Contribution to Asterisk Open Source Project

Universitat Oberta de Catalunya (UOC)

Master Oficial de Programari Lliure
Especialitat Desenvolupament d'Aplicacions de Programari lliure

Final Master Thesis

Sergio González Martín
Consultor: Gregorio Robles
Barcelona, June 2009

Abstract

With this final master thesis we are going to contribute to the Asterisk open source project. Asterisk is an open source project that started with the main objective of develop an IP telephony platform, completely based on Software (so not hardware dependent) and under an open license like GPL. This project was started on 1999 by the software engineer Mark Spencer at Digium. The main motivation of that open source project was that the telecommunications sector is lack of open solutions, and most of the available solutions are based on proprietary standards, which are close and not compatible between them.

Behind the Asterisk project there is a company, Digum, which is the project leading since the project was originated in its laboratories. This company has some of its employees fully dedicated to contribute to the Asterisk project, and also provide the whole infrastructure required by the open source project. But the business of Digium isn't based on licensing of products due to the open source nature of Asterisk, but it's based on offering services around Asterisk and designing and selling some hardware components to be used with Asterisk.

The Asterisk project has grown up a lot since its birth, offering in its latest versions advanced functionalities for managing calls and compatibility with some hardware that previously was exclusive of proprietary solutions. Due to that, Asterisk is becoming a serious alternative to all these proprietaries solutions because it has reached a level of maturity that makes it very stable. In addition, as it is open source, it can be fully customized to a given requirement, which could be impossible with the proprietaries solutions.

Due to the bigness that is reaching the project, every day there are more companies which develop value added software for telephony platforms, that are seriously evaluating the option of make their software fully compatible with Asterisk platforms. All these factors make Asterisk being a consolidated project but in constant evolution, trying to offer all those functionalities offered by proprietaries solutions.

This final master thesis will be divided mainly in two blocks totally complementaries. In the first block we will analyze Asterisk as an open source project and Asterisk as a telephony platform (PBX). As a result of this analysis we will generate a document, written in English because it is Asterisk project's official language, which could be used by future contributors as an starting point on joining Asterisk. On the second block we will proceed with a development contribution to the Asterisk project. We will have several options in the form that we do the contribution, such as solving bugs, developing new functionalities or start an Asterisk satellite project. The type of contribution will depend on the needs of the project on that moment.

Contents

1. Introduction

The Open Source community has been growing up the last years with a great quantity of projects being created and some of them becoming a reference on their own areas of interest. Some examples of these open source leading projects are Apache as the most widely used web server, PHP reaching an important quote of web pages developed or Eclipse being one of the integrated development environment (IDE) most widely used. All these projects have been growing thanks to their contributors and the community of persons that they have behind.

Another one of these projects which is growing very fast the last years is Asterisk. The Asterisk project is battling in a field, the telephony platforms, which, as a telecommunication field, has a large tradition of proprietary solutions and platforms, so it is an area where an open source project has an special interest. In this final master thesis we will analyze the project and proceed with a contribution to the Asterisk project to study all the aspects regarding the contribution to an open source project.

1.1. Objectives

This final Master thesis is defined as the last subject (PFM) of the Master of Science in Free Software of the Universitat Oberta de Catalunya [1]. For that reason, we will try to apply all, or at least a part, of the knowledge gained during the different subjects of that Master. To achieve that we will establish some initial objectives which we will try to accomplish during the development of the current thesis. Those initial objectives are:

- Study the internal operation of an open software community.
- Study the steps to be followed to start a collaboration with an open software project.
- Study the architecture and code of the Asterisk IP Private Branch Exchange (PBX).
- Write a document which could serve to future Asterisk contributors as an introductory manual, which will ease the task of introducing themselves on the Asterisk project. This will be a part of the current document.
- Taking advantage that Asterisk is an open source product, we will execute

an analysis of the source code to establish a base for future contributions.

- Do several contributions to the project in the way that we find more suitable, once we have deeply analyzed the Asterisk project.

This last objective is the main final objective of this final master thesis, which is reflected on the title of the thesis, but to accomplish this objective, all the rest objectives must be accomplished.

1.2. Motivations

As this final master thesis is conceived as the summarizing subject of the Master of Science on Free Software, it should make use of most of the competencies acquired during the study of previous subjects by developing a complete work. Regarding the thematic of this thesis, as most of the elective subjects that I have coursed are of a technical thematic, such as software development or networking, this will be mainly a technical application project. Due to the nature of the Asterisk PBX, this thesis will cover the following areas of interest:

- Free Software, as Asterisk is an open source community.
- C language software development, as our main contribution will consist in coding several issues, such as bugs or new features, for the Asterisk PBX, which source code is written in C.
- IP Telephony, as Asterisk is an IP based PBX, we will take contact with some IP telephony related technologies such us protocols and codecs. It will we interesting to distinguish which of them are open and which are proprietaries.
- Networking, as Asterisk is an IP based PBX, underlying it will be the networking layer which will interfere in the correct function of the PBX.

A part from the academic motivation, I also have a professional motivation. Nowadays I work in a company [3] that develops a software suite for Call/Contact Centers and ACD (Automatic Call Distribution) environments. I take part of the development of a proxy which allows the communication between the rest of applications of the suite, which originally were developed to work with proprietary PBX (Avaya, Nortel and Ericsson), with an Asterisk PBX environment. For that reason, gaining a deep knowledge of the Asterisk PBX will be a help for my professional work.

1.3. Scope

In this thesis the main objective is to contribute to the Asterisk open source project by developing some code for the project. But, in order to achieve that, there are several tasks that must be completed previously. First of all, we need to perform a deep analysis of the open source project and its community with the objective of getting a first contact with it and to learn how the community functions internally. Two points of special interest will be discovering all the tools available to work with the Asterisk project and how should them be used.

Once we have studied the community itself the next step will be the study the product produced by it, that is, the Asterisk IP PBX itself. We will study its architecture, the way it is configured to work properly and, the most important part, the source code which, being an open source project, is freely accessible.

Once we have studied all those technical aspects of the project we will focus on introducing ourselves in the Asterisk community, it is all the required steps to do a contribution to the project for all the possible types of contributions available (developing code, testing functionalities, documenting, ...). From this point and all the aforementioned, we will construct a section on the thesis document which will be intended to be used by future contributors to ease its introduction to the Asterisk project. It's due to that reason that we have chosen an open license, like is the Creative Commons license, to the current document so it will be freely accessible and modifiable by future contributors, and it has been written in English language since it is the official language of the project.

Finally, the last point regarding this final master thesis, will be a contribution to the project, which, as it is the title of the thesis, will be the main objective of it. As the contributions will depend on the needs of the Asterisk project, we cannot define them at this point of the project. The only that we know is that, for coding contributions, there exists some projects called *Janitor Projects* [4] that serve as an introductory point on coding for Asterisk PBX. These project basically are some code maintenance tasks, so don't have a critical risk for Asterisk. So the first step of our contribution will be selecting one of these projects a solve it. Then, once we have developed the whole project, we will need that someone with write permissions on Asterisk's Subversion commits our code to the trunk. That's due to by default a new developer won't have write permissions to the repository. It only will be granted once the developed has proved its value by developing some contributions of good quality. Then we will study the Asterisk's needs of the moment and will choose which will be our next contributions based on that.

1.4. Expectations of the Contribution

Before starting the study of Asterisk, we will define what are our expectations from contributing to an open source project as is Asterisk, to check later, when we have done the contribution, if we have accomplished them.

First of all we expect that, due to the open source nature of the project, we will have an easy access to all the required information about the project which we allow that we can introduce ourselves to the project. Another point that we expect is to have a direct contact with the people of the community which will help us in this task of introducing to the project.

Once we have introduced ourselves to the project we expect to have direct contact with the project developers because we will need help to define the kind of contribution that we will do. In that point, we also expect a direct communication with the project developers in order to guide us in the way in that our contribution is developed.

1.5. Document Structure

On next sections we will proceed with the contribution to the Asterisk project. First of all we will describe the Asterisk project, describing its structure, the procedures and methodologies defined on it, the available infrastructure and the different ways of contributing with the project as well as the steps to follow to start a contribution with Asterisk open source project.

Next we will introduce us on the project main product: the Asterisk PBX. In this section we will analyze Asterisk as a Software product and as an IP telephony platform. As a Software product we will study its code, how it's organized, which functionalities are currently developed and how the different parts are connected through its internal architecture. As a telephony platform we will study how it works, how is the configuration process and which compatibilities and supports currently offers.

Following, once we have studied the Asterisk community and we have some knowledge about the Asterisk PBX and its code, we will proceed with some contributions to the Asterisk project, trying to develop some fixes and/or new features required by the project. To achieve that first of all we will join the community and start a first contribution through the development of one of the available *Janitor Projects*.

Finally we will evaluate our contribution exposing some conclusions about it, checking if our initial expectations have been fully or partially accomplished.

2. The Asterisk Open Source Community

Telecommunications is probably the last major electronic industry that has remained untouched by open source. Manufacturers still build solutions very expensive, incompatible with other manufacturers' solutions, of a complicated configuration and running on an obsolete hardware. To avoid all those problems, the Asterisk project was started.

The Asterisk open source project was started by Digium's engineer Mark Spencer, who nowadays is still leading the Asterisk's community. One of the majors achievements of the Asterisk project is that it has been generated among technology professionals, networking professionals and information professionals who, while traditionally have been at odds with each other, in the Asterisk's community they share they efforts. The members of the Asterisk community are generally welcoming new users but as the project is growing up so fast and many people is gaining interest on it, they become tired of being asked about that kind of questions whose answers can often be obtained independently. So self-teaching skills are expected for new contributors of the project.

New users don't fit any particular stereotype. While some will spend hours researching and experimenting with Asterisk, others are uninterested in such issues and want a simple way to get Asterisk set up and running. Inside the community, there are people with different skills sets and attitudes, so in order they can work together for the best profit to the Asterisk project, there are some tools and procedures required and offered by the project.

2.1. Project Tools

At this point, we are going to analyze all the tools offered by the project to its community to help them in the task of contributing to the project. All that community tools are maintained by Digium. We will also study the tools that we will need to have installed on our local machine when contributing to the Asterisk project. We will require all these tools, those which are local and those which are accessible remotely via web access, to continue with the development of this thesis and complete our main objective which is the contribution to the Asterisk project.

2.1.1. Web

The Asterisk project is centralized on its web [2]. From that web page we have an easy access to all the rest of the tools provided by the open source project. There is also a lot of information about the project and its structure and operation. It's mainly divided in six areas, accessible through a corresponding button:

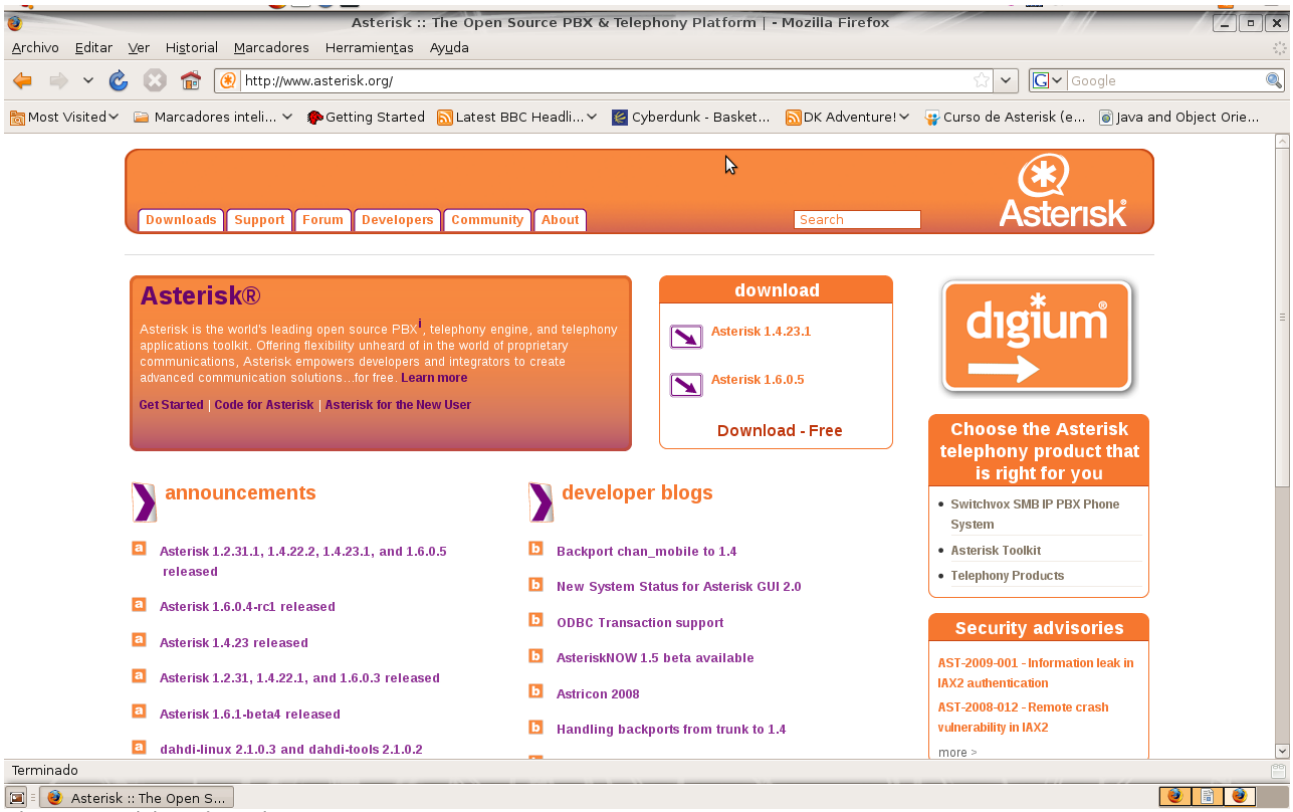

Fig. 1: Asterisk main webpage

- Downloads: From that area we have access to download all the packages related to the project. There are available those of the last stable version, the beta version and release candidate, and all the previously stable version.

- Support: In this section we have access to helpful documentation about Asterisk PBX which should be consulted prior to reporting a bug or asking a question to the community.

- Forum: A direct access to the project's official forums.

- Developers: From this section we have access to helpful information for Asterisk developers. There is defined the way in the Asterisk code is

maintained, how can it be accessed and modified and some guidelines to keep in mind when developing for Asterisk

- Community: In this section are located all the ways in which we can contact with the Asterisk community
- About: In this section some miscellaneous information is provided.

2.1.2. MailMan

The Asterisk community has an e-mail distribution list tool with MailMan [6]. Within those lists some aspects of Asterisk are discussed, depending on the thematic of the list. There are defined several distribution lists, but probably the more interesting are the following:

- asterisk-users: This list is the most used of all. There are discussed a lot of thematics on it, like configuration aspects, integration with third parties or developments around asterisk (but not inside it).
- asterisk-dev: In this list there are discussions regarding the development of Asterisk.
- asterisk-bugs: In this list are discussed the solution of these bugs that aren't of trivial resolution.
- asterisk-commits: In this list there are automatic generated e-mails when a commit is done to the asterisk project on subversion.

For our proposals, the asterisk-dev list is the list that is more interesting to be subscribed to than the others, although the asterisk-users list could be helpful while we study how asterisk works and how should it be configured. Despite this, we will only subscribe to the asterisk-dev list because it is of special interest for our final master thesis and if we needed any information regarding any other list, the archives of all the lists are accessible from [6]. So we will be able to consult any message of any lists if were needed.

To subscribe ourselves to the asterisk-dev, first of all we need to go to [6] and select the corresponding list. Then we have to introduce, in the corresponding field, the e-mail address where we want to receive the e-mails from that list. In that process we have the interesting option of receiving digest e-mails instead one copy of all the e-mails. That's interesting for disk space option, and more due to the fact that the whole list archive is freely accessible, so if we need additional information we can find it. Once we have submitted that information, we will receive a confirmation e-mail that is sent to confirm the authenticity of the given e-mail address. That e-mail contents a link which serves as confirmation of our e-mail address, and once we have accessed to that link we

will start receiving e-mails from the list. Another option to confirm address is by replying the sent e-mail without modifying the subject field, since it contents an identifier number.

2.1.3. Subversion

The source code of the asterisk project is maintained through a Subversion repository [7]. In that repository are maintained the code of the main Asterisk project and the source code of other Asterisk related project and some projects “satellite” to Asterisk.

On next chapter we will study the source code organization of Asterisk more in depth. We can have anonymous access to it in read only mode to checkout the code and review it, but we won't be able to commit our code to Asterisk's Subversion since the anonymous user doesn't have that kind of privileges. If we want to commit our changes, first of all we need to open an issue in the issue tracker (see section 2.1.4) and an senior Asterisk's developer with write privileges on Subversion should review it and commit it if convenient.

Prior to gain the write privilege to Subversion repository, we need to prove our value by doing several contributions of proved quality to the project. Once done that, which is decided by the project leaders, we will have access to an Subversion user with write privileges and we will be able to commit our changes ourselves.

2.1.4. Mantis

All the bugs and new features are tracked through Mantis [8]. To be able to report new issues, first of all we need to create a user. To do that we only have to provide an user name and a valid e-mail account. Then a confirmation e-mail is sent to the provided e-mail account in order to validate it. Done that, we have completed access to Mantis and we can report new issues and consult any existent issue.

The Asterisk's bug tracker is used to track bugs and documentation elements that need to be discussed in a permanent, semi-threaded manner and which can more easily store programmatic or textual difference files ("*diff -u*") in a manageable way. The Mantis bug tracker is designed to allow the user community to work on different issues so that the brunt of the work is moved from the shoulders of the small development staff onto the more distributed group.

The primary use of the bug tracker system is to track bugs, where "bug" means anything that causes unexpected or detrimental results in the Asterisk PBX. The secondary purpose it's to track some of the miscellaneous issues surrounding Asterisk, such as documentation and commentary.

2.1.5. Review Board

Asterisk project offers a tool for reviewing the source code developed for some new features or resolutions for bugs with Review Board [9]. With this tool, the new code developed when resolving non trivial bugs or implementing new features, can be published in a user friendly way and other developers can review the code and make any suggestion if needed. For a better communication purpose, this tool is connected with the asterisk-dev mail distribution list so every time a new review is posted, a new e-mail is sent to the distribution list so all the Asterisk's developers are informed. This tool is still in a beta state since it has been recently implanted.

To access to this tool we need a user name and password which is the same as that created for accessing Mantis. But before we can access, we need to read and sign the *Digium Open Source Software Project Submission Agreement* (see appendix B) which is a license agreement according to which Digium has license to use, reproduce, modify or redistribute any work submitted to the project.

