performed. For that, I focused on the WebKit Project[10, 11]. My original idea was quite ambitious, as I wanted to compare WebKit and Gecko, but soon I focused on WebKit. While this work[8] is really interesting, the conclusions I reached were quite biased: my main measure to evaluate code collaboration was the committer identifier, while WebKit project stores the real author of a commit in the ChangeLog, which I ignored. As a future extension of the work, a ChangeLog parser should be done. Wit that, and reusing the existing scripts, tools and procedures, a very useful report could be done.

## 2.2 Systems Integration with Libre Software

In the course of this subject we dealt with networking, systems administration, servers configuration and management, version control with git, security... With the goal of achieving a good understanding of the common necessities of a systems administrator, we also went through task automation, using Bash scripting and Perl.

As we had previously done in the Introduction to Libre Software module, we also used some practical work to consolidate the knowledge acquired:

- To better learn Bash scripting, we worked[1] with regular expressions, making use of tools like find, sed or grep. Besides regular scripts, we also worked with daemons, learning about runlevels during the way.
- About Perl development, we followed[2] the "Learning by doing" motto. In a brief and interesting tutorial, we learnt the language basis and, in the last two exercises, we even played with Last.fm's API.
- In the networking practices, we worked[3] with proxies, subnets, DNS, DHCP. . . Again, we were asked for a good understanding of the inner workings, not only for results.
- Finally, as a logical complement of the all the previous process, we had to learn about security. Open ports, running processes, different kinds of exploits, different types of securing a network... Due to the sensitive nature of this matter, this work results are not publicly available.

## 2.3 Libre Software development on Destkop/Mobile environments

During this subject we were trained in tools and technologies like those used in many Free and Open Source projects. From the development point of view, we learn different languages, such as C + GObject and Python, and technologies, such as GStreamer, libsoup, libxml, GTK+ and D-Bus, as well as git for source code versioning, Emacs as the IDE or gdb and valgrind for debugging and memory profiling. From the quality point of view, we were introduced to important concerns such as unit testing, internationalization, localization, accessibility or even packaging.

Instead of individual practices like the ones we had previously done in other modules, this module made use of workshops to consolidate the knowledge acquired. Initially, a project was chosen which would use several different technologies –the said GStreamer, libsoup, libxml, GTK+ and D-Bus–. That way, JaMp was born.

JaMp is a Jamendo client: it has a backend which accesses Jamendo using libsoup, parses Jamendo's responses with libxml and does multimedia playback using GStreamer. Those operations can be controlled from

[1] `https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/ias/scripting/?root=freeswmaster`

[2] `https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/ias/perl/?root=freeswmaster`

[3] `https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/ias/networking/?root=freeswmaster`

a frontend using GTK+, which communicates with the backend by means of a D-Bus API. An iterative approach was applied to develop the project: starting with the backend, written in C+GObject, then the D-Bus interface was created to communicate it with a PyGTK frontend, developed as the last part. The development was done via patches sent to a mailing list to allow peer-review. Then, the module coordinators would check and integrate those patches, pushing them to a git repository at gitorious. Later on, I achieved git committer status, also becoming one of the patch integrators.

Besides learning those new technologies and getting familiar with git, we also made extensive use of other good practices, such as testing new additions or features introduced, using gdb for debugging or valgrind to check for memory leaks. We also got the project infrastructure updated to support internationalization and localization (commits cdb8a34 and c263105), and checked accessibility concerns in the User Interface.

While the initial target was the GNOME desktop, the project was meant to be adapted to Maemo devices. After a Maemo workshop, the project build was updated –requiring only minimal modifications as seen in commit 8e61de0– to target these devices, and a specific user interface was started, using PyMaemo.

As a reward for the work done, we had the opportunity to present the work at the 7th GUADEC-ES. I was in charge of writing the application letter to present the work, and later I also did the slides and gave the presentation. Not only it was nice to share the work we had done, but also to meet and learn what real hackers are doing.