2.1.6. IRC Channels

The Asterisk community also needs a way to communicate instantly and it's granted with IRC Channels. In the IRC server Freenode [9], there exists some groups in which different Asterisk's communities, such us developers or users, can dialogue about the project. The official language for all the channels, except for those language specific, is English. The channels that currently exists at Freenode related with the Asterisk project are:

- #asterisk: Channel for general asterisk issues. This is the place where help can be found for using and configuring Asterisk. This channel will be of special interest in this final master thesis when we will be introducing ourselves to the Asterisk PBX (see next section).

- #asterisk-br: Channel for the Brazilian Asterisk community where aspects regarding Asterisk can be discussed in Brazilian.

- #asterisk-bugs: Channel to discuss if a reported issue is really a bug and to discuss the solution of it. This channel is directly connected with the Mantis system so every time a bug is reported or a bug is solved an automatic message is sent to this chat.

- #asterisk-commits: This channel is directly connected with the Subversion system so every time a new commit is done or a merge conflict appears, a new message is sent to the channel.

- #asterisk-cpp: This is a channel for the Asterisk C++ project, which is a project that is trying to adapt asterisk code to C++.

- #asterisk-dev: This is the Asterisk development channel. It is the place to discuss development issues such us new features for the project or the resolution of non trivial bugs. This channel will be of special interest in this final master thesis when we will be performing the contribution to the Asterisk project. It's connected with Subversion so with certain messages sent to the channel, a bot is called that can make some queries to Subversion. For instance, if there is a discussion about a certain commit number, it can be consulted through a message which return the description associated to the commit.

- #asterisk-doc: This is the place where documentation issues are discussed.

- #asterisk-es: Channel for the Spanish Asterisk community where aspects regarding Asterisk can be discussed in Spansih.

- #asterisk-gui: This is the channel to discuss issues about the Asterisk-GUI project, which is a graphical interface for administering Asterisk.

- #asterisk.de: Channel for the Deutsch Asterisk community where aspects regarding Asterisk can be discussed in German.

- #asterisknow: In this channels is intended for discussing issues about the AsteriskNOW project, which is a customized Linux distribution which includes Asterisk ready to use.

- #asteriskru: Channel for the Russian Asterisk community where aspects regarding Asterisk can be discussed in Russian.

Once we have reviewed the different IRC channels available at Freenode, we will need to create a user in order to access them, specially the aforementioned #asterisk and #asterisk-dev which are the most interesting for our purposes. To do that, we need to access via an IRC client to the Freenode server and send a messages some messages. First of all, with (1) we connect our client to an IRC server, which in this case is Freenode server. Then, we need to select a nickname (2) that we want to register and use later. This nickname shouldn't be previously registered because it must be unique. If the nickname is available

the server will response us remarking that. Then we need to send a message (3) to register definitively the nickname giving a password and an e-mail address. After that, a good option could be to set hidden our e-mail address, which could be done with message (4), so it isn't visible for the rest of the users. And finally, the last step will be identifying ourselves with the nickname and password previously registered with (5), and once we are inside Freenode we can join the IRC channels of our interest.

```
/server irc.freenode.net   (1)

/nick OurNickname   (2)

/msg nickserv register <your-password> <your-email>   (3)

/msg nickserv set hidemail on   (4)

/msg NickServ IDENTIFY <your-password>   (5)
```

2.1.7. Wiki

Although it isn't Asterisk project specific, it exists a wiki [11] which is considered the official wiki of the project. This wiki include a lot of information about VoIP (Voice over Internet Protocol) in general, and Asterisk in particular. Prior to this wiki, there existed an Asterisk documentation web page [12] which is intended to be the official documentation page for the Asterisk project, but nowadays it's unavailable due to a reconstruction of the site. Associated to the Asterisk documentation page, there exists a book [5] which is considered the *bible* of the project, a must-read for every new Asterisk community member.

2.1.8. Forums

There also are some forums offered by Digium [13], where different information related with Asterisk can be found. It's a place where we can post any problem regarding the Asterisk configuration and the response will be public so anyone with the same problem will have the solution accessible.

2.1.9. Knowledge Base (KBPublisher)

This tool [14] is a searchable library of troubleshooting advice and current information updated by Digium support technicians. In this library we can found how some common configuration errors can be corrected and how a certain hardware should be configured in order to work properly with Asterisk.

2.1.10. Blogs

The last tool offered by the Asterisk project, are the developers blogs [15]. In those blogs developers explain their ideas for new developments on Asterisk and their current developments for the project.

2.1.11. Local Tools

Finally we will analyze the tools that we will need to have installed on our local machine in order to contribute to the Asterisk project in the frame of our final master thesis. This will be the environment that we will use to work with Asterisk and to complete the current final master thesis. Those indispensable tools are:

- Linux distribution: Ubuntu Desktop 8.10. We will use a distribution with a x-window environment based on Gnome, as it is Ubuntu.
- Web Browser: Mozilla Firefox 3.1. This will be required to access all the project tools aforementioned.
- Integrated Development Environment (IDE): Netbeans IDE 6.1. To analyse the code and develop new code we will use this IDE which, despite it was created to work with Java, it has got a good plug-in to work with C/C++.
- Office application suite: OpenOffice.org 2.4.: To create documentation during this final master thesis.
- Subversion client: Netbeans plug-in. To access to the Subversion repository and get the last version of the code available for Asterisk, we will use the plug-in of Netbeans which will ease the task of updating the code and accessing to previous versions of the source code.
- Development libraries: In order to compile Asterisk we need to have

installed some development libraries.

- Project Planning tool: Planner 0.14.3. To keep the planning of the final Master thesis we will require this tool.
- IRC chat: Xchat-GNOME 0.24.1. To have a graphical access to the Freenode IRC server we will use this tool.

2.2. Established Procedures

Once we have studied all the tools offered and required to work with the Asterisk project, we will describe all the procedures defined within the project. All those procedures are public and can be consulted on the project main web page. Those procedures basically define the steps to be followed when doing a common action with the community, such as reporting a new bug or committing a new code development.

2.2.1. Getting Started

When someone wants to introducer himself to the Asterisk PBX, there exists an established procedure which consists of the following steps:

1. Choose the Asterisk version that best fits our needs.
2. Learn Asterisk architecture and features by reading some given resources.
3. Download and install the chosen version of Asterisk.
4. Learn Asterisk's configuration aspects by reading some given resources.

2.2.2. Coding for Asterisk

When we are going to code for Asterisk, having knowledge in a few key areas regarding coding for Asterisk will be extremely helpful in getting the code committed to the SVN repository for the Asterisk project (as initially we won't have write access to the respository). We should follow some steps described in

this project' standard procedure:

1. First of all we need to get the latest version of the source code of Asterisk.

   1.1 Optional: If we have been granted with a workspace in SVN, we will need to configure it in order to be able to commit our work to it.

2. Review the code structure to familiarize with it.

3. Build asterisk to check that we can compile it properly, which will be needed when developing for Asterisk. To do that we will need to have installed *ncurses*, *openssl, zlib* and their associated *devel* packages.

4. Prior to start writing code for Asterisk, we need to read the coding guidelines document, which is available on Subversion repository of Asterisk, and which explains the standard formatting that we will need to implement when writing code for Asterisk.

5. Once we have learned the rules for coding for Asterisk, we need to decide what will be our contribution. If it's our first contribution, the Asterisk project has defined several projects, that are called “Janitor Projects” and are intended to be a nice starting point for new developers. Those projects basically cover several code maintenance tasks, which let new developers to take a first code with the Asterisk source code. On the other hand, if we are experienced contributors our contribution could be any.

6. During the process of developing the contribution, if it's a non trivial development, we will need help from the rest of the development community to solve it in the best way for the project. In this case we will need to request a code review (using the code review tool showed on section 2.1.5) from the rest of asterisk developers.

7. Then, when we have finished our contribution, it will be the point of having it reviewed by the community. As we won't have write access to the Subversion repository, we will need to create

Every new development submitted to Subversion has associated a license agreement between the developer and Digium which grant the redistribution rights to Digium. The developer maintain its copyright over the source code developed. This way of work is very similar to that followed by the MySQL project.

2.2.3. Testing Asterisk and Reporting Issues

Once a new feature o bug patch development has been finished for the Asterisk project, exhaustive testing is required before it could be committed to the Asterisk's SVN repository. However, SVN maintainers and Bug Marshals do not have the time to test every patch or new feature. For this reason the project requires also contributors on performing this task. Working relationships are developed among Asterisk users on the `#asterisk-dev` and `#asterisk-bugs` IRC channels and mailing lists so that patches can be tested thoroughly.

Once a bug has been found, it's defined a procedure to follow for create a new issue on Mantis (seen at section 2.1.4) trying to minimize the time in which it's solved:

1. Extract SVN revision number through command *svn info*: If this is not the most updated SVN version, we should get the latest version of the code and check again if the problem still occurs. We must do that because this bug could be already fixed on the latest version.

2. Get platform (O/S): Provide the distribution and version of software you are currently running ("Linux Redhat 9.0" as an example).

3. Reproducing the problem: We should be able to explain the steps to follow to reproduce the problem. If it isn't really repeatable we'll need to be extremely detailed in giving our configuration notes and including as much debug information as possible, as non-repeatable problems tend to be almost impossible to guess resolution types.

4. Getting debugging output: Include all the outputs from various traces, debugging, etc, as attachments and not as pasted text in the bug report. The bug tracker does funky things with line wrapping, etc. and an attached file makes more sense.

5. Make sure the bug doesn't already exist on Mantis: This may take some time, but duplication is a real problem to the project. In addition, reading the other bugs we might find something useful.

6. Open bug on Mantis: To do that we will need to provide the following information:

    6.1 Naming the report: It's very important to name the issue appropriately. A detailed and good short description will help the bug marshals and developers to browse through the bug tracker and find your issue quickly. It is common use on the Asterisk bug tracker to put the text (minus quotes, but including brackets) of `"[patch]"` as the first part of a bug short description when the particular bug includes a source code patch. This helps the developers sort things appropriately

(we'll let you guess which bugs get worked on first).

6.2 Categorize the issue: We must add our issue into the proper category because it ease developer's work.

6.3 Use polite language: When describing the issue, we should use complete sentences, capitalization, and avoid slang. This specially applies to non-native English speakers who may have picked up bad habits from IRC channels.

7 Commit the new bug: Once we have added all the information, we only need to submit it and the bug would be created.

Following the described procedure our bug will be submitted to the bug tracking system and then it will be studied and solved by Asterisk's developers.

2.2.4. Project Recommendations

Finally, there are also defined some considerations that should be followed for achieving a better quality of the Asterisk PBX.

- What will slow down my bug/patch from being looked at:
  - No patch code (for features or some bugs)
  - Poor descriptions
  - No back trace or debug information (in crash instances)
  - No follow up to questions by others
  - Not having a contributor license agreement on file with Digium
- What will speed up my bug/patch being implemented?
  - Testing by 1 or more others to reproduce events or use patch
  - Good discussion by others in the bug notes
  - Clear C code with excellent comments
  - Clear debugging packet traces (for protocol-level VoIP issues)
- What does the Asterisk community need?
  - More testing by everyone to keep developer head-scratching time to a

minimum. Testing award you karma points in the issue tracker.

2.3. Community Key People

Finally we will present some people that have a key role inside the Asterisk project:

- Mark Spencer: He is the founder of Digium, Inc., in 1999 as Linux Support System while he still was a student of computer engineering at Auburn University. He used his Linux and C code knowledge to develop his own PBX, what was the beginning of Asterisk. As Asterisk gained popularity, Spencer shifted his business from Linux support to supporting Asterisk and opening up the telecommunication market. Today he is the Chairman and CTO of Digium. He is widely regarded as the pioneer of open source telephony and gives frequent keynote addresses to large technology audiences.

- Russell Bryant: He is a Software Engineer graduated from Clemson University in the Fall of 2006. Since 2004 he has been a core member of the Asterisk development team. He became Asterisk's first release mantainer when Asterisk 1.0 was released at the first Astricon in 2004. Since then he has contributed to almost all areas of Asterisk development. In 2005 he was hired by Digium where he works up today.

- John Todd: He is the Asterisk's community manager at Digium. His main responsibilities are the management of all the community tools, such as Subversion repository or Mantis bug tracking.

- Joshua Colp (file): He is a software developer and a core developer of Asterisk. He's originally from Nova Scotia, Canada but currently live in New Brunswick. He works full time at Digium developing for Asterisk. Among other developments, he has developed VLDTMF (Variable length DTMF), the speech recognition API or non-blocking Logger. calls either synchronously or asynchronously to a device with the same feature set available to all other applications.

2.4. Asterisk Project Events

Related to the Asterisk community, there has been established some events where people related with the Asterisk project can met together. The most

important among them is the AstriCon, a three day conference about Asterisk which is celebrated yearly. AstriCon's mission is to expand awareness and knowledge of Asterisk. Asterisk developers and experts gather from miles around to become a part of AstriCon, the first official Digium sanctioned Asterisk Event.

Since this year the Asterisk project has been added to the Google summer of code. The Google summer of code offer the possibility to students of working in some real world projects. This year, some of the Asterisk's core developers have offered to mentor some students while they develop their Google summer of code project on the Asterisk project.

3. The Asterisk PBX

In this section we will introduce the Asterisk PBX. To achieve this, first of all we will review the state of the art in the telephony PBX solutions fields. Done that, we will present the Asterisk architecture, how it's installed and configured. Finally we will review the code structure of the Asterisk project doing a brief introduction to some Asterisk satellite projects.

3.1. State of the Art

The Asterisk project was started as the first open source for IP telephony platforms and so there aren't any precedent of those platforms in the open source field. Regarding the open solutions we only find Trixbox, which started on 2004 as a fork from Asterisk (starting named as Asterisk@Home) with the main objective of developing an integrated solution for small office and home (SOHO) environments. So it is only a reduced version of Asterisk which can be run on computers less powerful.

We also find other open source projects that, although aren't designed to work as PBX and with no so many functionalities as actually offers Asterisk (as it is a mature project), they are presented as alternatives started by developers that weren't agreed with some decisions taken by the Asterisk project. Those projects still require a lot of development to be done in order to achieve an state of maturity similar to that of the Asterisk project. Some of these projects are:

- FreeSwitch [16]: This project is formed by some libraries which offer a communications platform. We could obtain a PBX based on those libraries which could be an alternative to Asterisk but all this development is still to be done.

- SipX (SIP PBX for Linux) [17]: It is an IP PBX which offers features very similar to Asterisk but with different implementations. It is probably the open source alternative closest to Asterisk in functionalities despite its community and its importance is lesser. SipX cooperates with the FreeSWITCH project since uses libraries from it for its multimedia server.

- OpenPBX [18]: This project, started inside the Voicetronix enterprise, offers a PBX solution with a very easy configuration and a fast deployment. The source code is very compact since it only is formed but

over 1000 of Perl code lines. Due to its reduced size, it should be compared with the Trixbox project instead the Asterisk project.

- PBX4Linux [19]: It is an ISDN PBX which connects ISDN phones, ISDN lines and H.323 gateways. Since it is limited to the ISDN technology, its possibilities are by far lesser than those of Asterisk PBX. It can be considered a dead project because there hasn't been any new release since 2005.

- CallWeaver (originally OpenPBX.org) [20]: Started as a fork of the Asterisk project but it hasn't achieved so many importance as the Asterisk projecte. Due to that, the development done in this project has been lesser, so this PBX offers less functionalities than Asterisk.

As we have aforementioned, the only alternatives to Asterisk being capable of offering a similar set of functionalities are proprietaries solutions. Those solutions aren't completely software and require of some kind of specific hardware to its functioning, and in most cases this hardware is completely not compatible among different vendors. So an important part of the cost of the proprietary PBXs, in addition to the licenses cost, is due to the price of the specific hardware required by the proprietary solution. Some of the vendors with a big market quote are Avaya, Nortel and Cisco.

The Asterisk project was started on 1999 (version 0.1) by Mark Spencer at LSS (currently renamed to Digium) due to the lack of an open source quality solution on the PBX field. It become popular on year 2002, and since then it has been growing in popularity and improving its performance and features offered. There has been four stable major version launched up to nowadays, which are:

- 1.0: launched on 2002 and the last version has been 1.0.12.

- 1.2: launched on 2005 and the last version has been 1.2.31.1.

- 1.4: launched on 2006 and the last version published is 1.4.25.

- 1.6: launched on the second half of 2008 and is the current trunk base version.

Asterisk is presented as a quality telephony solution, which hasn't got attached the purchase of any license neither the purchase of any specific Hardware, but it let choose the hardware more suitable for the requirements of every environment, in addition to allow, due to its open source nature, doing modifications in the source code to have a customized version of Asterisk.

In addition to the project, Digium offers additional version of Asterisk which require a license payment. Those solutions basically are based on the Asterisk open source PBX but offer support and additional services to it associated to this licensing.

3.2. Asterisk PBX Architecture

The Asterisk PBX has changed from previous versions, specially a big architectural change was done from version 1.2 to version 1.4. As we will work version over the trunk version of Asterisk, which is 1.6, we will focus on this version.

Asterisk architecture [2, 5] is fundamentally very simple, but different from most telephony products. Essentially Asterisk acts as a middleware, connecting telephony technologies on the bottom, to telephony applications on top, creating a consistent environment for deploying a mixed telephony environment. Telephony technologies can include VoIP services like H.323, SIP, IAX and MGCP (both gateways and telephones), as well as more traditional TDM technologies like T1, E1, ISDN PRI, POTS and PSTN services, Basic Rate ISDN (BRI) and more. Telephony applications include features such as call bridging, conferencing, voicemail, auto attendant, custom IVR scripting, call parking, intercom and more.

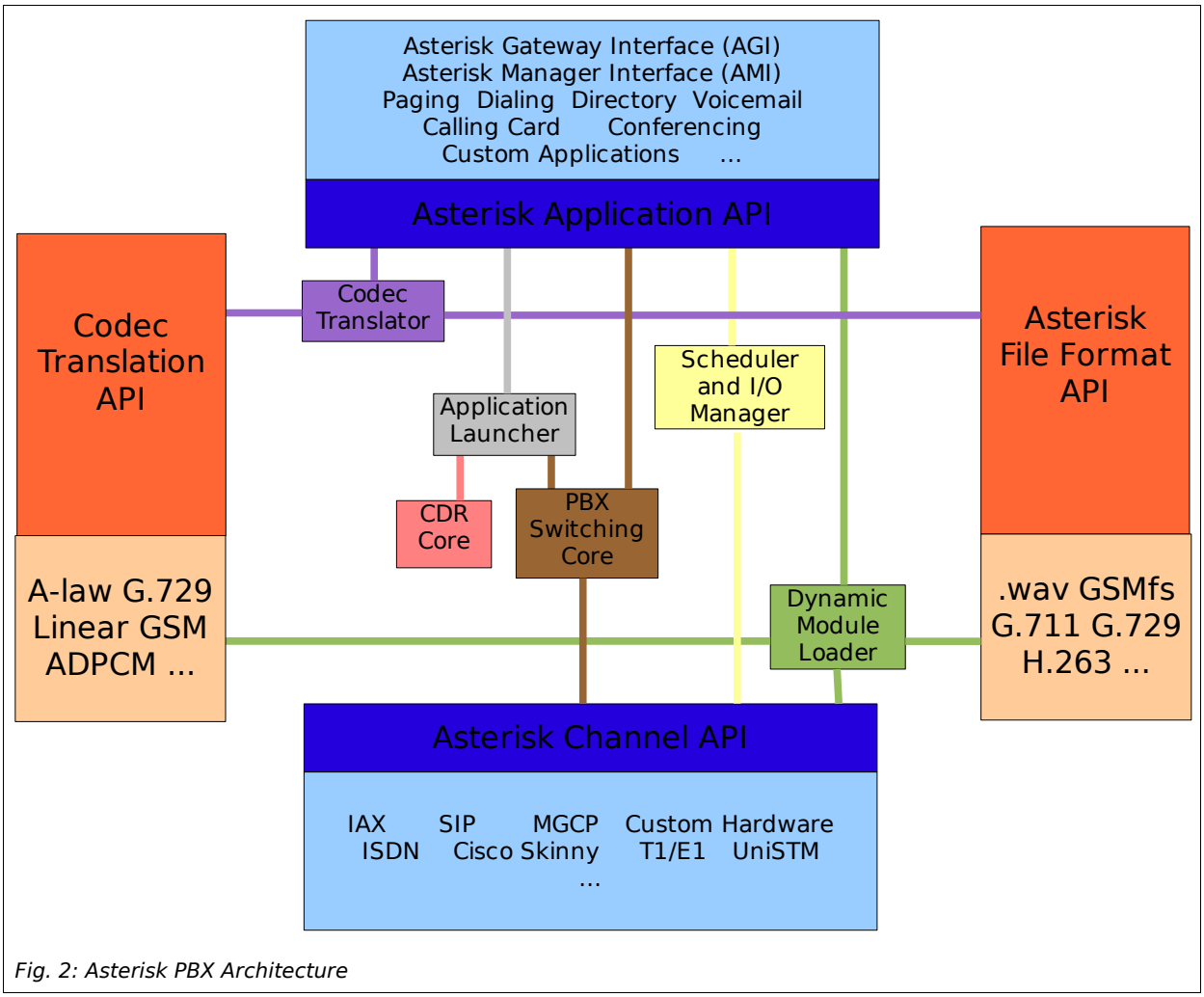

Fig. 2: Asterisk PBX Architecture

Asterisk has been carefully designed for maximum flexibility. Specific API (Application Programming Interface) are defined around an advanced, central PBX core system. The advanced core handles the internal interconnection of the PBX, cleanly abstracted from the specific protocols, codecs and hardware interfaces from the telephony applications which allows Asterisk to use any suitable hardware and technology available now or in the future to perform its essential functions - connecting hardware and applications.

Using these APIs, Asterisk achieves a complete abstraction between its core functions as a PBX server system and the varied technologies existing (or in development) in the telephony field. The modular form is what allows Asterisk to seamlessly integrate both currently implemented telephony switching hardware and the growing Packet Voice technologies emerging today. The ability to load codec modules allows Asterisk to support both the extremely compact codecs necessary for Packet Voice over slow connections such as a telephone modem while still providing high audio quality over less constricted connections.

The application API provides for flexible use of application modules to perform any function flexibly on demand, and allows for open development of new applications to suit unique needs and situations. In addition, loading all applications as modules allows for a flexible system, giving administrators the ability to design the best suited path for callers on the PBX system and modify call paths to suit changing communication needs.

Asterisk's core contains several engines that each play a critical role in the software's operation. When Asterisk is first started, the *Dynamic Module Loader* loads and initialize each of the drivers, which provide channel drivers, file formats, call detail record back ends, codecs, applications and more, linking them with the appropriate internal APIs. Then, the Asterisk's *Switching Core* begins accepting calls from interfaces and handling them according to the dialplan, using the *Application Launcher* for ringing phones, connecting to voicemail, dialing out outbound trunks, etc. The core also provides a standard *Scheduler and I/O Manager* that applications and drivers can take advantage of. Asterisk's *Codec Translator* permits channel which are compressed with different codecs to seamlessly talk to one another. Most of Asterisk's usefulness and flexibility come from the applications, codecs, channel drivers, file formats, and more, which plug into Asterisk's various programming interface.

3.2.1. Loadable Module APIs

Four APIs are defined for loadable modules, facilitating hardware and protocol abstraction. Using this loadable module system, the Asterisk core does not have to worry about details of how a caller is connecting, what codecs are in use, etc.

- **Channel API:** The channel API handles the type of connection a caller is arriving on, being it a VoIP connection, ISDN, PRI, Robbed bit signaling, or some other technology. Dynamic modules are loaded to handle the lower layer details of these connections.

- **Application API:** The application API allows for various task modules to be run to perform various functions. Conferencing, Paging, Directory Listing. Voicemail, In-line data transmission, and any other task which a PBX system might perform now or in the future are handled by these separate modules.

- **Codec Translator API:** Loads codec modules to support various audio encoding and decoding formats such as GSM, Mu-Law, A-law, and even MP3.

- **File Format API:** Handles the reading and writing of various file formats for the storage of data in the file system.

3.2.2. Asterisk Core

A part of the aforementioned APIs, the rest of required PBX operations are performed by the Asterisk core, in addition to the interaction with the APIs. The main features performed by the core of Asterisk are:

- **PBX Switching:** The essence of Asterisk, is a Private Branch Exchange Switching system, connecting calls together between various users and automated tasks. The Switching Core transparently connects callers arriving on various hardware and software interfaces.

- **Application Launcher:** Launches applications which perform services for uses, such as voicemail, file playback, and directory listing.

- **Codec Translator:** Uses codec modules for the encoding and decoding of various audio compression formats used in the telephony industry. A number of codecs are available to suit diverse needs and arrive at the best balance between audio quality and bandwidth usage.

- **Scheduler and I/O manager:** Handles low-level task scheduling and system management for optimal performance under all load conditions.

- **Dynamic Module loader:** It allows that all the modules of every API can be loaded and unloaded on real time.

3.3. Asterisk Installation and Configuration

When we decide to install an instance of the Asterisk PBX we have different options. First of all we need to decide which version of those available we want to install. Currently there are mainly three active branches of the project: the 1.6 branch (also the trunk of the project) which is the trunk version and it is where new features are developed; the 1.4 branch, which was the previous development branch, where some new features are developed and bugs are solved; and the 1.2 branch, where only critical bugs are solved. If we are wishing to have the last version of Asterisk with new features then we should try a 1.6.X version of Asterisk, but if we are looking for a more stable version of Asterisk we should choose either 1.2.X or 1.4.X version, although it's recommended to use 1.4.X versions over 1.2.X since it's architecture makes it more reliable.

Another election that we need to do is the way in we will install Asterisk. We have several option to do that. First of all there are pre-compiled packages for some distributions like Red Hat (RPM packages) or Debian (DEB packages) which ease the installation of Asterisk. Installing Asterisk in this way has the drawback that we won't have the maximum performance that we could have with our machine since it isn't compiled explicitly for our machine configuration. To avoid this drawback we should compile Asterisk from source code. This can be done by downloading a tarball with the selected version or obtaining the source code from Subversion repository.

There exist another option to have an Asterisk PBX installation that is probably the easiest way. It consist in an Asterisk related project, called AsteriskNOW [21], which consist in a customized GNU/Linux distribution (based on CentOS) which has Asterisk pre-installed and once we install the distribution on a machine we have Asterisk running properly.

In our case, we will use the trunk version (1.6 branch) of Asterisk, since our main objective of the final master thesis is to develop over it. And we will install it from source code obtained from Subversion, since the trunk version is only available from the Asterisk's Subversion repository.

Prior to install Asterisk PBX it's recommended to install some packages which contain drivers that we could need in the future and it's a good idea to install them from the beginning. Those packages are libPRI (libraries for PRI interfaces), dahdi-linux (drivers for the Digium Hardware for Asterisk) and dahdi-tools (tools for configuring the Digium Hardware drivers). Those packages can be also obtained from Subversion. Since our objective in this final Master thesis is to contribute only on the Asterisk main project, we won't need to install the Subversion's trunk version of those packages and we can get any release of them in a tar.gz format and compile and install them.

The steps that we will follow to install Asterisk on our environment will be the

following:

1. Obtain the source code from Subversion. This can be done with the command:

   `svn checkout http://svn.digium.com/svn/asterisk/trunk` **(6)**

   If we wanted to obtain any branch instead of the trunk version, we only need to change the trunk directory for that of the corresponding branch.

2. Check that we have all the required libraries and tools for compiling Asterisk with all its features since we don't know initially in which area of Asterisk will be our contribution. Those are the *gcc*, *gcc-c++* and *make* compilers and libraries *ncurses-devel, libtermcap-devel, kernel-devel, openssl-devel, unixODBC-devel, newt-devel, zlib-devel* and *libtool*, among others.

3. Configure the source code to our specific machine, by executing the following command from the path where we have downloaded the Asterisk source code:

   ./configure **(7)**

4. Compile the source code from the configuration done:

   `make` **(8)**

5. Installation of the generated binaries:

   make install **(9)**

6. Create a base configuration:

   `make samples` **(10)**

7. Run Asterisk. This must be done with root privileges:

   `sudo asterisk` **(11)**

Done that we will have Asterisk installed but without being configured. Asterisk PBX has a enormous number of interface types to which can connect. These include Analog interfaces (such as your telephone line and analog telephones), Digital circuits (such as T1 and E1 lines) or VoIP protocols (such as SIP and IAX). Asterisk doesn’t requires any specialized hardware (not even a sound card) even though it is common to expect a telephone system to physically connect to a voice network. There are many types of channel cards that allow us to connect your Asterisk to things like analog phones or PSTN circuits, but they are not essential to the functioning of Asterisk. On the user (or station) side of the system, we can choose from all kinds of softphones that are available for Windows, Linux, and other operating systems, or use almost any physical IP

phone. On the carrier side, if we don’t connect directly to a circuit from your central office, you can still route your calls over the Internet using a VoIP service provider.

In order to achieve this operation on the PBX, we have to configure Asterisk properly. Asterisk configuration is based in some configuration files located on the /etc/asterisk directory. All these files have got a similar structure; basically they are text files which content some sections or categories (identified because they have []) and on every section there are a collection of parameter-value pairs. The most important files that we will need to firstly configure to have our Asterisk installation running properly are:

- **asterisk.conf**: This file defines the location for the configuration files, the spool and the modules as well the location to write the log files. It's recommended to use the default settings unless we know the implications of changing them. Basically it's formed by a [directories] section which defines the locations aforementioned, and a [options] section which can define some start up options.

- **modules.conf**: This file controls which modules are loaded or not on the start up. The file only contains a [modules] section in which we specify which modules must be loaded on start up and which mustn't.

- **manager.conf**: This file contains the configurations for the Manager Interface (AMI, see section 3.3.5) connections.

- **extensions.conf**: This file contains all the asterisk dial plan (see section 3.3.1).

- **sip.conf**: In this file the SIP protocol is configured. The authentication for endpoints, such as SIP phones and service providers, is also configured on this file. Asterisk uses this file to determine which calls are willing to accept and where those calls should go in relation to the dial plan.

- **iax.conf**: Similar to sip.conf, in this file the IAX2 (Inter Asterisk eXchange revision 2) protocol is configured.

- **logger.conf**: In this file log files are configured, specifying which files are generated and which level of log messages are written on every file.

- **rtp.conf**: This file controls the RTP (Real-time Transport protocol) ports that Asterisk uses to generate and receive RTP traffic. The RTP protocol is used by SIP, H.323, MGCP and possibly other protocols to carry media between endpoints.

Done that we will have our Asterisk PBX ready to use. The next step that we will review in order to have a complete knowledge of the Asterisk installation are the default directories related with Asterisk:

- **/etc/asterisk**: As we have seen, this directory contains the Asterisk

configuration files.

- **/usr/lib/asterisk/modules/**: This directory contains all of the Asterisk loadable modules. Within this directory are the various applications, codecs, formats, and channels used by Asterisk. By default, Asterisk loads all of these modules at startup. Any module can be disabled in the modules.conf file, but we must be aware that certain modules are required by Asterisk or are dependencies of other modules. Attempting to load Asterisk without these modules will cause an error at startup.
- **/var/lib/asterisk/**: This directory contains the *astdb* file and a number of subdirectories. The astdb file contains the local Asterisk database information, which is somewhat like the Microsoft Windows Registry. The Asterisk database is a simple implementation based on v1 of the Berkeley database. The db.c file in the Asterisk source states that this version was chosen for the following reason: "*DB3 implementation is released under an alternative license incompatible with the GPL. Thus, in order to keep Asterisk licensing simplistic, it was decided to use version 1 as it is released under the BSD license*". The subdirectories on this directory are:
  - **agi-bin/**: The agi-bin/ directory contains your custom scripts, which can interface with Asterisk via the various built-in AGI applications.
  - **firmware/**: contains firmware for various Asterisk-compatible devices. It currently contains only the iax/ subdirectory, which holds the binary firmware image for Digium's IAXy.
  - **images/**: Applications that communicate with channels supporting graphical images look in the images/ directory. Most channels do not support the transmission of images, so this directory is rarely used. However, if more devices that support and make use of graphical images are released, this directory will become more relevant.
  - **keys/**: Asterisk can use a public/private key system to authenticate peers connecting to your box via an RSA digital signature. If you place a peer's public key in your keys/ directory, that peer can be authenticated by channels supporting this method (such as the IAX2 channels). The private key is never distributed to the public. The reverse is also true: you can distribute your public key to your peers, allowing you to be authenticated with the use of your private key. Both the public and private keys—ending in the .pub and .key file extensions, respectively—are stored in the keys/ directory.
  - **moh/**: When you configure Asterisk for Music on Hold, applications utilizing this feature look for their MP3 files in the moh/ directory. Asterisk is a bit picky about how the MP3 files are formatted, so you should use constant bitrate (CBR) encoding and strip the ID3 tags from your files.
  - **sounds/**: All of the available voice prompts for Asterisk reside in this directory. The contents of the basic prompts included with Asterisk are

in the *sounds.txt* file located in the Asterisk source code directory. Contents of the additional prompts are located in the *sounds-extra.txt* file in the directory to which you extracted the asterisk-sounds package earlier in this chapter.

- **/var/spool/asterisk/**: The Asterisk spool directory contains several subdirectories, including dictate/, meetme/, monitor/, outgoing/, system/, tmp/, and voicemail/. Asterisk monitors the outgoing directory for text files containing call request information. These files allow the generation of a call simply by moving the correctly structured file into the outgoing/ directory. Call files being placed into the outgoing/ directory can contain useful information, such as the Context, Extension, and Priority where the answered call should start, or simply the application and its arguments. You can also set variables and specify an account code for Call Detail Records. The content of its subdirectories are:
  - **dictate/**: This directory is the default location where the Dictate() dial plan application looks for files.
  - **meetme/**: This directory is the location where MeetMe() dial application conference recordings are saved.
  - **monitor/**: Recordings from either one-touch recording (the w and W flags to the Dial() application), the MixMonitor(), or Monitor() applications are stored in this directory.
  - **system/**: This directory is used by the System() application for temporary storage of data.
  - **tmp**/: This directory is used, to hold temporary information. Certain applications may require a place to write files to before copying the complete files to their final destinations. This prevents two processes from trying to write to and read from a file at the same time.
  - **voicemail/**: All voicemail and user greetings are contained within the voicemail/ directory. Extensions configured in **voicemail.conf** that have been logged in to at least once are created as subdirectories of voicemail/.
- **/var/run/**: This directory contains the process ID (PID) information for all active processes on the system, including Asterisk (as specified in the **asterisk.conf** file). Note that /var/run/ is OS-dependent and may differ.
- **/var/log/asterisk/**: This directory is where Asterisk logs information. The type of information being logged to the various files can be configured by editing the **logger.conf** file located in the /etc/asterisk/ directory. Basic configuration of the **logger.conf** file
  - **cdr-crv/**: The /var/log/asterisk/cdr-csv directory is used to store the CDRs in comma-separated value (CSV) format. By default information is stored in the Master.csv file, but individual accounts can store their own CDRs in separate files with the use of the accountcode option (see Appendix A for more information).

The last point that we should know is how to run Asterisk PBX. Asterisk can be loaded in a variety of ways. The easiest way is to start Asterisk by running the binary file directly from the Linux command-line interface. If we are running on a system that uses the *init.d* scripts, you can easily start and restart Asterisk that way as well. However, the preferred way of starting Asterisk is via the *safe_asterisk* script.

The Asterisk binary is, by default, located at /usr/sbin/asterisk. If we run /usr/sbin/asterisk, it will be loaded as a daemon. There are also a few switches we should be aware of that allow us to (re)connect to the Asterisk CLI, set the verbosity of CLI output, and allow core dumps if Asterisk crashes (for debugging with gdb). Asterisk needs to be executed with root privileges so it can performs all the required operations to run. To explore the full range of options, run Asterisk with the `-h` switch (12):

```
# /usr/sbin/asterisk -h      (12)
```

Next is a list of the most commonly used options:

- `-c`: Console. This will start Asterisk as a user process (not as a server), and will connect you to the Asterisk CLI. This option is good when we are debugging your startup parameters, but should not be used for a normal system (if Asterisk is already running, this option will not work and will issue a complaint).
- `-v`: Verbosity. This is used to set the amount of output for CLI debugging. The more *"v"*s, the more verbose.
- `-g`: Core dump. If Asterisk were to crash unexpectedly, this would cause a core file to be created for later tracing with gdb. We generally do not use this in production, unless we are writing code for Asterisk and want to debug any resulting crashes.
- -r: Remote. This is used to reconnect remotely to an already running Asterisk process. (The process is remote from the standpoint of the console connecting to it but is actually a local process on the machine. This has nothing to do with connecting to a remote process over a network using a protocol such as IP, as this is not supported). This is the most common option and it is what we would use to connect to Asterisk on a system where it is running as a daemon/service that was started by init at boot time.
- `-x "<CLI command>"`: Execute. Using this command in combination with `-r` allows us to execute a CLI command without having to connect to the CLI and type it manually. An example would be to send a restart, which we would do by typing `asterisk -rx "reload"` from the command line.

3.3.1. Dial Plan

The dial plan is truly the heart of any Asterisk system, as it defines how Asterisk handles inbound and outbound calls. In a brief, it consists of a list of instructions or steps that Asterisk will follow. Unlike traditional phone systems, Asterisk's dial plan is fully customizable. To successfully set up our own Asterisk system, we will need to understand the dial plan.