## 2.4 Libre Software development on Web environments

In the course of the Web development module, different well-known technologies were learnt, such as PHP –with templates using Smarty–, CSS and Typo3, Java –later using Maven and the ZK framework–, Ruby on Rails, Django and JavaScript.

As we had previously done in the in the Desktop & Mobile module, to consolidate the knowledge acquired, a Java project was chosen. It was developed across several workshops, and made use of jUnit for unit testing, Maven for the project management and ZK for the User Interface. The same principles of code collaboration were used, such as peer-review sending patches to a mailing list. Besides, alternating those workshops there were different sessions to introduce Ruby on Rails and Django: for them, as well as for the initial PHP sessions, specific tasks and practices were needed[4].

The project developed in this module was a Rich Internet Application

[4]Django's Receitas Galegas and Ruby on Rails' blog

for the Getting Things Done methodology[5]. Following the Model-View-Controller pattern, first the model was created for the tasks, tasks lists, users and notes. All those pieces got unit tests checking their functionality, and later a ZK User Interface was provided allowing users to create and manage their own lists of tasks.

## 2.5 Studies of Libre Software communities

During this module, which was held transversally to the others, several important Libre Software communities were studied. Android, GNOME, KDE, Debian, OpenSolaris, Maemo and the Morfeo community, all had a session. During those sessions, we were told how the community worked: from the way they took decisions to their infrastructures, conferences or funding. In some of them, practical workshops were also held: that way we got a brief introduction to Qt on KDE, to Android application development in the Android session, to Debian packaging on Debian's, and got to play with OpenSolaris or EzWeb and MyMobileWeb in the OpenSolaris and Morfeo sessions.

It is worth mentioning the session we had with Carlos Guerreiro, from Nokia. He gave us a great review of Maemo's history, pointed to some interesting things he believes the future can bring in the mobile devices world and discussed with us his vision of *Free and Open Source Software* from the corporate point of view.

[5]`http://git.igalia.com/cgi-bin/gitweb.cgi?p=riagtd.git;a=summary`

# Chapter 3

# Practicum work

In this chapter, the master's *practicum* work in the Grilo project at Igalia will be described. First, a brief introduction about the project and the objectives set by the Igalia staff will be provided, also including those requirements dictated by the nature of the project.

After that, the actual work done on the project will be presented. Its prerequisites –such as technologies used, other projects needed, compilation issues. . . –, the difficulties found during the work –and their solutions or workarounds, when possible–, and contributions originated in its scope will be described.

Finally, conclusions and results for the work will be presented. Master's subjects will be linked to the technologies, tools and projects used, and new possible improvements –and potential issues– will be identified.

## 3.1 Introduction

Grilo[1] is a framework focused on making media discovery and browsing easy for application developers. It provides a high-level API that abstracts the differences between different providers, such as Youtube, Jamendo, Shoutcast, Vimeo, etc. It is licensed under the LGPL and developed in C using GLib and GObject, and it is already a GNOME project.

As the practicum work for the master, collaboration was to be done in the following areas:

- Improve Grilo's binding support, by
  - Studying the mechanisms for the automatic binding generation using GObject Introspection
  - Analyze Grilo's APIs and their conformity with the automatic binding generation infrastructure. Suggest improvements.

[1] `http://live.gnome.org/Grilo`

  - Perform the actual improvements on the introspection support
  - Test generated bindings

- General improvements in the framework and plugins system, by
  - Analyzing and fixing defects
  - Design and development of new features

The work being developed should be done as a real Free Software project. As a result, the tools and methods employed had to be those common in this type of projects, such as mailing lists, bugzilla, wiki, git..., being the communications always in English. The preferred communication channel would be the mailing list, but the IRC would be used for real-time communications.

## 3.2 Work done

There were two main tasks for the *practicum*: on the one hand, there was the binding support. On the other, more general improvements or fixes. However, many of those general improvements were made during the course of the enhancement of the bindings support.

### 3.2.1 Getting started with Grilo

In order to feel comfortable enough with Grilo to being able to collaborate to it, I started reading its documentation. It provided a good general overview of the project, detailed its API, and offered some code examples ready to be compiled.

Going through its documentation, I was able to contribute my first patches: fixing some typos or mistakes found in the documentation, adding some configuration lines to the examples so they could work and small thinks like that.

Thanks to those patches, I started becoming familiar with the code, its style and organization, and the build process.

### 3.2.2 Bindings support. What is GObject Introspection?

Once that I felt I was ready to deal with Grilo, I started reading documentation about binding generation, in general, and GObject Introspection, in particular.

It makes sense to build many kinds of applications using, at least, two different levels and languages. On one side, there are those based on C + GObject: usually they have great performances, being good for graphics, multimedia, and low level tasks in general. On the other side, there are those with a managed, with garbage collection, runtime, such as JavaScript, Python, Java, .NET... They are better suited for application logic such as configuration, layout, dialogs and/or tasks where performance is not that important.

It is important, then, to allow developers chose their language of choice. Not only they will be more productive: they will be happier. While the GNOME platform has been historically quite binding friendly, writing bindings usually involves an important amount of job to be done by hand. At the same time, the amount of resources devoted to a given project are limited, so having a developer occupy his time writing bindings will reduce the amount of time he can spend adding new features or fixing bugs, for example.

As a consequence, the GObject Introspection project was born[2]. With one of its major goals being to become a convenient bridge between the low-level, high-performance world and the high-level, "less-efficient" one, GObject Introspection solves the problem of writing bindings by hand by putting all the needed *metadata* inside the GObject library itself, using annotations in the comments. This will lead to less duplicate work from binding authors, and a more reliable experience for binding consumers. Additionally, because the introspection build process will occur inside the GObject libraries themselves, a goal is to encourage GObject authors to consider shaping their APIs to be more binding friendly from the start, rather than as an afterthought.

### 3.2.3 How do automatic bindings with GObject Introspection work?

In order to benefit from automatic bindings using GObject Introspection, a three step process must be followed.

1. **In the library developer side:** First, a C + GObject library must be annotated with some metadata –while there are some default assumptions, some annotations have to be made.

2. **In the GObject Introspection side:** The GObject Introspection scanner tool must scan the library to extract the metadata and generate a *gir* file: an XML containing type information, memory allocation and more. Then, the GObject Introspection compiler will compile the *gir* file to generate a typelib.

3. **In the target language side:** The language wanting to use that introspection information must provide a way to load that typelib.

Since GObject Introspection is quite a young project, there are still few languages using it. JavaScript, from both Seed and Gjs, or Python, via

[2]`http://live.gnome.org/GObjectIntrospection`

PyGObject, are two languages with a relatively mature support. Others, such as Lua, are still being developed.

It is worth noting that GTK-Doc should also understand all the annotations used in GObject Introspection –if not, it is a bug–, so they get recognized when scanning a library.

### 3.2.4 Bringing automatic bindings to Grilo

Once that I knew how automatic bindings worked, I started to analyze which things were missing in order to make Grilo automatically generate introspection bindings.

#### Initial status

Build support for GObject Introspection in Grilo had already been added by the time I started contributing to it. There was also a small test case, written in JavaScript, which made use of the automatic bindings. However, due to some previous changes in the way Grilo's metadata keys were stored, the test did not work.

#### The ParamSpec issue

These keys had been previously just constants, defined as integers or strings. However, since an important feature for Grilo was to allow developers define their own metadata keys, an approach using GObject's ParamSpecs –aliased as Grl.KeyIDs– had been introduced. While those ParamSpecs served their purpose just fine, their were totally *GObject Introspection unfriendly.* First it was needed to annotate each of them as a ParamSpec, as the aliased type would not work at the gir and typelib level. Then, it became clear that Gjs did not support them, and probably would not in the short term.