The Asterisk dial plan is specified in the configuration file named extensions.conf. The dial plan is made up of four main concepts: contexts, extensions, priorities and applications. If we installed the sample configuration files when we installed Asterisk, with command (10), we will most likely have an existing extensions.conf file.

3.3.1.1. Contexts

Dial plans are broken into sections called contexts. Contexts are named groups of extensions, which serve several purposes. Contexts keep different parts of the dial plan from interacting with one another. An extension that is defined in one context is completely isolated from extensions in any other context, unless interaction is specifically allowed.

Contexts are denoted by placing the name of the context inside square brackets ([ ]). The name can be made up of the letters A through Z (upper- and lowercase), the numbers 0 through 9, and the hyphen and underscore. For example, a context for incoming calls looks like this:

```
[incoming]
```

All of the instructions placed after a context definition are part of that context, until the next context is defined. At the beginning of the dial plan, there are two special contexts named [general] and [globals]. The [general] section contains a list of general dial plan settings (which we'll probably never have to worry about), and the [globals] context contains the definition of global variables that could be used on all the dial plans defined. So we must avoid the use of context names [general] and [globals].

When we define a channel (which is how we connect things to the system), one of the parameters that is defined in the channel definition is the context. In other words, the context is the point in the dial plan where connections from that channel will begin.

Another important use of contexts (perhaps the most important) is to provide security. By using contexts correctly, we can give certain callers access to features (such as long distance calling) that aren't made available to others. If we don't design our dial plan carefully, we may inadvertently allow others to fraudulently use our system.

3.3.1.2. Extensions

In the world of telecommunications, the word extension usually refers to a numeric identifier given to a line that rings a particular phone. In Asterisk, however, an extension is far more powerful, as it defines a unique series of steps (each step containing an application) that Asterisk will take that call through. Within each context, we can define as many extensions as required. When a particular extension is triggered (by an incoming call or by digits being dialed on a channel), Asterisk will follow the steps defined for that extension. It is the extension, therefore, that specify what happens to calls as they make their way through the dial plan. Although extensions can certainly be used to specify phone extensions in the traditional sense, in an Asterisk dial plan they can be used for much more.

The syntax for an extension is the word exten , followed by an arrow formed by the equals sign and the greater-than sign, like this:

```
exten =>
```

This is followed by the name (or number) of the extension. When dealing with traditional telephone systems, we tend to think of extensions as the numbers we would dial to make another phone ring. In Asterisk, we get a whole lot more.

A complete Asterisk's extension is composed of three components:

- The name (or number) of the extension.
- The priority. Each extension can include multiple steps. The step number is called the priority.
- The application (or command) that performs some action on the call.

These three components are separated by commas, like this:

```
exten => name,priority,application()
```

3.3.1.3. Priorities

Each extension can have multiple steps, called priorities. Each priority is numbered sequentially, starting with 1, and executes one specific application. As an example, the following extension would answer the phone (in priority number 1), and then hang it up (in priority number 2):

```
exten => 123,1,Answer()

exten => 123,2,Hangup()
```

In older releases of Asterisk, the numbering of priorities caused a lot of problems. Asterisk does not handle missing steps or misnumbered priorities, and debugging this is difficult. Beginning with version 1.2, Asterisk addressed this problem. It introduced the use of the n priority, which stands for “next”.

Each time Asterisk encounters a priority named n, it takes the number of the previous priority and adds 1. This makes it easier to make changes on the dial plan, as we don’t have to keep renumbering all our steps.

Also starting with Asterisk version 1.2 and higher, common practice is to assign text labels to priorities. This is to ensure that we can refer to a priority by something other than its number, which probably isn’t known, given that dial plans now generally use unnumbered priorities. To assign a text label to a priority, simply add the label inside parentheses after the priority, like this:

```
exten => 123,n(label),application()
```

3.3.1.4. Applications

Applications are the workhorses of the dial plan. Each application performs a specific action on the current channel, such as playing a sound, accepting touch-tone input, dialing a channel, hanging up the call, and so forth. In a previous example, we were introduced to two simple applications: Answer() and Hangup() .

Some applications, such as Answer() and Hangup(), need no other instructions to do their jobs. Other applications require additional information. These pieces of information, called arguments, can be passed on to the applications to affect how they perform their actions. To pass arguments to an application, we place them between the parentheses that follow the application name, separated by commas. Occasionally, we may also see the pipe character (|) being used as a separator between arguments, instead of a comma. But we can use whichever we prefer.

Finally we will show some of the available dial plan applications to have an idea of what kind of actions can be performed on a Asterisk's dial plan:

- **Answer([delay])**: Causes Asterisk to answer the channel if it's currently ringing. If the current channel is not ringing this application does nothing. If a delay is specified, Asterisk will answer the call and then wait delay milliseconds before going on to the next priority in the dial plan.

- **HangUp(cause-code)**: Unconditionally hangs up the current channel. If supported on the channel, cause-code will be specified to the remote end as the reason for ending the call. Acceptable values for cause-code are 16(Normal call clearing), 17 (Busy), 19 (No answer), 21 (Rejected) or 34(Congestion).

- **[E]AGI(program[,arguments])**: Executes an Asterisk Gateway Interface (AGI, see section 3.3.4) compliant program on the current channel. The program must be set with execution permissions in the underlying file system. The program path is relative to the Asterisk AGI directory (by default /var/lib/asterisk/agi-bin/). If we want to access to the inbound audio stream from within our AGI program, we should use EAGI.

- **SendDTMF(digits[,timeout_ms])**: Sends the specified DTMF digits on a channel. Valid DTMF digits include 0-9, *, # and A-D. We may also use the letter w as a digit, which indicates a 500 ms wait. The timeout_ms argument is the amount of time, in milliseconds, between digits. If not specified defaults to 250 ms.

- **Set(n=value,[n2=value2..[,options]])**: Sets the variable n to the specified value. Also sets the variable n2 to the value of value2. If the variable name is prefixed with _, single inheritance is assumed. If the variable name is prefixed with __, infinite inheritance is assumed. Inheritance is used when we want channels created from the current channel to inherit the variable from the current channel. If the options parameter is set to g, the variables will be set as global variables instead of channel variables.

- **System(command)**: Executes a command in the underlying operating system. This application sets a channel variable named SYSTEMSTATUS to either FAILURE or SUCCES, depending on whether or not Asterisk was successfully able to run the command.

- **Transfer([Technology/]destination[,options])**: Requests that the remote caller be transferred to the given optional Technology and destination. If the Technology is set to IAX2, SIP, Zap, etc.., then transfer will happen only if the incoming call is of the same channel type. Upon completion, this application sets a channel variable named TRANSFERSTATUS to one of the values: SUCCESS (the transfer was successful), FAILURE (the transfer was not successful) or UNSUPPORTED (the transfer was not supported by the underlying channel driver). If the options parameter is set to the letter j and the transfer is not supported or successful, and there exists a priority n+101 (where n is the current priority), that priority will be taken next.

- **Dial(tech/username:password@hostname/extension[&tech2/peer2...][,ring-timeout[,flags[,URL]]])**: Allows to connect together all of the various channel types. This is probably the most important application in Asterisk. Any valid channel type (such as SIP, IAX2, H.323, MGCP, Local or Zap) is accepted but the parameters that need to be passed to each channel will depend on the information the channel type needs. When a channel type that is network-based is specified, we can pass the destination host, username, password and remote extension as part of the options, or we can refer to the name of a channel entry in the appropriate .conf file. All the required information will then need to be obtained from that file.

3.3.2. VoIP Protocols

Asterisk, as an IP based PBX, makes use of VoIP protocols through their channels. VoIP channels in Asterisk represent connections to the protocols they support. Each protocol we wish to use requires a configuration file, containing general parameters defining how the system handles the protocol as well as specific parameters for each channel or device to be referenced in the dial plan.

VoIP systems work by digitalizing the voice in data packets, sending them through a data network and then reconstruct the voice on the destination side. This process starts with the analog signal from a telephone which is digitalized in PCM (Pulse Code Modulation) signals through a codec. The PCM samples are compressed to data packets that can be transmitted, for instance, through a WAN private network. In the other side the inverse process is performed.

In the case we want to transmit the VoIP through the public Internet, we need of an interface which allows the interaction between the phone network and the IP network, which is called gateway and its main responsibility is the conversion between analog voice to compressed IP packets on transmitter side and the conversion from compressed IP packets to analog voice in the receiver side.

All the network require of some way of addressing, routing and signaling to work properly. Addressing is required to identify the origin and destination of any call. With routing, the network found the best path to send a packet between two points of the network. And signaling alerts the destination extensions and the network elements of their state and their role when establishing a new connection.

As we have mentioned, IP telephony requires some protocols for establishing and controlling the connection, which is signaling the connection. Among them, the most popular protocols are H.323 and SIP. The H.323 recommendations contains collection of protocols, such as H.245, H.225.0, H.332, H.450.1, H.450.2, H.450.3, H.235 and H.246. H.323 is based in the Q.931 signaling protocol of ISDN. On the other hand, SIP makes uses of some mechanisms that are used on HTTP protocol. Both technologies use RTP protocol to the multimedia data transport.

3.3.2.1. RTP protocol

The RTP protocol is a real time protocol which is oriented to the transmission of information on real time such as voice or video. This protocol is a user session protocol which relies on UDP (User Datagram Protocol), making use of the checksum and multiplexing services to allow programs which make transmission of this kind of data handling the real time unicast or multicast transmissions. Although RTP itself isn't trustful, it provides relations with other

protocols and applications from lower layers and with some resources provided by switches and routers to warrant trustful.

The tool that RTP uses to achieve real time transmissions is the RTCP (Real-Time Control Protocol), which provides a feedback about some control information. With this, we can monitor the quality of the transmission and we can diagnose possible network problems. In addition to this, RTCP synchronizes the audio and video transmission, knows the number of users present on a conference call and with that, calculates the rate at which the packets should be sent.

To the data compression, RTP uses an application called *vocoder,* which allows to reduce the original 64 kbps of an standard digitalized telephone audio signal, up to 8 kbps with a loss of quality which isn't so much perceptible.

In Asterisk, RTP configuration is done through the *rtp.conf* file. Basically, in this file we define the UDP range of ports that will be used for RTP connections. By default this is defined between 10,000 (*rtpstart* parameter) and 20,000 (*rtpend* parameter). For every bidirectional SIP call between two endpoints, five ports are generally used: port 5060 for SIP signaling, one port for the data stream and one port for the RTCP in one direction, and an additional two ports for the data stream and RTCP in the opposite direction.

UDP datagrams contain a 16-bit field for a CRC (Cyclic Redundancy Check), which is used to verify the integrity of the datagram header and its data. It uses polynomial division to create the 16-bit checksum from the 64-bit header. This value is then placed into the 16-bit CRC field of the datagram header, which the remote end can use to verify the integrity of the received datagram. This can be controlled through *rtcpchecksum* parameter.

3.3.2.2. H.323 protocol

The H.323 standard allows the interchange of multimedia services such as video, audio or real time data) over packet networks and is a part of the ITU-T H.32X recommendations which provide multimedia communications over all network types.

H.323 specifies the following network elements that play a role in every multimedia communications:

- Terminals: Are used for real time bidirectional communications. It's used for audio communications and optionally can be used for video and audio communications. Can be a computer or a device running a multimedia application which uses H.323.

- Multi point Control Units (MCU): It is responsible for managing multi point conferences and is composed of two logical entities referred to as the Multipoint Controller (MC) and the Multipoint Processor(MP). In more practical terms, an MCU is a conference bridge not unlike the conference bridges used in the PSTN today. The most significant difference, however,

is that H.323 MCUs might be capable of mixing or switching video, in addition to the normal audio mixing done by a traditional conference bridge. Some MCUs also provide multi point data collaboration capabilities. What this means to the end user is that, by placing a video call into an H.323 MCU, the user might be able to see all of the other participants in the conference, not only hear their voices.

- Gateways: Are used for interconnecting different networks. It connects H.323 networks with networks that aren't H.323, such as ISDN or PSTN. To achieve the interconnection, protocols from one network are traduced to protocols of the other network. Inside a H.323 network, a gateway is not required.

- Gatekeepers: A Gatekeeper is an optional component in the H.323 network that provides a number of services to terminals, gateways, and MCU devices. Those services include endpoint registration, address resolution, admission control, user authentication, and so forth. Of the various functions performed by the gatekeeper, address resolution is the most important as it enables two endpoints to contact each other without either endpoint having to know the IP address of the other endpoint.

  Gatekeepers may be designed to operate in one of two signaling modes, namely "direct routed" and "gatekeeper routed" mode. Direct routed mode is the most efficient and most widely deployed mode. In this mode, endpoints utilize the RAS protocol in order to learn the IP address of the remote endpoint and a call is established directly with the remote device. In the gatekeeper routed mode, call signaling always passes through the gatekeeper. While the latter requires the gatekeeper to have more processing power, it also gives the gatekeeper complete control over the call and the ability to provide supplementary services on behalf of the endpoints.

- H.323 zones: An H323 zone is a collection of gateways, MCUs and terminals that are managed by a gatekeeper. This must include, at least, one terminal and optionally gateways and MCUs, but only one gatekeeper.

The H.323 standard defines some protocols which are used by the standard. Those protocols are (see fig. 3):

- Audio codecs: Allow the codification of audio signals that are transmitted by the terminal and the decoding of received data onto an audio signal. Every terminal must support at least the ITU-T G.711 codec (raw 64 kbps codec). Additionally it can also work with G.722, G.723, G.728 and G.729.

- Video codecs: Allow the codification of the video signals that are transmitted by the terminal and the decoding of received data onto a video signal. The support to those codecs, which are H.261 and H.263, is optional.

- Registration, Admission and Status (RAS) H.225: It's a protocol that is used between the terminals and gateways, and the gatekeeper. RAS is used by the registering functions, the admission control and the bandwidth management among other functions. For sending RAS messages, RAS channels are used, which is a signaling channel that is open between a terminal and a gatekeeper before the establishment of any other type of channel.

- Call signaling H.225: This signaling is used for the establishment of connections between two H.323 terminal points, through the interchange of messages through the signaling channel. This signaling channel exists between two H.323 terminal points or between a terminal point and a gatekeeper.

- Control signaling H.245: This signaling is used for managing the operation of H.323 endpoints interchanging control messages that send the following information: capacity interchange, the opening and closing of logical channels used for transport multimedia, flow control and commands and general indications.

- Real Time Protocol (RTP): Performs the sent and reception of multimedia data (see section 3.3.2.1).

- Real Time Control Protocol (RTCP): Provides RTP control.

- H.323 is independent of the packet network and the transport protocols over it works, so it doesn't establish specifications for them.

The H.323 standard is used by Asterisk by creating H.323 channels. The H.323 channels are managed by the chan_h323 module and is configured by the *h323.conf* file. In this file some parameters related to the standard are configured in its general section, such as configuring the way the gatekeeper is discovered with the *gatekeeper* parameter. In addition, there also are defined all the H.323 extensions (which actually are H.323 terminals) by adding a new section with the extension name as section name for each new user.

3.3.2.3. SIP protocol

The SIP (Session Invitation Protocol) protocol was defined by the IETF trying to define a mechanism to invite users to sessions point to point and multicast sessions. It's based on HTTP and SMTP and thus uses TCP as transport protocol and it's based on text. SIP is in continuous evolution and improvement. Its main advantages are its simplicity, scalability, distributed functionality and versatility with Internet protocols.

SIP can't send a session description to a user until it hasn't been localized. Frequently the user can be found on different locations. To allow that, SIP

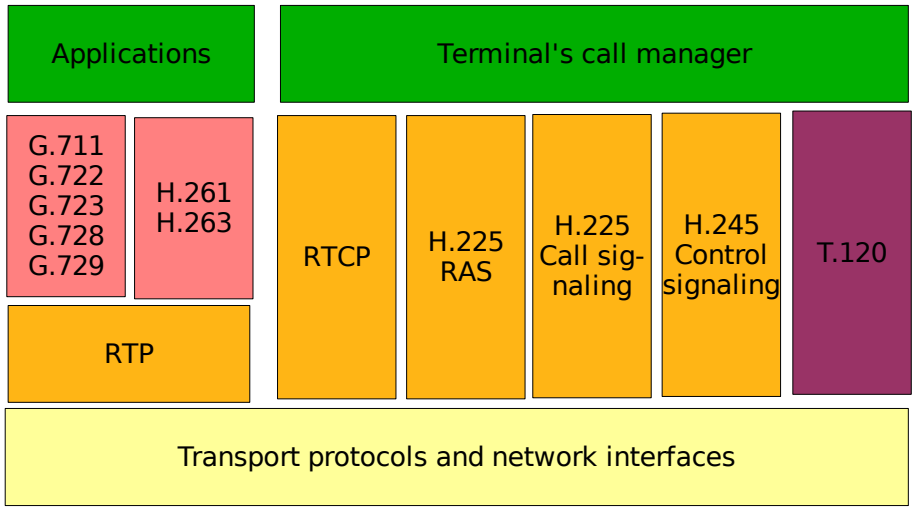

Fig. 3: H.323 protocols

supports transparently the name mapping and the readdressing of services, allowing the implementation of intelligent network services on telephony.

The SIP's URLs, which follow the RFC 2396, are used on the message to indicate the caller (From), the destination (Request-URI) and who is being called (To) in a SIP request, and to specify the redirection address (Contact).

The SIP protocol defines a collection of entities that take part on a SIP communication, which are:

- User Agent Client (UAC): It's who call. A client at application level which starts a SIP request.
- User Agent Server (UAS): It's a server at application level which contacts the user when a SIP request is received and responses on the user's name. The response to the request is accepted, rejected or redirected.
- User Agent (UA): It's an application which contains both the UAC and UAS. When an user want to talk with other, executes a program that contains an UA. The user interacts through the interface. When the first user click on the call to the second user button, the UA sends an appropriate SIP message to establish the call. All the interactions between users and the SIP protocol are done through UA.
- Redirect server: They help localizing UA by providing alternative locations where the user can be found. Generally the redirect server doesn't returns directly the user address but it return the address of another server that has more knowledge on the specified user.
- Proxy server: The proxy servers accept sessions requests generated by UA and request the address information about the destination user to the

registrar server. Then, it redirects the invitation directly to the destination user if it's located on the same domain, or redirects it to another proxy of the corresponding domain.

- Forking proxy server: When a proxy server sends an invite of the same session to different points, it's forking the invitation. So the forking proxy servers can execute parallel or sequential searches depending on its configuration.

- Address group: The proxy servers can create address groups and try to contact with all the users of the same group until it finds the user who is available.

- Registrar server: The registrar servers are SIP servers which accept invitations. A registrar server is commonly located next to a redirect server or a proxy server.