The workaround chosen was to annotate those keys as *uints*: with that modification, that simple JavaScript test worked fine. As the binding consumer is not required to know that KeyIDs are ParamSpecs –he is even discouraged, since that is a specific implementation detail– and all the methods returning KeyIDs or receiving KeyIDs are ParamSpec-agnostic, that workaround should work fine. However, it relies on being able to store a pointer in an *uint*: in platforms when that assumption is wrong, it would fail.

#### Going upstream

After some discussion in the mailing list, I was suggested to use upstream versions for GObject Introspection and Gjs: maybe in the latest versions the issues we were experiencing would be solved. Until then, I had been using Ubuntu Lucid Lynx's packages, which became quickly outdated when dealing with projects so young and fast-paced. In order to start using the upstream versions, I set up JHBuild:

*JHBuild is a tool designed to ease building collections of source packages, called "modules". JHBuild uses "module set" files to describe the modules available to build. The "module set" files include dependency information that allows JHBuild to discover what modules need to be built and in what order.*

By using JHBuild with the GNOME 3 "module set", I started using GObject Introspection and Gjs from upstream. However, since that did not fix the ParamSpec issue, I decided to test the bindings from another language: Python.

### Using Grilo from Python: PyGObject

Getting JHBuild to build latest PyGObject was a bit tricky: the documentation was outdated, and pointed to a no longer valid PyGI module. With Javier Jardón and Nacho's (Ignacio Casal) help, I realized that only PyGObject was necessary.

As PyGObject did support ParamSpecs, it became the *official* language for testing the bindings. I then started to annotate all the methods in Grilo, one by one. In order to be able to check the annotations and functionality of the bindings, I cloned an existing test application –grilo-test-ui– from C to Python, using PyGObject.

### Unit testing from Python

Although writing a PyGObject clone for grilo-test-ui proved very useful, to identify existing issues in some of the methods more commonly used, there was a lot of code unchecked. To achieve a better coverage, I suggested the creation of unit tests in Python, using PyGObject to invoke Grilo's methods.

These unit tests had a double effect: not only they were useful to identify parts of the code not introspectable or presenting wrong annotations –which can lead to wrong *free*s and / or crashes– but also to test the library's behavior. Right now, Python unit tests are fully integrated in the project's build –provided that GObject Introspection support is found in the target system–, covering Grilo's most common methods, and having led to the identification of several bugs.

### Current status of the bindings

Right now, save for three functions non-introspectable, all the API can be accessed via GObject Introspection from Python. In Gjs, as ParamSpecs are still unsupported, only the API which does not use them works. However, when the ParamSpecs are annotated as *uints*, Gjs can achieve the same degree of functionality as Python.

Currently, a Gjs' branch has been suggested, containing a clone of the C application grl-inspect, which provides an overview of the currently available plugins, and allows checking JavaScript's support of the bindings in a more complex way than the initial test case. While that branch has not been pushed to the git repository yet, it is available[3] in the mailing list archives.

### 3.2.5 Other improvements and fixes

Besides the work done on the annotation field, several other improvements were made, such as code rewritings, build fixes or improvements or documentation enhancements. Among them, it is worth mentioning Vala's metadata improvements, needed for generating its binding, or the rewriting of the *get_sources* method, so it could return a GList instead of an array, not supported in Gjs.

The full list of contributions to the Grilo core is as follows:

```
Simon Pena (64):
      doc: Added grl_init call
      doc: Minor corrections
      doc: Corrected obsolete tags
      core: Replaced grl-config defines with functions
      core: Replaced grl-media defines with functions
      core: Replaced grl-media-video defines with functions
      core: Replaced grl-media-image defines with functions
      core: Replaced grl-media-audio defines with functions
      annotations: Annotated grilo.c
      annotations: Annotated grl-plugin-registry
      annotations: Annotated grl-media-source
      annotations: Annotated grl-media-source callbacks
      core: Make grl-metadata-key more introspection friendly
      annotations: Annotated grl-multiple
      annotations: Annotated grl-data
      python: Cloned grilo-test-ui
      core: remove uneed frees
      core: removed wrong free at GrlMediaPlugin finalize()
      annotations: Fixed 'lookup_source' annotation
      core: Remove value destroy_func from the plugins' hash table
      core: check for 'key-depends' implementation
      annotations: Added missing colon
      annotations: Replaced GObject.ParamSpec* with GObject.ParamSpec
      annotations: Replaced Grl.Media* with Grl.Media
      annotations: Replaced Grl.MediaSource* with Grl.MediaSource
      annotations: Skipped non introspectable functions
```