- Localization server: The localiztion servers aren't SIP entities by itself but they take an important role in any architecture which uses SIP. It stores and return possible localizations of the users.

- Client/Server transactions: SIP is based in the HTTP protocol which is a request/response protocol. A client is a SIP entity which generates requests and a server is a SIP entity which receives those requests and returns responses. SIP uses the same procedures as HTTP to manage sessions.

On the Asterisk side the sip protocol is configured through the sip.conf configuration file. On the general section of this file some protocol parameters are configured such as. Furthermore, there also are defined all the SIP extensions (which actually are SIP UA) by adding a new section with the extension name as section name for each new user.

3.3.2.4. IAX2 protocol

The IAX2 protocol provides control and transmission of multimedia date through IP networks. IAX can be used with any type of multimedia data although it has been designed focusing on the control of VoIP calls. Their main design objectives were:

1. Minimize the use of bandwith for control and multimedia data, with special focus on individual voice calls.

2. Provide a transparent support of NAT

The signaling component of the IAX is a master-slave control protocol more similar to the SIP protocol than to the MGCP (Media Gateway Control Protocol). Regarding the multimedia data, the sequence information and the timing is included on the IAX frames. The multimedia transport doesn't uses RTP.

The basic design of the protocol was done with a multiplexed signaling and multiple multimedia data stream over a single UDP (by default the 4569 UDP port) association between to Internet hosts. As the signaling and multimedia data share the same UDP port, IAX avoid any problem when working over a NAT-enabled networks.

The IAX messages are called frames. There are sever basic types of frames. A bit called F on the frame indicates if it's a *Full Frame* or if it's of other type. The *Call Number* is a integer number of 15 bits which is used to identify the data stream among all the multiplexed streams in a host. A phone call has to call numbers associated to it, one for every communication direction. There also is a *Timestamp* that can be of 16 or 32 bits.

A Full frame is user to send signaling, audio or video in a trustful way. The full frames are the only type of message that is sent in a trustful way. This implies that the destination host must return some type of response message to confirm the reception of the frame. There are two type of control information that is passed using Full frames, which are control frames and IAX control frames. The control frames provide specific administration of the devices connected on the destination point. The IAX control frames provide specific administration of the destination such as handling interactions of the IAX protocol.

The Mini frames are used to send multimedia data with a minimum overhed. With that we optimize the bandwidth.

Regarding Asterisk, the IAX configuration is done through the iax.conf configuration file. This file contains a general section with some protocol specific parameters, such as the *bindport* parameter which defines the UDP port used for IAX communications, or *register* directives to register our Asterisk PBX with a remote Asterisk PBX system. There also are defined all the IAX extensions by adding a new section with the extension name as section name for each new user.

Finally we will review some differences between SIP and IAX [11], which are the two VoIP protocols most widely used with Asterisk:

- IAX is more efficient than RTP for any number of calls and for any codec. The benefit is anywhere from 2.4 kbps for a single call to approximately tripling the number of calls per megabit for G.729 when running in trunk mode.

- IAX is information-element encoded rather than ASCII encoded. This makes implementations substantially simpler and more robust to buffer overrun attacks since absolutely no text parsing or interpretation is required. The size of IAX signaling packets is smaller than those of SIP, but that is generally not a concern except with large numbers of clients frequently registering. Generally speaking, IAX2 is more efficient in its encoding, decoding and verifying information, and it would be extremely

difficult for an author of an IAX implementation to somehow be incompatible with another implementation since so little is left to interpretation.

- IAX has a very clear link and network layer separation, meaning that both signaling and audio have defined states, are robustly transmitted consistently, and that when one end of the call abruptly disappears, the call will terminate, even if no more signaling and/or audio is received. SIP does not have such a mechanism, and its reliability from a signaling perspective is obviously very poor and clumsy requiring additional standards beyond the core RFC-3261.

- IAX's unified signaling and audio paths permit it to transparently navigate through NAT and provide a firewall administrator only a single port to have to open to permit its use. It requires an IAX client to know absolutely nothing about the network that it is on to operate.

- IAX's authenticated transfer system allows the transfer of audio and call control off a server-in-the-middle in a robust way such that if the two endpoints cannot see one another for any reason, the call continues through the central server.

- IAX clearly separates Caller-ID from the authentication mechanism of the user. SIP does not have a clear method to do this.

- SIP is an IETF standard. IAX is not an official published standard at this time, but there has been published an IETF informational Draft [22] looking for the standardization of IAX2.

- IAX allows an endpoint to check the validity of a phone number to know whether the number is complete, may be complete, or is complete but could be longer. There is no way to completely support this in SIP.

- IAX always sending DTMF out of band so there is never any confusion about what method is used.

- IAX support transmission of language and context, which are useful in an Asterisk environment. That's pretty much all that comes to mind at the moment.

3.3.3. Command Line Interface (CLI)

The command line interface of Asterisk is the interface from where the whole Asterisk administration is done. To connect to to the Asterisk's CLI we only need to execute the asterisk command with the -c parameter or with the -r if we have previously started asterisk in server mode.

The CLI allow the interaction with a running Asterisk server and it will be very useful for troubleshooting and monitoring purposes. Since the CLI employs tabbed name completion, we can press the Tab key to see a list of possible commands. This makes the CLI very easy to use. Next we will review some CLI commands:

- **!*command***: Executes a given shell command. If followed immediately by a carriage return, Asterisk starts an interactive shell.
- **Dial [extension[@context]]**: Dials a given extension (optionally, in the context specified) through the CONSOLE channel. This command is available only if either chan_oss.so or chan_alsa.so modules are loaded.
- **Set verbose *level***: Sets the verbosity level of the console. A setting of 0 means no information on calling activity will be displayed. This command has exactly the same effect as the -v flags we provide on the call of the asterisk command.
- **Show channels [concise]**: Lists the currently defined channels and some information about them. If concise is specified, the format is abridged and presented in a more easily machine-parsable format.
- **show dialplan [context]**: Shows the current state of the dialplan as loaded into memory. If a context name is appended to the end of the command, only that context will be shown. This command is useful for verigying the order of pattern matching as well.
- **stop now**: Stops immediately, terminating all active calls.
- **sip show users**: Displays a listing of all users configured in sip.conf.
- **reload [module ...]**: Reloads configuration files for all listed modules that support reloading (or for all supported modules, if none specified).

3.3.4. Asterisk Gateway Interface (AGI)

The Asterisk Gateway Interface, or AGI, provides a standard interface by which external programs may control the Asterisk dial plan. Usually, AGI scripts are used to do advanced logic, communicate with relational databases (such as PostgreSQL or MySQL), and access other external resources. Turning over control of the dial plan to an external AGI script enables Asterisk to easily perform tasks that would otherwise be difficult or impossible.

Instead of releasing an API for programming, AGI scripts communicate with Asterisk over communications channels (file handles, in programming parlance) known as STDIN, STDOUT, and STDERR. Those are channels by which programs

in Unix-like environments receive information from and send information to external programs. STDIN, or standard input, is the information that is sent to the program, either from the keyboard or from another program. In this case, information coming from Asterisk itself comes in on the program's STDIN file handle. STDOUT, or standard output, is the file handle that the AGI script uses to pass information back to Asterisk. Finally, the AGI script can use the STDERR (standard error) file handle to write error messages to the Asterisk console.

The communication between Asterisk and an AGI script follows a predefined pattern. Next, we are going to enumerate the steps. When an AGI script starts, Asterisk sends a list of variables and their values to the AGI script. The variables might look something like this:

```
agi_request: test.php

agi_channel: Dahdi/1-1

agi_language: en

agi_callerid:

agi_context: default

agi_extension: 123

agi_priority: 2
```

After sending these variables, Asterisk sends a blank line. This is the signal that Asterisk is done sending the variables, and it is time for the AGI script to control the dial plan. At this point, the AGI script sends commands to Asterisk by writing to STDOUT. After the script sends each command, Asterisk sends a response that the AGI script should read. These actions (sending commands to Asterisk and reading the responses) can continue for the duration of the AGI script. In order to work properly, the AGI script must be executable. To use an AGI script inside our dial plan, we simply call the AGI() application, with the name of the AGI script as the argument, like this:

```
exten => 123,1,Answer()

exten => 123,2,AGI(agi-test.agi)
```

AGI scripts often reside in the AGI directory (usually located in /var/lib/asterisk/agi-bin), but we can specify the complete path to the AGI script. In addition to the AGI() application, there are several other AGI applications suited to different circumstances. The EAGI() (enhanced AGI) application acts just like AGI() but allows our AGI script to read the inbound audio stream on file descriptor number three. The DeadAGI() application is also just like AGI(), but it works correctly on a channel that is dead (i.e., a channel that has been hung up). As this implies, the regular AGI() application doesn't work on dead channels. The FastAGI() application allows the AGI script to be called across the network, so that multiple Asterisk servers can call AGI scripts from a central

location.

Finally we will mention some of the most interesting AGI commands that can be called on AGI scripts in order to show which operations can be done from an AGI script:

- ANSWER: Answers the channel if it's not already in an answered state.
- CHANNEL STATUS *[channelname]*: Queries the status of the channel indicated by channelname or, if no channel is specified, the current channel.
- EXEC *application options*: Executes the dial plan application *application* passing *options* as parameters to it.
- GET DATA *filename [timeout] [max_digits]*: Plays the audio file specified by filename and accepts DTMF digits, up to the limit set by max_digits. Similar to the Background() dialplan application.
- GET FULL VARIABLE *variablename [channelname]*: If the variable indicated by variablename is set, returns its value in parentheses. This command understands complex variable names and built-in variable names, unlike GET VARIABLE.
- HANGUP *[channelname]*: Hang ups the specified channel or, if no channel is specified, the current channel.
- RECORD FILE *filename format escape_digits timeout [offset_samples] [BEEP] [s=silence]*: Records the channel audio to the specified file until the reception of a defined escape (DTMF) digit. The format argument defines the type of file to be recorded (wav, gsm, etc.). The timeout argument is the maximum number of milliseconds the recording can last, and can be set to -1 for no timeout. The offset_samples argument is optional; if provided, it will seek to the offset without exceeding the end of the file. The BEEP argument will play a beep to the user to signify the start of the record operation. The silence argument is the number of seconds of silence allowed before the function returns despite the lack of DTMF digits or reaching the timeout. The silence value must be preceded by s= and is also optional.
- SAY ALPHA *number escape_digits*: Says a given character string, returning early if any of the given DTMF digits are received on the channel.
- SET VARIABLE *variablename value*: Sets or updates the value for the variable name specified by variablename . If the variable does not exist, it is created.

3.3.5. Asterisk Manager Interface (AMI)

The Asterisk Manager Interface (AMI) is a powerful programmatic interface. It allows external programs to both control and monitor an Asterisk system. This interface is often used to integrate Asterisk with existing business processes and systems, CRM (Customer Relationship Management) software. It can also be used for a wide variety of applications, such as automated dialers and click-to-call systems. The Asterisk Manager Interface listens for connections on a network port. A client program can then connect to the Asterisk Manager Interface on that port, authenticate itself, and send commands to Asterisk. Asterisk will then respond to the request, as well as update the client program with the status of the system. To use the Manager, we must define an account in the file *manager.conf*.

It is important to keep in mind that the Manager interface is designed to be used by programs, not fingers. That's not to say that we can't issue commands to it directly, so don't expect a typical console interface, because that's not what Manager is for. Commands to Manager are delivered in packages with the following syntax (lines are terminated with Carriage Return followed by Line Feed or CR+LF):

```
Action: <action type>

Key 1: Value 1

Key 2: Value 2

etc ...

Variable: Value

Variable: Value

etc...
```

For example, to authenticate with Manager (which is required if we expect to have any interaction whatsoever), we would send the following:

```
Action: login

Username: admin

Secret: password

<CR+LF>
```

An extra CR+LF on a blank line will submit the entire package to Manager. Once authenticated, we will be able to initiate actions, as well as see events generated by Asterisk. On a busy system, this can get quite complicated and

become totally impossible to keep track of with the unaided eye. To turn keep Asterisk from sending events, we can add the Events parameter to our login command, like this:

```
Action: login

Username: admin

Secret: password

Events: off

<CR+LF>
```

If we're worried about sending our secret across the wire in plain text (which we should be), we can also authenticate using an MD5 challenge-response system, which works very similar to HTTP digest authentication. To do this, we first call the Challenge action, which will present we with a challenge token:

```
Action: Challenge

AuthType: MD5

Response: Success

Challenge: 840415273
```

We can then take that challenge token, concatenate the plain text secret onto the end of it, and calculate the MD5 checksum of the resulting string. The result can then be used to login without passing our secret in plain text.

```
Action: Login

AuthType: MD5

Username: admin

Key: e7a056e1488882c6c509bbe71a049978

Response: Success

Message: Authentication accepted
```

Once we've successfully logged into the AMI system, we can send commands to Asterisk by using the other actions. Finally we will show some of the available AMI actions in order to know the operations that can be performed through AMI:

- AbsoluteTimeout: Hangs up a channel after a certain time. Asterisk will acknowledge the timeout setting with a Timeout Set message. Parameters:

  - Channel: [required] The name of the channel on which to set the absolute timeout.
  - Timeout: [required] The maximum duration of the call, in seconds.
  - ActionID: [optional] An identifier that can be used to identify the response to this action.
- AgentLogoff: Logs off the specified agent for the queue system. Parameters:
  - Agent: [required] Agent ID of the agent to log off.
  - Soft: [optional] Set to true to not hangup existing calls.
  - ActionID: [optional] An identifier which can be used to identify the response to this action.
- Agents: This action lists information about all configured agents.
- ChangeMonitor: The ChangeMonitor action may be used to change the file started by a previous Monitor action. The following parameters may be used to control this. Parameters:
  - Channel: [required] Used to specify the channel to record.
  - File: [required] The new filename in which the monitored channel will be recorded.
  - ActionID: [optional] An identifier that can be used to identify the response to this action.
- Command: Runs an Asterisk CLI command as if it had been run from the CLI. Parameters:
  - Command: [required] Asterisk CLI command to run.
  - ActionID: [optional] An action identifier that can be used to identify the response from Asterisk.
- Events: Enables or disables sending of events to this manager connection. Parameters:
  - EventMask: [required] Set to on if all events should be sent, off if events should not be sent, or system,call,log to select which type of events should be sent to this manager connection.
  - ActionID: [optional] An identifier which can be used to identify the response to this action.
- ExtensionState: This command reports the extension state for the given

extension. If the extension has a hint, this will report the status of the device connected to the extension. Parameters:

- Exten: [required] The name of the extension to check.
- Context: [required] The name of the context that contains the extension.
- ActionId: [optional] An action identifier that can be used to identify this manager transaction.

- GetConfig: Retrieves the data from an Asterisk configuration file. Parameters:
  - Filename: [required]Name of the configuration file to retrieve.
  - ActionID: [optional] An identifier that can be used to identify the response to this action.
- GetVar: Gets the value of a local channel variable or global variable. Parameters:
  - Channel: [optional] The name of the channel from which to retrieve the variable value.
  - Variable: [required] Variable name.
  - ActionID: [optional] An identifier that can be used to identify the response to this action.
- ListCommands: Lists the action name and synopsis for every Asterisk Manager Interface action.
- Monitor: Records the audio on a channel to the specified file. Parameters:
  - Channel: [required] Specifes the channel to be recorded.
  - File: [optional] The name of the file in which to record the channel. The path defaults to the Asterisk monitor spool directory, which is usually /var/spool/asterisk/monitor. If no filename is specified, the filename will be the name of the channel, with slashes replaced with dashes.
  - Format: [optional] The audio format in which to record the channel. Defaults to wav.
  - Mix: [optional] A Boolean flag specifying whether or not Asterisk should mix the inbound and outbound audio from the channel in to a single file.
  - ActionID: [optional] An identifier that can be used to identify the

response to this action.

- Originate: Generates an outbound call from Asterisk, and connect the channel to a context/extension/priority combination or dialplan application. Parameters:
  - Channel: [required] Channel name to call. Once the called channel has answered, the control of the call will be passed to the specified Exten/ Context/Priority or Application.
  - Exten: [optional] Extension to use (requires Context and Priority).
  - Context: [optional] Context to use (requires Exten and Priority).
  - Priority: [optional] Priority to use (requires Exten and Context).
  - Application: [optional] Application to use.
  - Data: [optional] Data to pass as parameters to the application (requires Application).
  - Timeout: [optional] How long to wait for call to be answered (in ms).
  - CallerID: [optional] Caller ID to be set on the outgoing channel.
  - Variable: [optional] Channel variable to set. Multiple variable headers are allowed.
  - Account: [optional] Account code.
  - Async: [optional] Set to true for asynchronous origination. Asynchronous origination allows we to originate one or more calls without waiting for an immediate response.
  - ActionID: [optional] An identifier that can be used to identify the response to this action.
- QueueAdd: Adds a queue member to a call queue. Parameters:
  - Queue: [required] The name of the queue.
  - Interface: [required] The name of the member to add to the queue. This will be a technology and resource, such as SIP/Jane or Local/203@lab/n. Agents (as defined in agents.conf) can also be added by using the Agent/1234 syntax.
  - MemberName: [optional] This is a human-readable alias for the interface, and will appear in the queue statistics and queue logs.
  - Penalty: [optional] A numerical penalty to apply to this queue member. Asterisk will distribute calls to members with higher penalties only

after attempting to distribute the call to all members with a lower penalty.

- Paused: [optional] Whether or not the member should be initially paused.
- ActionID: [optional] An action identifier that we can use to identify the response to this manager transaction.

- SetVar: Sets a global or channel variable. Parameters:
  - Channel: [optional] Channel on which to set the variable. If not set, the variable will be set as a global variable.
  - Variable: [required] Variable name.
  - Value: [required] Value.
  - ActionID: [optional] An identifier that can be used to identify the response to this action.

3.4. Asterisk Code Structure

The source code of the Asterisk project is divided in some subversion projects, despite the main of the project is that named Asterisk. The Asterisk main project code is divided in some directories that group the different source files with those of the same type. First of all we need to locate the asterisk source code directory. In our case we have done a Subversion's checkout of the code onto the /usr/src/asterisk directory (see section 3.3).

If we review the content of this directory we easily can describe the code structure of Asterisk. The most important parts that we will find inside the directory are:

- Makefile files: Those files define the rules that the gcc compiler will follow when compiling and installing the Asterisk source code. Inside every directory that contains source files there also exists a Makefile file which describes the rules for the files contained within the directory. Those Makefiles are called from the root Makefile. There also are three directories used by make tools: the **autoconf/**, **build_tools/** and **menuselect/** directories.

- **apps/**: Inside this directory there are included the source code of the dial plan applications such as the Dial application,which is in app_dial.c file, or the SendDTMF application, which is in app_senddtmf.c file.

- **bridges/**: This folder contains source files that implement the Asterisk's call bridging features.

- **channels/**: Inside this directory are defined the different types of channels that Asterisk can handle. Every source file contains a kind of driver for every supported channel. We can find chan_sip.c as the source file which controls the SIP channels or chan_h323.c as the source file which controls H.323 channels.

- **cdr/**: Inside this directory we find source files which control the back end connection with different databases for CDR (Call Detail Record) purposes. We can find the source file cdr_pgsql.c which acts as back end for PostgreSQL databses or the source file cdr_odbc.c which acts as back end with ODBC database connections.

- **codecs/**: Inside this directory there are the source files which control the codification and decodification onto the different supported formats. We can find the codec_gsm.c as the source file which controls the GSM codec or the codec_g711.c as the source file which controls the G.711 codec.

- **doc/**: This folder contains documentation in text files that is of special interest for developer. For instance we can find the code guidelines on the CODING_GUIDELINES file or information about the Asterisk's speech recognition API on the speechrec.txt file.

- **formats/**: Inside this directory we can find the source files which control the multimedia files handling. That is the play of sound files of several formats. For instance we can find the control of WAV files on the format_wav.c source file or the control of JPEG files on the format_jpeg.c source file.

- **funcs/**: This folder contains source files with dial plan function. Those functions can be used from the dial plan. For instance we can find the source file func_channel.c which contain channel related dial plan functions, or func_module.c which contain Asterisk's modules handling functions.

- **include/**: This directory contains all the header files for all the source files of the Asterisk project.

- **main/**: Inside this folder we can find the main Asterisk source code. Them asterisk main file is located inside this folder and named asterisk.c. All the PBX functionalities are implemented on source files that are locate inside this directory. For instance on the manager.c source file is implemented the Asterisk Manager Interface server (and the handling of basic AMI commands) or on the cli.c source file where the command line interface is managed (and the handling of basic CLI commands).

- **pbx/**: Inside this folder we can find the modules which manages the PBX

functioning. That is mainly the parsing of the dial plan from the configuration file and its real time handling while the Asterisk execution

- **res/**: Inside this directory we can find some resource modules, which are modules which provides Asterisk with additional resources. Those modules can include some Asterisk dial plan application or some Asterisk Manager Interface command which are useful for the module management. For instance, we can find the res_agi.c source file we contains the AGI commands handling for Asterisk, or the res_odbc.c source files which contains the ODBC database connections handling for Asterisk.

- Some test directories which contain some kind of samples useful for understanding Asterisk:

  - **agi/**: Inside directory there are some examples of AGI scripts in different languages.

  - **configs/**: This folder contain the configuration files samples which are copied to the /etc/asterisk/ directory if we execute the “make samples” command after installing Asterisk.

  - **contrib/**: Inside this file we can find some miscellaneous files which can be useful when developing for asterisk. For instance we can find the festival-1.95.diff which contains some modifications that need to be done to the Festival base configuration in order to work properly with the Festival Asterisk's dial plan application, or a init.d/ subdirectory which contains the start script for several GNU/Linux distributions.

  - **sounds/**: This folder contains the default Asterisk recorded sounds that can be used for testing purposes. They are available on all the supported file formats.

  - **utils/**: Inside this directory we can find useful programs that can be used from for testing purposes on the development process We can find frames.c which contains functions for processing sound files.

  - **test/**: This folder contains some Asterisk modules that are used for test some Asterisk aspects. We can find a test_schel.c source file which contains a module for testing the Asterisk thread scheduler.

We can note that the source code structure is closely related to the Asterisk PBX architecture. It has been organized the source code in that way so it would be easy to find a concrete source file inside the source code structure.

3.4.1. Asterisk's Satellite Projects

In addition to the Asterisk main project, in the project's Subversion repository there are defined other projects related with Asterisk. As we have mentioned during the installation process of Asterisk, some of them are recommended to be installed by default with Asterisk (which are libPRI, dahdi-linux and dahdi-tools), and the rest are tools that can improve Asterisk for a determined purpose. Among all the existing satellite projects, that are about twenty projects, we remark the following as the most interesting:

- libPRI: This project is developing a C implementation of the Primary Rate ISDN specification. It was based on the Bellcore specification SR-NWT-002343 for National ISDN. It has been tested to work with different vendors such as Nortel, Ericsson or Avaya.

- dahdi-linux: This satellite project is developing the kernel modules for DAHDI (Digium Asterisk Hardware Device Interface). It requires the dahdi-tools to work properly.

- dahdi-tools: This project is developing the user tools to configure the kernel modules included in dahdi-linux package.

- asterisk-addons: This project is developing additional modules for Asterisk which are, for one reason or another, not included in the normal base distribution. We must keep in mind when using them that many of these module are in an experimental start and are not recommended to be used on production environments.

- Asterisk-gui: This project is developing a framework for the creation of graphical interfaces for configuring and managing Asterisk PBX. It also includes some graphical interfaces for specific situations. We can find another project which is developing a GUI for configuring Asterisk and is probably more widely used than Asterisk-GUI based solution. This project is FreePBX [23] but its maintained outside the Asterisk's project repository.

- Astconfig: This project is developing a Gtk window based configuration tool for Asterisk.

3.4.2. Coding Guidelines

Finally, the last point that we will review prior to start our development contribution to the Asterisk project and finish this documentation contribution

to the project is reviewing the Asterisk's coding guidelines [24] (for up to date information read the subversion's file). When we develop for the Asterisk project we should keep in mind those guidelines in order to produce a standardized code for Asterisk and to have our code easily committed to the project.

This document gives some basic indication on how the asterisk code is structured. Its first part covers the structure and style of individual files and the second part (which is still uncompleted) covers the overall code structure and the build architecture.

3.4.2.1. Structure and style of individual files

First of all are defined some general rules which are the base of the Asterisk development rules:

- All code, filenames, function names and comments must be in English.
- Don't annotate our changes with comments like "/* JMG 4/20/04 */";
- Comments should explain what the code does, not when something was changed or who changed it. If we have done a larger contribution, make sure that we are added to the CREDITS file.
- Don't make unnecessary whitespace changes throughout the code. If we make changes, submit them to the tracker as separate patches that only include whitespace and formatting changes.
- Don't use C++ type (//) comments.
- Try to match the existing formatting of the file we are working on.
- Use spaces instead of tabs when aligning in-line comments or #defines (this makes our comments aligned even if the code is viewed with another tab size)

The next point that is reviewed on the coding guidelines is the source file structure and the header file structure. Next, some guidelines for declaring functions and variables are stated:

- Do not declare variables mid-block (e.g. like recent GNU compilers support) since it is harder to read and not portable to GCC 2.95 and others.
- Functions and variables that are not intended to be used outside the module must be declared static.
- When reading integer numeric input with scanf (or variants), do _NOT_ use '%i' unless we specifically want to allow non-base-10 input; '%d' is always a better choice, since it will not silently turn numbers with leading

zeros into base-8.

- Strings that are coming from input should not be used as a first argument to a formatted *printf function.

The next point to keep in mind is to use internal API functions where it is possible. For instance, if we are using a string handling function we should review if there exists a equivalent function on ast_str API and use it if exists.

Another point that is commented in this part of the guidelines is how the code should be formatted, that is how to indent it, where to put white spaces and where not, ... Regarding the function naming the only constraint is to name ast_*** all the public functions that are all those not marked as static.

The next aspect in which the guidelines document focuses is the handling of pointers and memory allocations:

- Dereference or localize pointers : Always dereference or localize pointers to things that are not yours like channel members in a channel that is not associated with the current thread and for which you do not have a lock.

- Use *const* on pointer arguments if possible : Use *const* on pointer arguments which your function will not be modifying, as this allows the compiler to make certain optimizations. In general, use *const* on any argument that you have no direct intention of modifying, as it can catch logic/typing errors in your code when you use the argument variable in a way that you did not intend.

- Do not create your own linked list code: As a common example of this point, make an effort to use the lockable linked-list macros found in include/asterisk/linkedlists.h. They are efficient, easy to use and provide every operation that should be necessary for managing a singly-linked list.

- Avoid needless allocations: Avoid needless malloc(), strdup() calls. If you only need the value in the scope of your function try ast_strdupa() or declare structs on the stack and pass a pointer to them. However, be careful to never call alloca(), ast_strdupa() or similar functions in the argument list of a function you are calling; this can cause very strange stack arrangements and produce unexpected behavior.

- Allocations for structures : When allocating/zeroing memory for a structure, avoid the combination of ast_malloc() and memset(). Instead, always use ast_calloc(). This will allocate and zero the memory in a single operation. In the case that uninitialized memory is acceptable, there should be a comment in the code that states why this is the case.

- Using sizeof(*tmp) instead of sizeof(struct foo) eliminates duplication of the *struct foo* identifier, which makes the code easier to read and also

ensures that if it is copy-and-pasted it won't require as much editing.

- String Duplications: The functions strdup() and strndup() can not accept a NULL argument. However, the ast_strdup and ast_strdupa functions will happily accept a NULL argument without generating an error. Furthermore, it is unnecessary to have code that malloc/calloc's for the length of a string (+1 for the terminating '\0') and then using strncpy to copy the copy the string into the resulting buffer. This is the exact same thing as using ast_strdup.

Following, a critical aspect is deeply detailed which is the locking in Asterisk. That's due because Asterisk has a multi threaded architecture and the access to shared memory is critical in such environment.

And the last guidelines of this part is focused in finishing up the contribution properly before submitting our code:

- Look at the code once more. When you achieve your desired functionality, make another few refactor passes over the code to optimize it.

- Read the patch . Before submitting a patch, *read* the actual patch file to be sure that all the changes you expect to be there are, and that there are no surprising changes you did not expect. During your development, that part of Asterisk may have changed, so make sure you compare with the latest SVN.

- Listen to advice . If you are asked to make changes to your patch, there is a good chance the changes will introduce bugs, check it even more at this stage. Also remember that the bug marshal or co-developer that adds comments is only human, they may be in error.

- Optimize, optimize, optimize. If you are going to reuse a computed value, save it in a variable instead of recomputing it over and over. This can prevent you from making a mistake in subsequent computations, making it easier to correct if the formula has an error and may or may not help optimization but will at least help readability.

3.4.2.2. Code structure and build architecture

This part of the document is still being written so we found uncompleted information. Basically the Asterisk's build architecture relies on the autoconf to detect the system configuration and on a locally developed tool, which is called menuselect, to select the build options and modules list, and on gmake to do the build.

4. Contributing to Asterisk project

In this point of the master thesis, once we have introduced ourselves to the Asterisk community and understood the Asterisk PBX, we are prepared to start with our contribution to the Asterisk project. The Asterisk project defines a number of projects, named Janitor Projects, which are intended to be a starting point for new developers. As we are new developers for the Asterisk project, we will start contributing to it in this way. Those Janitor projects mainly are some code maintenance tasks or performance improvement tasks, for which core developers haven't enough time to develop, and then they have defined these Janitor projects to serve as an starting point for new developers.

4.1. Contributing with a Janitor Project

Inside the Asterisk's subversion repository there is maintained a file named *janitor-projects.txt* [4] which contains the list of the proposed projects on this moment. In this moment the projects offered are the following:

- Audit uses of usleep() to ensure that the argument is never greater than 1 million. On some systems, that is considered an error. In any such cases, convert the usage over to use nanosleep(), instead.
- There are a bunch of places where the result of pbx_builtin_getvar_helper() gets stored and used. This is not thread safe. This code should be replaced with the following thread-safe version:

```
const char *var;

ast_channel_lock(chan);

if ((var = pbx_builtin_getvar_helper(chan, "MYVAR"))) {

      var = ast_strdupa(var);

}

ast_channel_unlock(chan);
```

- Convert all existing uses of astobj.h to astobj2.h
  - (chan_sip already in progress in a branch)

- There are many places where large character buffers are allocated in structures. There is a new system for string handling that uses dynamically allocated memory pools which is documented in include/asterisk/stringfields.h. Examples of where they are currently used are the ast_channel structure defined in include/asterisk/channel.h, some structures in chan_sip.c, and chan_dahdi.c.

- There is a convenient set of macros defined in include/asterisk/linkedlists.h for handling linked lists. However, there are some open-coded lists throughout the code. Converting linked lists to use these macros will make list handling more consistent and reduce the possibility of coding errors.

- Clean up and add Doxygen Documentation. When generating the documentation with make progdocs, a lot of warnings are generated. All of these need to be fixed. There is also plenty of code that still needs to be documented. All public API functions should be documented. That is pretty much anything in include/asterisk/*.h.

- Check all ast_copy_string() usage to ensure that buffers are not being unnecessarily zeroed before or after calling it.

- Find any remaining open-coded struct timeval manipulation and convert to use new time library functions.

- Use the ast_str API in strings.h to replace multiple calls to strncat(), snprintf() with funky math, etc.

- Audit all channel/res/app/etc. modules to ensure that they do not register any entrypoints with the Asterisk core until after they are ready to service requests; all config file reading/processing, structure allocation, etc. must be completed before Asterisk is made aware of any services the module offers.

- Ensure that Realtime-enabled modules do not depend on the order of columns returned by the database lookup (example: outboundproxy and host settings in chan_sip).

As we can see, most of them consist on maintenance tasks that allow new developers to take a first contact with the Asterisk's source code and don't require heavy changes for Asterisk so they have low risk for the project.

4.1.1. Choosing a project

Once we have presented what are the Janitor Projects and which ones are

available, we are going to choose the project we will work on and start working on it. After asking some questions to the Asterisk development core team, we select the Janitor project to develop defined as: *There are many places where large character buffers are allocated in structures. There is a new system for string handling that uses dynamically allocated memory pools which is documented in include/asterisk/stringfields.h. Examples of where they are currently used are the ast_channel structure defined in include/asterisk/channel.h, some structures in chan_sip.c, and chan_dahdi.c.*

So this Janitor project consists on examining Asterisk source code and finding structures definitions which contain strings defined in the C standard way (`char *`), and replace the definition and usage of them by the macros provided by the headers file stringfields.h. With those macros we will provide dynamically allocated memory pools which will improve the memory management of Asterisk and then help to improve Asterisk's performance.

4.1.2. Project Analysis

In stringfields.h field there are defined objects and macros used to manage fields in structures without requiring them to be allocated as fixed size buffers or requiring individual allocations for each field. To use these utilities, we simply need to redefine the existing structure containing strings, which might look like the following code:

```
struct sample_fields{
      int x1;
      char *foo;
      char *bla;
      long x2;
}
```

using the declaration macro `AST_DECLARE_STRING_FIELDS` as follows:

```
struct sample_fields {
      int x1;
      AST_DECLARE_STRING_FIELDS(
            AST_STRING_FIELD(foo);
            AST_STRING_FIELD(bla);
      );
      long x2;
};
```

When an instance of this redeclared structure is allocated (either statically or

dynamically), the fields and the pool of storage of them must be initialized:

```
struct sample_fields *x;
x = ast_calloc(1, sizeof(*x));
if (x == NULL || ast_string_field_init(x, 252)) {
      if (x)
            ast_free(x);
      x = NULL;
      ... //handle error
}
```

Fields will point to an empty string by default, and will revert to that when `ast_string_field_set()` method is called with a NULL argument. A string field will hence never contain NULL. Calling `ast_string_field_init(x, 0)` will reset fields to the initial value while keeping the pool allocated.

Reading the fields defined in the new way is much like using `const chat *` fields in the structure, so we cannot write to the field or to the memory it points to. Writing to the fields must be done using the wrapper macros listed below and assignments are always by value:

- `ast_string_field_set()` stores a simple value.
- `ast_string_field_build()` builds the string using a printf-style format.
- `ast_string_field_build_va()` is the version with variant arguments of the above (for portability reasons it uses two vararg arguments).
- variants of these function allow passing a pointer to the field as an argument.

Finally, when the structure instance is no longer need, the fields and their storage pool must be freed as follows:

```
ast_string_field_free_memory(x);
ast_free(x);
```

4.1.3. Development of the Janitor Project

To develop this Janitor project first of all we will examine Asterisk source project searching structs which contain string fields that will be candidates to be translated its definition to use the *stringfields.h* macros and take advantage of the dynamically allocated memory pools. As the Asterisk project has got so many files of source code, we need to delimit our activity to some of them

because if not it will became difficult to handle, specially for testing purposes.

4.1.4. Commit changes to the project

Once we have finished this Janitor project we need to commit the changes to the repository. If we were Asterisk's core developers, we would have write access to the Asterisk's Subversion repository so we could commit the changes immediately to the repository. This is done in this way only if the changes are trivial and we are sure of them. If the changes are more complex it's recommended to post them on the review board tool (see section 2.1.5) so the changed can be reviewed by the rest of Asterisk's developers community prior to committing them to the repository.

In our case, as we are new Asterisk's developers, we haven't write access to the repository so the way we commit the changes is different. First of all we need to extract our changes in the source code by creating a patch with `svn diff` command. With this command we will create a .diff file which contains the difference between the local files and the repository's trunk version one. So this patch can be applied to any copy of the trunk version of the repository with the `patch` command. We need to do this because our changes must be reviewed by some Asterisk's core developer with write access to the repository, and then, if he wants to test our changes, he needs a way to obtain the code and with patches this can be done easily. It's also very important to mention the Subversion's revision in which is based our patch since there could be some code commitments to the repository that could modify some of the files modified by us.

Once we have the patch extracted from the source code, we need to create an issue in the Mantis bug tracker in order that the core developers are able to see our changes. To create the issue we need to go to the bug tracker page [8] and login with the user that we created on section 2.1.4. Once we are logged, we click on the *Report Issue* link. On the first page, we need to select the project in which we want to create the issue, because in the bug tracker there also are tracked the Asterisk's satellite projects (see section 3.4.1). In our case we need to select Asterisk project (see fig. 4).