[3]http://mail.gnome.org/archives/grilo-list/2010-October/msg00011.html

```
      core: Made GrlPluginInfo introspectable
      annotations: Added missing transfer modes
      annotations: Annotated grl-metadata-source
      annotations: Annotated metadata_source cb GErrors as uints
      annotations: Fixed 'get_info_keys' annotation
      core: Replaced grl-media-plugin defines with functions
      annotations: Annotated 'register_metadata_key'
      annotations: Removed unnecessary annotations
      annotations: Media in cb is transfer full
      annotations: Fixed grl_data_get_keys transfer mode
      doc: Updated examples to use new log system
      core: Make get_sources return a GList
      core: Make get_sources_by_operations return a GList
      annotations: Fixed grl_init annotations
      doc: Improve GrlMediaAudio documentation
      doc: Improve GrlMediaImage documentation
      doc: Improve GrlMedia documentation
      doc: Add GrlLog documentation
      doc: Fix GrlMediaVideo typo
      doc: Skip GrlDataSync
      doc: Added GrlError documentation
      doc: Fix GrlUtil documentation
      build: Define major, minor and micro version vars
      build: Fixed gir build rules
      vala: Improved metadata
      core: Moved tests infrastructure to /tests
      core: Updated infrastructure to support python tests
      tests: Command line arguments can be passed to testrunner
      tests: Tested the PluginRegistry class using GI
      tests: Tested the PluginMedia class using GI
      tests: Tested the MetadataSource class using GI
      tests: Removed setUp/tearDown code in python tests
      annotations: removed transfer mode for 'in' args
      annotations: Fixed transfer modes in grl-metadata-source
      core: reworked GrlMetadataSource's filter methods
      tests: updated GrlMetadataSource filter tests
      annotations: fixed GrlConfig annotations
      python: added plugin configuration to grilo-test-ui
```

There were also minor contributions to the Grilo plugins

```
Simon Pena (4):
      vimeo: Updated calls to grl-media-video
      flickr: Removed const qualifier from grl-config
```

```
bookmarks: Updated call to grl-media
filesystem: Updated call to grl-media
```

and to GTK-Doc (Fix 'scope notified' annotation), GObject Introspection (gio-2.0.c: Add missing annotations) and PyGObject (Don't check the inner type when comparing gpointers). During this work, several bugs were filed, all of them being fixed save for the ParamSpec one in Gjs.

## 3.3 Conclusions and future work

Automatic bindings using GObject Introspection are a really promising project. The number of possibilities they can offer is just huge –just imagine developing a Django Web Application which could use, natively, all the plugins and functionality that Grilo offers!

However, while GObject Introspection is really interesting, it still feels young. The number of languages offering good support for it is still small and there are still important changes in its behaviour: recently, the default *transfer mode* stopped being *full*, requiring library developers to explicit state it. As a result, many projects stopped offering automatic bindings. While the decision is easy to understand (requiring library developers to state which *transfer mode* they need guarantees that there will not be leaks nor wrong frees), it serves as an example of how unstable the project is at the moment.

With regards to collaborating on a Free Software project, contributing to Grilo has been a great experience. I became familiar with the tools like gdb, to be able to debug both C + GObject code or to locate a estrange crash in Python invoking Grilo methods; valgrind, to check if I was having a leak due to incorrectly freeing memory; JHBuild, to build a GNOME environment in a sandbox, using upstream versions; git, to create branches, apply patches, check the history, locate an error thanks to *git bisect*...

I also felt comfortable with languages I had not used too much, such as JavaScript or Python, learning about unit testing with the latter, and continued learning about C + GObject.