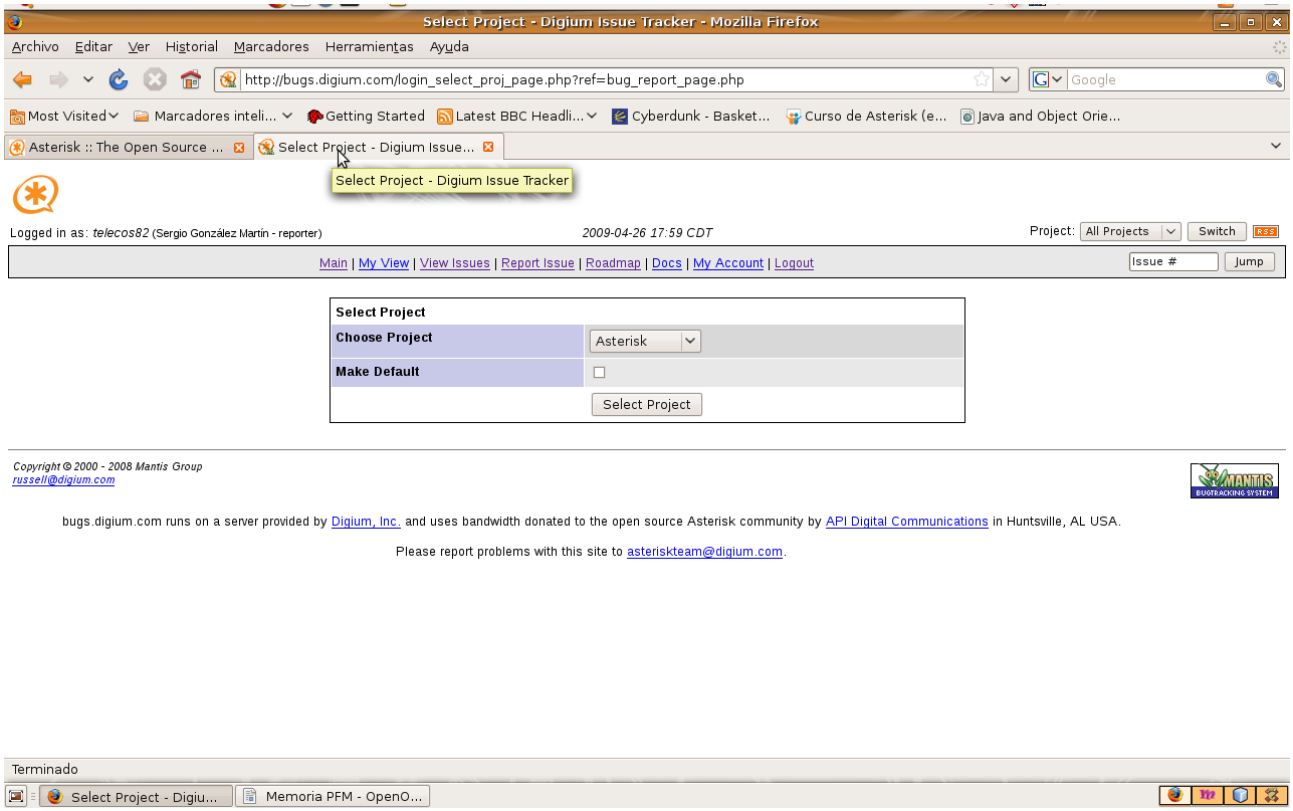

Fig. 4: Report issue - Page 1

Done that, on the next page we need to provide the information related with the bug that we are willing to open (see fig.5 and fig. 6). The first field asked is the category. On this field we should specify the Asterisk module where the issue applies. In our case we should select “General” as this Janitor project applies on all the Asterisk source code, not any specific module. The next fields are Reproducibility and Severity of the bug. As, in this case, what we are reporting isn't explicitly a bug, the Reproducibility field will be set to “N/A” (not apply), and Severity to the lesser possible, which is “feature”.

Next we need to specify the Product Version, which is the version of Asterisk where the resolution of the bug will be included. In this case, as we are working on the Subversion's trunk version of Asterisk we will let this field in blank as it doesn't applies. Following we need to provide a Summary of the issue and a deeper description of the bug which describes what we are reporting. In this case we should specify that we are sending code that implements the chosen Janitor Project.

Finally we need to provide Asterisk's version information. In the first field of this block we need to select the Asterisk's version where the bug was found. If it were a bug and it was present in previous version, there should be specified the older version where the bug exists. In our case, as we are working with Subversion's trunk version we should specify “SVN”. The next fields only apply if we are on a Subversion version, which is our case. We should specify in which branch we where working, in our case “Trunk”, and the revision on which we

were working and from where we have extracted the diff files. In our case “192876”.

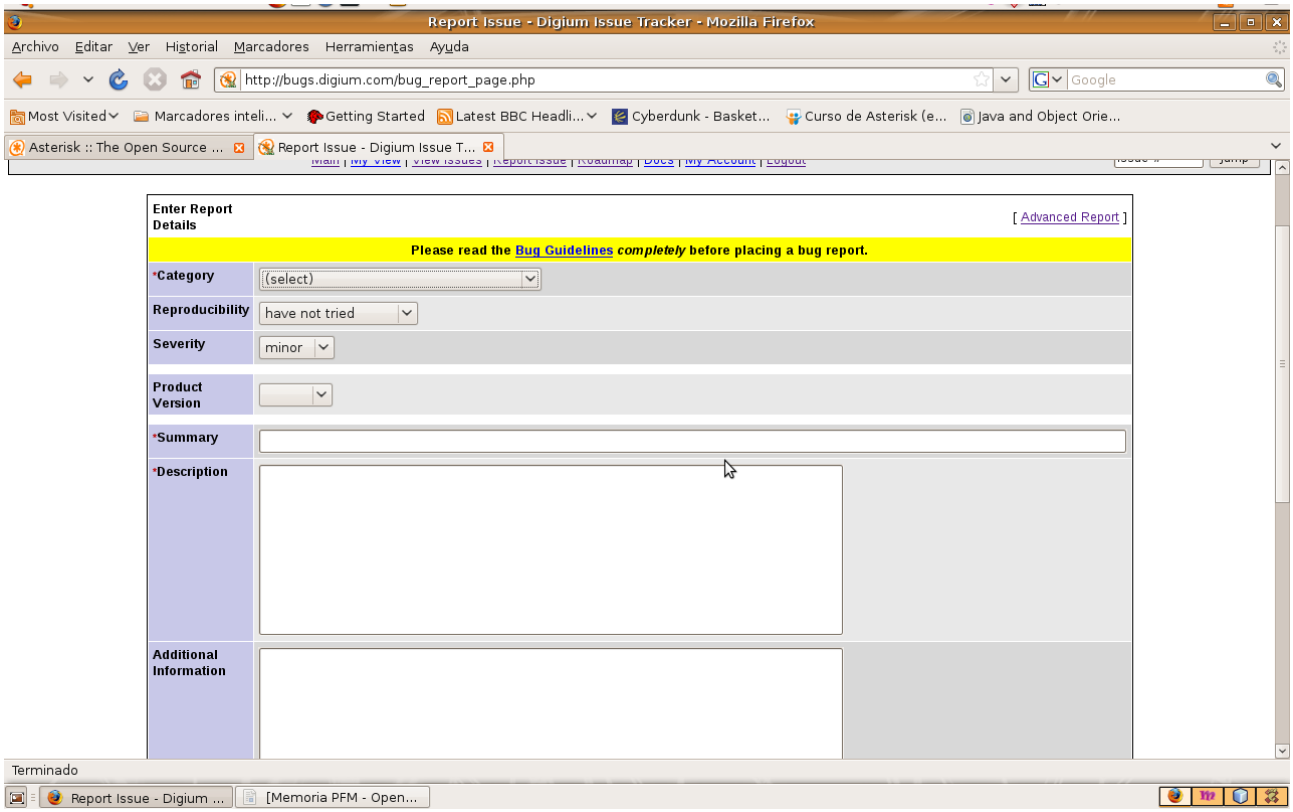

Fig. 5: Report issue – Page 2 (1)

With that we have reported our contribution to the Asterisk's project and we need to wait to have our code reviewed by a *bug marshal.* Then, once the code is reviewed, the *bug marshal* has made some comments regarding the naming of some string fields that we have changed to use the stringfields API. After those comments we correct the naming and extract a new .diff file with the new code. With those changes our code has been accepted and committed to the Asterisk's trunk version.

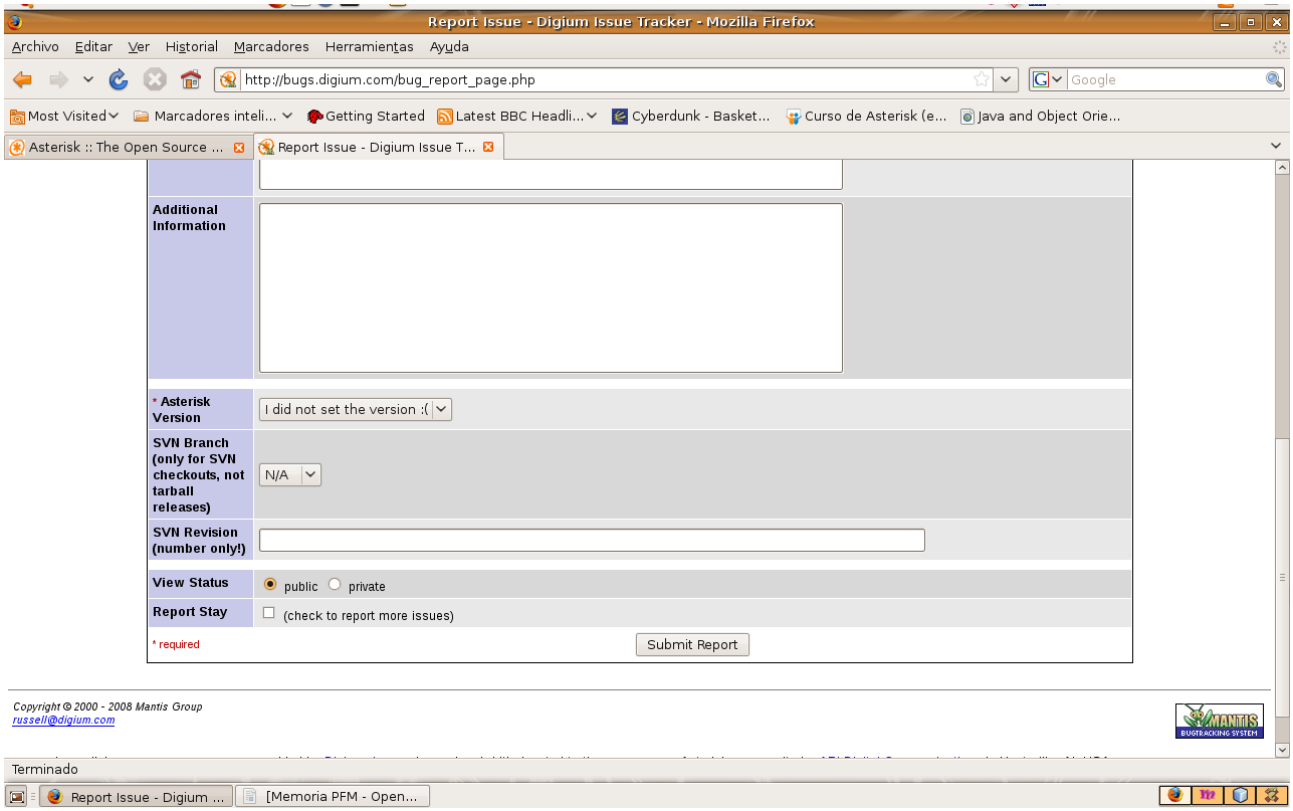

Fig. 6: Report issue – Page 2 (2)

4.2. Contribution 1: Reporting a bug

After an update of our local Asterisk project from the Subversion repository we achieved a state where the source code didn't compile. Thus we have found a bug in the Subversion's trunk version of Asterisk. Prior to reporting the bug, we should investigate the cause of the compilation error and check if we are able to solve it. Then we should report the bug and provide the resolution if we were able to solve it.

4.2.1. Extracting information about the bug

In this case, the output that we obtain when compiling Asterisk is the following:

```
Generating embedded module rules ...
   [CC] app_osplookup.c -> app_osplookup.o
app_osplookup.c: En la función ‘osp_create_provider’:
app_osplookup.c:197: error: ‘OSPT_CERT’ no se declaró aquí (primer uso en esta función)
```

```
app_osplookup.c:197: error: (Cada identificador no declarado solamente se reporta una vez
app_osplookup.c:197: error: para cada funcion en la que aparece.)
app_osplookup.c:197: error: expected ‘;’ before ‘localcert’
app_osplookup.c:198: error: expected ‘;’ before ‘cacerts’
app_osplookup.c:199: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribute__’ before ‘*’
token
app_osplookup.c:199: error: ‘pcacerts’ no se declaró aquí (primer uso en esta función)
app_osplookup.c:339: error: ‘localcert’ no se declaró aquí (primer uso en esta función)
app_osplookup.c:343: error: ‘cacerts’ no se declaró aquí (primer uso en esta función)
app_osplookup.c: En la función ‘osp_validate_token’:
app_osplookup.c:565: error: ‘OSPC_NFORMAT_E164’ no se declaró aquí (primer uso en esta
función)
app_osplookup.c: En el nivel principal:
app_osplookup.c:629:   error:   expected   declaration   specifiers   or   ‘...’   before
‘OSPEFAILREASON’
app_osplookup.c: En la función ‘osp_check_destination’:
app_osplookup.c:633: error: ‘OSPE_DEST_OSPENABLED’ no se declaró aquí (primer uso en esta
función)
app_osplookup.c:633: error: expected ‘;’ before ‘enabled’

….
```

Inspecting the output we can find some constants that are not found by the compiler. We should search the code, concretely in the app_osplookup.c source file which is generating all the errors, and look for those constants. If we search in this file we can see that the constants OSPT_CERT, OSPC_NFORMAT_E164, ... are used but aren't defined. As those compiling errors where introduced after an update of the Subversion code, we will check the history of that file and try to revert the code to a previous revision.

If we revert the code to the revision 190829, then Asterisk compiles with no errors, so we can assure that the problem was introduce in this revision and it isn't due to any change on our local environment. To know in which consist the change in app_osplookup.c for this revision we should check the Subversion commit comments which explains in a brief the changes done in a concrete commit. In this case it says “Updated for OSP Toolkit 3.5.”, which means that has been updated to work with a newer version of OSP library and it has broken compatibility with previous versions of OSP.

If we check the changes we can see that some constants and some functions have changed its names so it makes it not compatible with previous versions of OSP library. In our case we have installed version 3.4.2, which is the default of Ubuntu's repositories.

The code of app_osplookup.c is maintained directly by TransNexus [26], which are the developers of OSP libraries so we can't work on this issue because we have not a proper knowledge on the aforementioned libraries. So our contribution with this bug will consist and a bug report and, once a patch is provided to allow compatibility with previous versions of OSP libraries, test that patch.

4.2.2. Reporting the bug

Once we have investigated and obtained all the required information about the bug we can proceed with the bug report on the Asterisk's bug page [8]. First of all we should check that there doesn't exist any bug about the same issue reported. To do that we should use the the filters provided in the Asterisk's bug tracking page to search any bug related with OSP libraries. After searching we find an open bug, the 14988, which is about this compile error. So in this point we can only confirm the bug and expect for a future patch from TransNexus to test it. To keep us updated with the last modifications on the bug we will monitor the bug through Mantis.

4.3. Contribution 2: Working on a bug solution

The next contribution that we will perform will consist on solving an existing bug. To do that, first of all we need to select a bug in “new” state and reported over the Subversion's trunk version of Asterisk. In this case we have found the bug 15020 on the Mantis web page [8]. This bug consists on an Asterisk's crash when using res_clialiases module. This module allows to define some command aliases for the Asterisk CLI commands (see section 3.3.3). The problem here is that in some cases, when working with res_clialiases enabled and using the default aliases defined, there is a segmentation fault when using the auto completion tab key or when executing some alias commands.

4.3.1. Working on the bug

The bug was reported specifying that, when enabling res_clialiases modules, and using default clialiases.conf configuration file, there are some cases that cause an Asterisk crash due to a segmentation fault. The res_clialises modules allows to define aliases for CLI commands in order to make them customizable.

First of all we need to check if we can reproduce the crash in our local environment. To do that we need to enable the module, which means selecting it in the `make menuselect` program, and uncomment the default aliases defined in order to have them available from the console. As we have the option load all modules enabled we don't need to force the load of this module on start up because it will be don automatically. In the bug description is specified a case that provokes a crash which is the “dial” alias. This is an alias for the CLI

command “console dial”. It says that if we write dial on CLI interface and press TAB key to show the available options or if we press return key to execute the command we get an Asterisk crash. We do that and then we get the segmentation fault described on the bug.

The next step will be to reproduce the Asterisk crash while debugging Asterisk in order to extract information from the back trace called just before the crash. For that, we need to open asterisk with GDB by executing:

```
gdb asterisk
```

Once we are on the GDB console we need to provide the arguments which we want to use to start Asterisk with GDB command `set args`. In this case, as we don't want to start Asterisk as a separate process, we need to set the -c option, and as we want minimal verbose we will set verbosity level to 1 with -v. Thus we will execute:

```
(gdb) set args -cv
```

And finally we start asterisk with GDB command `run`:

```
(gdb) run
```

Once Asterisk has been started and we are on the command prompt we introduce the alias which causes the segementation fault which is dial. Once we get the crash, we execute GDB command bt (backtrace) which show use the function executed that caused the segmentation fault:

```
*CLI> dial
Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0xb7a6c6d0 (LWP 14537)]
0xb7b04a4c in ?? () from /lib/tls/i686/cmov/libc.so.6
(gdb) bt
#0  0xb7b04a4c in ?? () from /lib/tls/i686/cmov/libc.so.6
#1  0xb7b069c5 in malloc () from /lib/tls/i686/cmov/libc.so.6
#2  0xb7b0a350 in strdup () from /lib/tls/i686/cmov/libc.so.6
#3  0x080a5f30 in parse_args (s=0xb7bf1ff4 "|\025", argc=0xbf27d214, argv=0xbf27d0f0,
max=64, trailingwhitespace=0xbf27d210)
    at /usr/src/asterisk/include/asterisk/utils.h:521
#4  0x080a6982 in __ast_cli_generator (text=0x846f708 "console active", word=0x82abe7d
"", state=0, lock=1) at cli.c:2234
#5    0xb8041302  in  cli_alias_passthrough  (e=0x846f690,  cmd=-3,  a=0xbf27d470)  at
res_clialiases.c:106
#6  0x080a6be6 in __ast_cli_generator (text=<value optimized out>, word=0x82abe7d "",
state=0, lock=1) at cli.c:2295
#7    0xb8041302  in  cli_alias_passthrough  (e=0x846f690,  cmd=-3,  a=0xbf27d6f0)  at
res_clialiases.c:106
#8  0x080a6be6 in __ast_cli_generator (text=<value optimized out>, word=0x82abe7d "",
state=0, lock=1) at cli.c:2295
#9    0xb8041302  in  cli_alias_passthrough  (e=0x846f690,  cmd=-3,  a=0xbf27d970)  at
res_clialiases.c:106
#10 0x080a6be6 in __ast_cli_generator (text=<value optimized out>, word=0x82abe7d "",
state=0, lock=1) at cli.c:2295
….
```

From this traces we can see that the problem is due to an infinite loop since there appear a lot of calls to cli_alias_passthrough. Once we have located the

problem in one function, we can examine the cli_alias_passthrough function, defined on source file res_clialiases.c. The part which is involved in this crash is the following, which is the code that is called when cli_alias_passthrough is called from ast_cli_generator, which is the case of study:

```
static char *cli_alias_passthrough(struct ast_cli_entry *e, int cmd, struct ast_cli_args *a){
….
case CLI_GENERATE:
      line = a->line;
      line += (strlen(alias->alias));
      if (!ast_strlen_zero(a->word)) {
            struct ast_str *real_cmd = ast_str_alloca(strlen(alias->real_cmd) + strlen(line) + 1);
            ast_str_append(&real_cmd, 0, "%s%s", alias->real_cmd, line);
            generator = ast_cli_generator(ast_str_buffer(real_cmd), a->word, a->n);
      } else {
            generator = ast_cli_generator(alias->real_cmd, a->word, a->n);
      }
      ao2_ref(alias, -1);
      return generator;
….
}
```

It seems that the problem is that the function is being called recursively infinite times, which causes a segmentation fault. The problem here is that the cli_alias_passthrough is prioritizing an alias over a command which has the same name (console in this case). So we should control that if an alias is exactly the same that a real command, it shouldn't do any treatment for this alias. This can be done by adding a new if clause just before existing one as follows:

```
….
if (!strncasecmp(alias->alias, alias->real_cmd, strlen(alias->alias))) {
      generator = NULL;
} else if (!ast_strlen_zero(a->word)) {
….
```

Finally we need to test our solution with the situations that cause the segmentation fault. To do that we modify the source file res_clialiases.c adding the previous code, recompile asterisk with `make` and `make install` commands and start Asterisk with gdb to examine results if the crash persists. After we have assured that this works is the time to contribute our solution to the bug.

4.3.2. Contributing the resolution

To contribute the solution we would work through the Asterisk's bug tracking system since this bug has an easy resolution. As we have not write access to Subversion and thus can't commit our changes immediately, we added a comment to the bug page describing in what consisted the problem an the solution that we have developed. Then, as we are waiting for a response, we click on monitor the bug in order to be noticed every time a change is done in that bug.

After a pair of days, Russell Bryant, one of the bug Marshall of the project, responses us agreeing with the solution and commiting the changes that we have developed that in this case consist on a few lines of code.

4.4. Contribution 3: Testing a patch for a new feature

As the next contribution we will perform a contribution in a different way than the previous. After several conversations through the IRC chat with the Asterisk's core developers, they told us that the biggest needs of the project in this moment are the testing contributions since there are a lot of new developments that require huge tests to be performed. So the next contribution will consist on selecting a patch submitted to the bug tracking system, apply the patch on our Asterisk source code and perform tests to check if the patch is valid or not. Among all the available bugs we will select one that affects the trunk version of Asterisk because we are working on it.

The bug that we have selected for this contribution, is the 15135 which consists on a new feature for Asterisk. This new feature provides dynamic parking lots, which means that allows to define parking lots dynamically for instance, through the dial plan execution. A parking lot is where a call can be placed in a waiting state to be attended in the future.

4.4.1. Testing the patch

To test the patch first of all we need to apply the patch to our local Asterisk. To do that, first of all we need to check out our Asterisk source code to the revision specified on the bug, which in this case is the revision number 194982 of the trunk version, which is where we are working. To do that we execute:

```
svn update -r 194982
```

Once we have established the same base source code from where the patch was extracted we can apply it assuring that it will be applied correctly, by executing:

```
wget 'https://issues.asterisk.org/file_download.php?file_id=22683&type=bug' -O -
| patch -p0
```

After that we have our code with the patch applied and we are ready for testing the patch. The first thing that we have to test is that Asterisk can be compiled properly and it doesn't gives any compilation message related with the patch. We check that the compilation is correct and the next step will be reviewing the code to have a better understand of the new feature and to detect if the source code added accomplish with the project's coding guidelines. To do that, in our NetBeans IDE, we do a diff from our local main/features.c file (which is the only file affected for this patch) with the file of the repository at revision 194982, and we will see the differences in a easy way.

Once we have reviewed the source code, the only error that we have seen is that there is a string channel variable that is copied using the `ast_strdup()` function which creates a new copy of the string, so it isn't required to lock the channel during it's use because it's only reading the channel variable to make a copy as it has been done. Then we need to add a comment on the Mantis issue to remark the problem that we have detected and the author of the patch could solve it properly.

Finally, once we have checked that it follows coding guidelines, the next step will be to check that the new functionality works properly. To check if the functionality works properly, first of all we need to configure an environment that allow us to make use of the new functionality. To do that we need to activate the dynamic parking lots which can be done adding a new configuration parameter named *parkeddynamic,* set to yes, onto the features.conf configuration file. If this variable isn't set to yes, then the parking lots will be created statically as normally.

The next step will be defining an extension which makes use of dynamic parking lots. That is an extension which calls to the Park() dial plan application after setting the channel variables ${PARKINGLOT} to define the parking lot to be used, ${PARKINGDYNCONTEXT} to define the context of the parking lot and ${PARKINGDYNPOS} to define the position inside the parking lot where the hold call will be placed. The dial plan that we will use to test this patch will be the following:

```
exten => test,1,Answer()
exten => test,n,Set(PARKINGLOT=parklot)
exten => test,n,Set(PARKINGDYNCONTEXT=parkcontext)
exten => test,n,Set(PARKINGDYNPOS=1-20)
exten => test,n,Park()
exten => test,n,HangUp()
```

Once we have configured our environment to enable the dynamic parking lots we need to test its functioning. This will consist on generate calls and place them to the parking lots, checking if they have been created dynamically by inspecting the CLI output and the asterisk debug information available on log files (located under /var/log/asterisk/ directory).

After making a test call we see, inspecting the CLI output, that the behaviour is as expected:

```
    -- Executing [test@from-internal:1] Answer("SIP/4001-0a3ce360", "") in new
stack
    -- Executing [test@from-internal:2] Set("SIP/4001-0a3ce360",
"PARKINGLOT=parklot") in new stack
    -- Executing [test@from-internal:3] Set("SIP/4001-0a3ce360",
"PARKINGDYNCONTEXT=parkcontext") in new stack
    -- Executing [test@from-internal:4] Set("SIP/4001-0a3ce360",
"PARKINGDYNPOS=1-20") in new stack
    -- Executing [test@from-internal:5] Park("SIP/4001-0a3ce360", "") in new
stack
  == Parked SIP/4001-0a3ce360 on 701 (lot default). Will timeout back to
extension [from-internal] s, 1 in 45 seconds
    -- Added extension '701' priority 1 to parkedcalls (0xa3f3a28)
    -- <SIP/4001-0a3ce360> Playing 'digits/7.gsm' (language 'en')
    -- <SIP/4001-0a3ce360> Playing 'digits/0.gsm' (language 'en')
    -- <SIP/4001-0a3ce360> Playing 'digits/1.gsm' (language 'en')
    -- Started music on hold, class 'default', on SIP/4001-0a3ce360
```

To check the parked calls that we have in one instant we can execute CLI command `parkedcalls show`:

```
*CLI> parkedcalls show
 Num  Channel           (Context          Extension   Pri ) Timeout
*** Parking lot: parlot
701   SIP/4001-0a3ce360 (parkcontext      s           1   ) 24s
---
1 parked call in total.
```

And we can see how the parked call has been placed in the context and lot that we have dynamically specified. After several tests placing several calls concurrently on call we see an strange behaviour because it seems that in some cases it isn't retrieving the channel variables properly. Reviewing the code we can see that the code that retrieves those channel variables isn't locking the channel while doing it so it:

```
parkinglotname_copy = ast_strdupa(S_OR(pbx_builtin_getvar_helper(chan,
"PARKINGDYNAMIC"),""));
dyn_context = ast_strdupa(S_OR(pbx_builtin_getvar_helper(chan,
"PARKINGDYNCONTEXT"),""));
dyn_range = ast_strdupa(S_OR(pbx_builtin_getvar_helper(chan,
"PARKINGDYNPOS"),""));
```

This should be protected in some way by locking the channel before this piece of code and unlocking after because if some of the channel variables is changed while any other is being read it could bring to incoherent states.

4.4.2. Giving feedback from the patch

The last point of this contribution will be giving the feed back obtained to the author of the patch in order to help him, which will be our contribution to the Asterisk project. In the feedback we should comment to the author the problem about the channel variables that we have found and that the new feature work properly a part of it. We will report it as a comment to the Mantis bug tracking system and will monitor the issue to be aware about changes occurred on it. When a newer version of the patch is available we should apply it to our source code and test again the feature to check if the problem that we found has been properly solved.

4.5. Contribution 4: Testing a patch to solve a bug

The last contribution to the Asterisk project that we will start in the framework of this final Master thesis will be the test of a patch for solving a found bug. In this case, the bug number 15234, the reporter found an Asterisk crash when unloading a module in certain conditions. The reporter of this bug is an Asterisk core developer, so he has write access to the Subversion repository, but he has done a request for testing of the patch prior to committing the code to the repository by creating a new issue on the project's bug tracking system.

4.5.1. Testing the patch

The patch affect the res_timing_dahdi.c source file, which solves the bug when unloading the module if there is no *dahdi* timing interface registered on that machine. This patch adds a control prior to unloading the module, which check if the machine has a *dahdi* interface prior to unloading it.

To test the patch first of all we need to update our source code to the revision in which the patch is based (in this case revision number 198375), by executing:

```
svn update -r 198375
```

and the apply the patch to our local copy executing the following command on asterisk source code directory:

```
wget 'https://issues.asterisk.org/file_download.php?file_id=22860&type=bug' -O -
```

```
| patch -p0
```

Once we have the patch applied we need to recompile asterisk with it and test it. To do that first of all we need to force the condition of the crash, which is having no *dahdi* interface in our machine and then unload the module to check if it makes Asterisk crash.

To force that we have no *dahdi* interface we remove the *dahdi* packages from our system. Then we try to unload the module and check that Asterisk doesn't crash, so the patch works properly.

4.5.2. Giving feedback from the patch

Finally we need to give feedback to the author about the patch to help them to commit the source code to the repository. We can do it either adding a comment to the bug or either through the #asterisk-dev IRC channel if we find him available. In this case, as we have found eliel available through the IRC we send him a message telling him that the patch works fine for us.

5. Conclusions

With these contributions to the Asterisk open source project we have studied the whole project and its community and started a contribution with it. The study of the community and the Asterisk PBX itself has implied more work than expected initially because the Asterisk project is a project in a mature state, which has reached an state of bigness (in terms of source code lines).

For that reason, learning Asterisk functioning and studying it's source code has been a hard task but it is very helpful in order to perform better contributions to the project as having a good Asterisk background is necessary.

This fact has implied that we have been able to perform less contributions to the Asterisk project as we initially expected, because it has been necessary to dedicate more time to the study of Asterisk, since this final Master Thesis has been conceived as an starting point of contributing with an open source community, which means that I will continue contributing to the project.

In addition to this, when we started the project we didn't define any specific Asterisk area where we could contribute because we didn't know which implications could have doing any type of contribution. On this point of the contribution with the project, we have seen that as Asterisk is a big modular solution, there are some modules which require specific hardware (for instance an E1/T1 card) or specific knowledge (for instance knowledge on OSP libraries), so our contributions cannot very helpful with them.

Furthermore there are some Asterisk features which require an special and complex configuration in order to use it (and thus test it), as for instance, an stress environment which we currently have not available, so we cannot perform contributions about them. We haven't focused in prepare them since it is outside the scope of this final Master thesis, but for future contributions we will prepare such environments by creating some virtual machines that allow us to have different environments on our local machine.

Due to the time limitation of this final Master thesis, there are some contributions that are unfinished because we are expecting feedback from other person, and it has overpassed the final Master thesis dead line. Despite this, the contribution work has been include in this document as is interesting to see the different types of contributions available and the different work type required for every kind.

All these imply that the contributions that we have been able to perform, have been very limited because, due to the temporal limitation of this final Master thesis, we haven't been able to prepare specific environments for them. For future contributions (after this final Master thesis) we will prepare some environments, as an stress test environment using virtual machines, which will

let us to wide our possibilities of contribution with the Asterisk project.

So from this final Master thesis we have get an strong knowledge on the operation of Asterisk open source community, we have also get knowledge on the Asterisk PBX from the point of view of configuration and usage firstly, and lately with the source code understand, and finally we have started a contribution with the Asterisk project which we expect to continue for years.

Working with the Asterisk community we have seen that, as an open source community, all the information is freely available. For that reason, it is very appreciated in new developers being self-taught, and thus, some core developers aren't open to answer “dummy” questions as they can be solved by reading the available documentation.

References

[1] Official web page of the Free Software Master studies: http://www.uoc.edu/masters/oficials/estudis/master_oficial_programari_lliure_estudis.htm

[2] Official web page of Asterisk project: http://www.asterisk.org

[3] Presence Technology: http://www.presenceco.com

[4] Updated list of the Asterisk's Janitor Projects: http://svn.digium.com/view/asterisk/trunk/doc/janitor-projects.txt?view=markup

[5] J. Van Meggelen, L. Madsend and J. Smith, *Asterisk, the future of telephony*. 2nd edition, O'Reilly Media.

[6] Asterisk's distribution lists: http://www.asterisk.org/support/mailing-lists

[7] Asterisk code repository: http://svn.digium.com/svn/

[8] Asterisk's Mantis bug tracking system: http://bugs.digium.com/

[9] Asterisk's review board: http://reviewboard.digium.com/

[10] Freenode IRC channels: http://www.freenode.net/

[11] Asterisk wiki: http://www.voip-info.org/

[12] Asterisk documentation: http://www.asteriskdocs.org/

[13] Asterisk's official forums: http://forums.digium.com/index.php?c=1

[14] Asterisk's knowledge base: http://kb.digium.com/

[15] Asterisk's official blogs: http://blogs.digium.com/

[16] FreeSWITCH project: http://www.freeswitch.org/

[17] SipX project: http://www.calivia.com/sipxpbx

[18] OpenPBX project: http://www.voicetronix.com.au/openpbx/

[19] PBX4Linux project: http://pbx4linux.net/

[20] Callweaver project: http://www.callweaver.org/

[21] AsteriskNOW project: http://www.asterisknow.org/

[22] IAX2 IETF standardization draft revision 5: http://www.ietf.org/internet-

drafts/draft-guy-iax-05.txt

[23] FreePBX project main page: http://www.freepbx.org

[24] Asterisk's project coding guidelines: http://svn.digium.com/svn/asterisk/trunk/doc/CODING-GUIDELINES

[25] TransNexus web page: http://www.transnexus.com

Appendix A – Creative Commons BY-NC-SA License

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. ***"Adaptation"*** *means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.*

b. ***"Collection"*** *means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(g) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A*

work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. **"*Distribute*"** *means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.*

d. **"*License Elements*"** *means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.*

e. **"*Licensor*"** *means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.*

f. **"*Original Author*"** *means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.*

g. **"*Work*"** *means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.*

h. **"*You*"** *means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.*

i. **"*Publicly Perform*"** *means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to*

broadcast and rebroadcast the Work by any means including signs, sounds or images.

j. **"*Reproduce*"** *means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.*

2. Fair Dealing Rights. *Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.*

3. License Grant. *Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:*

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
d. to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights described in Section 4(e).

4. Restrictions. *The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:*

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that

restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(d), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(d), as requested.

b. *You may Distribute or Publicly Perform an Adaptation only under: (i) the terms of this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-NonCommercial-ShareAlike 3.0 US) ("Applicable License"). You must include a copy of, or the URI, for Applicable License with every copy of each Adaptation You Distribute or Publicly Perform. You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License. You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.*

c. *You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in con-nection with the exchange of copyrighted works.*

d. *If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the*

Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(d) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

e. *For the avoidance of doubt:*
    i. ***Non-waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;*
    ii. ***Waivable Compulsory License Schemes**. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,*
    iii. ***Voluntary License Schemes**. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).*
f. *Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to*

the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING AND TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THIS EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. *EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.*

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. *Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.*
c. *If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.*
d. *No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.*
e. *This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.*
f. *The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.*

Creative Commons Notice

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons

without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.

Appendix B - Digium Open Source Software Project Submission Agreement

Digium Open Source Software Project Submission Agreement v3.0

****

IMPORTANT - PLEASE READ CAREFULLY:

This document ("Agreement") constitutes a legal agreement. By signing this Agreement below, You, either an individual or the organization indicated below ("You"), agree to be legally bound. You may want to consult an attorney before signing. You acknowledge that you are entering into this Agreement in consideration of the opportunity to contribute to and participate in Digium's open source software projects, which opportunity is of value to You.

NOTE: If you sign this agreement now, it will NOT apply to any patches you have already uploaded to the issue tracker. If your patches were uploaded after June 28th, 2007 then you must delete them and re-upload them after signing this agreement; a bug marshal cannot update them. If your patches were uploaded on or before that date and you had a disclaimer on file at that time, please contact a bug marshal to get a predated license agreement entered for your patches.

****

"*Submission*" *means any work of authorship, software code, documentation, creation, images or sound, provided by You to Digium via Digium's official project submission system, in human or machine readable form, at any time (both prior and subsequent to Your execution of this Agreement).*

You hereby grant Digium a perpetual, worldwide, royalty-free, irrevocable, non-exclusive, and transferable license to use, reproduce, prepare derivative works of, publicly display, publicly perform, distribute the Submissions, and to sublicense such rights to others. The rights granted may be exercised in any form or format, and Digium may distribute and sublicense to others on any licensing terms, including without limitation: (a) open source licenses like the GNU General Public License (GPL), or the Berkeley Science Division license (BSD); or (b) binary, proprietary, or commercial licenses. If Your Submission is derived from software released by Digium under the GPL, Digium as licensor thereof waives such requirements of the GPL as applied to that software to the limited extent necessary to allow you to provide the Submission and the foregoing license to Digium.

You hereby grant Digium a perpetual, worldwide, royalty-free, irrevocable, non-exclusive, sublicenseable and transferable license under any patent You own or

control, now or in the future, to make, have made, use, sell, offer for sale, or import Submissions or any modifications thereof, including without limitation any combinations of the Submissions or modifications thereof with software, technology or services of Digium or its affiliates.

You hereby represent that you are the sole and original author of all Submissions and that, to the best of your knowledge, the Submissions do not infringe upon the rights of any third party. If you are providing the Submission on behalf of an organization or you are an employee of an organization, the person signing this Agreement represents that he or she is expressly authorized to execute this Agreement on that organization's behalf. Except for the express representations set forth above, the Submission and all licenses granted above are made on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You acknowledge that the decision to include the Submission in any code base is entirely the decision of Digium, and this Agreement does not guarantee that the Submissions will be included in any code base. The parties agree that any facsimile copy of this Agreement will be binding upon the parties to the same effect as originals.