Improving Grilo's support of GObject Introspection also gave me the chance to keep an eye on other interesting projects such as PyGObject, Gjs or GTK-Doc. I had the opportunity to find and file bugs on them, sometimes even providing patches.

As for the future work, while it should be important to keep the GObject Introspection support updated (checking changes related to default parameters or new features), it would also be interesting to provide new plugins. Improving the existing Python unit tests would be useful and creating Gjs' ones would allow a better evaluation of that language's support. Seed's matureness, compared to Gjs, should be checked, and a better and larger test in Vala should definitely be added, too.

# Chapter 4

# Conclusions

This report has covered all the work done during the third edition of the Master on Free Software.

First, the background and motivations leading to the enrolment were stated, and a set of objectives was introduced to allow a proper self-evaluation once that the master was over. The evaluation is really satisfactory, but a more detailed one will be done in the last section of this chapter.

Then, the subjects covered during the master were reviewed and the work done on each area, both theoretical and practical, was presented. While the fields of knowledge covered were really broad, all the subjects were really enjoyable. The next section will present some specific conclusions about them.

After that, the master's *practicum* work done in the Grilo project at Igalia was explained, as well as other contributions originated from that work. More complete conclusions will be presented in the second section of this chapter.

## 4.1 Subjects

The subjects taken during the master covered a really wide range of topics. While my main interests were on the development modules, I was happily surprised with a subject I had not think of: Software Engineering techniques applied to *Libre Software* projects.

There are really large *Libre Software* projects (such as the Linux kernel, GNOME, KDE or many others) with plenty of open information available (software code repository, bugs databases, mailing lists archives. . . ), being possible to analyze all these data sources, getting to know the health of the project, who the main developers are, how bugs are distributed between releases, how active the community around the project is. . . But not only can *Libre Software* projects benefit from Software Engineering techniques, but also those techniques can be improved. Accessing to proprietary software code to perform certain analysis was expensive and required researchers to sign NDAs, and that would cause those analysis to be impossible to replicate, banning peer review. As a consequence, I felt like doing Software Engineering analysis on *Libre Software* projects was not only an innovative approach but also a logical one.

### 4.1.1 FOSDEM

As a complementary activity which could be considered part of the *Introduction to Libre Software* and *Detailed Studies of Libre Software Communities*, the students were sponsored to go to the FOSDEM in Brussels: The Free and Open source Software Developers' European Meeting, one of the most important *Libre Software* conferences. During a weekend in February we had the opportunity to attend to several events: from a Beer event, where developers would meet and have a beer, to lots of different keynotes, passing through a key-signing party.

Meeting the key developers of many important projects and communities, listening to them talking about their work or just enjoying the ambient: it was an unforgettable experience.

### 4.1.2 GUADEC-ES

After the work done on the Desktop & Mobile development project, and taking advantage from the GUADEC-ES being in A Coruña, we were suggested to submit a talk proposal.

The GUADEC-ES is the main GNOME conference in Spanish language. In its 7th edition, it was based on A Coruña, which made easy for us attending to it. I was responsible of elaborating the talk proposal, explaining the development done in the Desktop & Mobile application.

As the proposal got approved, I went to write the presentation slides and give the talk. It was a really satisfactory experience, even if I felt that we still had a lot of work to do to get close to other GNOME *hackers* –and become *hackers* ourselves. Nevertheless, the conference was very interesting and people there was helpful and willing to share what they knew, so it was a perfect complement to the development courses.

## 4.2 Practicum

During the *practicum*, I had the opportunity to put in use all the knowledge acquired during the master's subjects. I employed technologies such as git, for source code versioning and generating patches for peer reviews, and did extensive use of the mailing list and IRC to communicate with other Grilo developers (setting up a server for making Grilo IRC logs available).

As a consequence of using so *cutting-edge* technologies, it was common to find issues either in GObject Introspection itself or in PyGObject, Gjs or even GTK-Doc (as it needs to recognize the annotations used in GObject Introspection). That led me to identify, file and sometimes fix bugs present in these projects, while communicating with other upstream *hackers*. All this experience was really rewarding, and as a result of it, I also got committer permissions to the GNOME project.

## 4.3 Summary

As a final summary of this report, a review of each individual is provided:

Formalizing Knowledge: While initially I only focused on formalizing my systems administration knowledge, now that the master is over, I feel like I have consolidated all fields studied. From said systems administration to programming, passing trough licensing or Software Engineering, I now have a really broad vision of these areas. I have discovered new sources for further learning, good channels to ask to experts and a number of contacts in the field who will surely prove helpful. A good example for that already happened during the GUADEC-ES, when well-known *hackers* Javier Jardón and Nacho (Ignacio Casal Quinteiro) helped me solve some JHBuild issues.

Mobile and Desktop Development: As I have hoped, mobile development was focused in a way which allowed reusing of existing components and technologies, also present in desktop developments. We enjoyed Maemo and Android sessions, and were also told about MyMobileWeb, a project which targets mobile devices from the web development point of view. Accomplishing this objective was really rewarding: besides the practical development being done in the master scope, I could give a great push to a personal project of mine, Butaca, which follows the same architectonic design principles. Having the opportunity to attend to FOSDEM and GUADEC-ES and giving a presentation in the later was simply great.

Desktop Development: The existing collaboration between KDE and GNOME communities is really a relief, and should work fine against the typical said that they are fighting each other and wasting resources. The freedesktop project, being a strong *foundation*[1] on which both projects can rely is a great example on how collaboration and competition are the engine which moves *Libre Software*

Licensing: As I previously said with regards to *formalizing knowledge*, I know feel like I have consolidated my knowledge of the licensing field.

[1]As in buildings

I can now tell from different licenses, classify them between *permissive* and *copyleft*, know what are these kind of licenses better for, and how they interact with each other. With this information, I can also help deciding which license fits better a new project in an area where it is being innovative, or which one is more suitable when there are already other options.

Role Played: My first contribution to a *Libre Software* project got accepted quite early during the master, being a patch to mlstats which improved the FROM field extraction when parsing mailing lists. After that, I contributed with many improvements to Grilo in the *practicum* scope, from enhancements in the automatic binding generation to code optimizations or other fixes. While doing the contributing to Grilo, I also had the chance to file bugs and provide patches to different issues in other projects, such as GTK-Doc, PyGObject, or GObject-Introspection.

# Bibliography

[1] Flossmetrics. Floss-based business models - sme guide. http://guide.flossmetrics.org/index.php/6._FLOSS-based_business_models, 2009.

[2] Flossmetrics. start [Melquiades Wiki]. http://melquiades.flossmetrics.org/wiki/doku.php, 2009.

[3] freedesktop.org. freedesktop.org - home. http://www.freedesktop.org/, 2010.

[4] LibreSoft. start [GSyC/LibreSoft tools]. http://tools.libresoft.es/, 2010.

[5] Simón Pena. e-Motion: Covering your mobility needs. https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/isl/business-model/?root=freeswmaster, 2010.

[6] Simón Pena. Eye of Gnome - Most Valuable Member. a proyect preview. https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/dlsc/eog_exercise/beamer/?root=freeswmaster, 2010.

[7] Simón Pena. Open Source Software & Mobile Devices. https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/isl/mobile-os/report/?root=freeswmaster, 2010.

[8] Simón Pena. WebKit Project – A FLOSS report. https://forge.morfeo-project.org/plugins/scmsvn/viewcvs.php/trunk/spenap/dlsc/webkit_analysis/?root=freeswmaster, 2010.

[9] MASTER ON FREE SOFTWARE. Practical Approach: Analysing Libre Software Communities. http://gsyc.escet.urjc.es/moodle/mod/resource/view.php?id=2961, 2009.

[10] WebKit. The WebKit Open Source Project. http://webkit.org, 2010.

[11] Wikipedia. WebKit – Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/WebKit`, 2010.