Institut **S**upérieur
d'**I**nformatique, de
Modélisation et de
leurs **A**pplications

Complexe des Cézeaux
BP 125
63173 Aubière Cedex

European
Organization for
Nuclear **R**esearch
Dpt LHCb

Geneva
Switzerland

Internship report of 3rd year
Focus Software engineering and Computing Systems

Development of a dynamic network monitoring tool

Interactive map based on LLDP and SNMP

Presented by Christophe HAEN

ISIMA responsible Emmanuel MESNARD

Enterprise responsible Enrico BONACCORSI

April-September 2010

Institut **S**upérieur
d'**I**nformatique, de
Modélisation et de
leurs **A**pplications

Complexe des Cézeaux
BP 125
63173 Aubière Cedex

European
Organization for
Nuclear **R**esearch
Dpt LHCb

Geneva
Switzerland

Internship report of 3rd year
Focus Software engineering and Computing Systems

Development of a dynamic network monitoring tool

Interactive map based on LLDP and SNMP

Presented by Christophe HAEN

ISIMA responsible Emmanuel MESNARD

Enterprise responsible Enrico BONACCORSI

April-September 2010

Thanks

First of All, I would like to thank Enrico BONACCORSI, my enterprise tutor, for all the attention he gave to my work, all the pedagogy that he showed me and everything he taught me. All those points made my internship a really pleasant and enriching experience.

I also would like to thank all the LHCb Online team for being so welcoming and friendly. All of them were always ready to answer any of my questions and satisfy my curiosity.

Thanks as well to Guoming LIU for all the help he gave me, and the experience he shared with me. His deep knowledges about the material used in LHCb were very useful to solve some of the problems I faced with.

Abstract

The European Organization for Nuclear Research (French: Organisation Européenne pour la Recherche Nucléaire), known as CERN is the world's largest particle physics laboratory, established in 1954 near Geneva. CERN's main function is to provide the particle accelerators and other infrastructure needed for high-energy physics research.

The LHCb (standing for "Large Hadron Collider beauty") experiment is one of six particle physics detector experiments built on the Large Hadron Collider, the world's largest and highest-energy particle accelerator. This experiment produces a huge amount of data which needs to be treated. This task is processed by some 2000 computing servers and 400 control servers. The LHCb Online team is responsible of the data, from their creation in the detector, till the storage.

The purpose of my internship was to develop a software from scratch able to dynamically discover the network, draw a map of it, gather information of the network equipments, and implement basic monitoring tasks. This allow to have a global overview of the complex infrastructure.

Keywords : CERN, LHC, LHCb, network, monitoring

Contents

List of Figures

Introduction

Within the context of my studies during the third year of engineer formation at ISIMA, I did my internship in the CERN laboratory, the world's largest particle physics laboratory, in Geneva (Switzerland). In the 15 person team of LHCb Online department, I achieved, under the direction of Mister Enrico BONACCORSI and between the 6th of April and the 2nd of September 2010, the design and the development of a networking tool, able to dynamically discover network equipments and gather part of their configuration, produce a map of it, and execute basic monitoring tasks.

The Large Hadron Collider (LHC) is the world's largest and highest-energy particle accelerator. Its purpose is to test various predictions of high-energy physics. The collisions of the particles take place in six different places all around the accelerator. There stand the six detectors : ATLAS, CMS, ALICE, LHCb, TOTEM and LHCf.
I took my internship in the LHCb experiment, and more precisely with the LHCb Online department.

Every time two particles collide in the detector, data about this event are produced. Because of the huge amount of collision per second, the total amount of data produced is too big to be stored, and one need to reduce it by selecting only interesting events. This task is processed by some 2000 computing servers and 400 control servers. In order to work with such a big infrastructure, it is interesting to have a tool that provides global overview of it.

The goal of my internship was to develop such a tool. The software was intended to discover dynamically the network and gather information about its equipments, draw a map of it, and perform basic monitoring tasks.

This report will be divided in three main parts. First of all, I will analyze the context in which I made my internship, this means description of the CERN and the LHCb, and the specifications of my project. In a second part, I will focus more on the application itself, and give more technical details. To finish, I will develop the difficulties I met and the solutions I found, and I will present you some results of the software.

1 Work environment and project specifications

1.1 Presentation of the laboratory and experiment

1.1.1 Presentation of CERN

The *European Organization for Nuclear Research* (French: *Organisation Européenne pour la Recherche Nucléaire*), known as CERN is the world's largest particle physics laboratory, situated in Geneva on the Franco–Swiss border.

The organization has twenty European member states, and is currently the workplace of approximately 2,600 full-time employees, as well as some 7,931 scientists and engineers (representing 580 universities and research facilities and 80 nationalities).

CERN's main function is to provide the particle accelerators and other infrastructures needed for high-energy physics research. Numerous experiments have been constructed at CERN by international collaborations to make use of them. It is also noted for being the birthplace of the World Wide Web. The main site at Meyrin also has a large computer centre containing very powerful data processing facilities primarily for experimental data analysis, and because of the need to make them available to researchers elsewhere, has historically been (and continues to be) a major wide area networking hub.

1.1.2 Presentation of the LHC

The *Large Hadron Collider* (LHC) is the world's largest and highest-energy particle accelerator. The LHC lies in a tunnel 27 kilometres in circumference, as much as 175 metres beneath the CERN.

It was built by the CERN with the intention of testing various predictions of high-energy physics, including the existence of the hypothesized Higgs boson and of the large family of new particles predicted by supersymmetry.

It is funded by and built in collaboration with over 10,000 scientists and engineers from over 100 countries as well as hundreds of universities and laboratories.

1.2 Presentation of the LHCb experiment

Six detectors have been constructed at the LHC, located underground in large caverns excavated at the LHC's intersection points. One of the detector is the LHCb, which stands

for *Large Hadron Collider beauty*. LHCb is a specialized b-physics [1] experiment, particularly aimed at measuring the parameters of CP violation in the interactions of b-hadrons.

After giving you an overview of the detector, I will describe you the data acquisition process.

1.2.1 The LHCb detector

Basically described, the LHC is like a huge circular tube in which we inject 2 groups of particles (called *beams*) in opposite directions. Those two beams will collide in predefined places : where the detectors are located.

Two particles smashing together will produce plenty of other particles that physicists want to analyze. The problem is that the life time of those new particles is really short, and we cannot store them. Thus, detector acts exactly as a radar : it takes a snapshot of the new particles produced.

As you can see on the figure 1, the detector of the LHCb is a concatenation of several layers. Each of them has a specific goal. We could say that each layer takes a different picture of the same event, which provides a few information. When we crosscheck all pictures of a given event, we have all the information we are looking for.

Here is a short description of the different layers (called sub-detector) :

- Vertex Locator : (VELO) the collision happen inside this "tube". It is used to measure the particle trajectories close to the interaction point.
- RICH-1 : it is used for particle identification of low-momentum tracks.
- Magnet : it changes the particle trajectory according to the type of particle
- TT, T1 to T3 : those elements constitute the tracking system. It is used to reconstruct the trajectories of charged particles and to measure their momenta.
- RICH-2 : it is used for particle identification of high-momentum tracks.
- SPD/PS : the Scintillator Pad Detector and the PreShower are parts of calorimetry system. Their job is to sort the particles in order to be well analyzed by the two following layers.

[1] From time to time in this report, I may use some physic terms. In order not to make the lecture to messy, I invite people interested in details about the physic done in LHCb to see the annexes

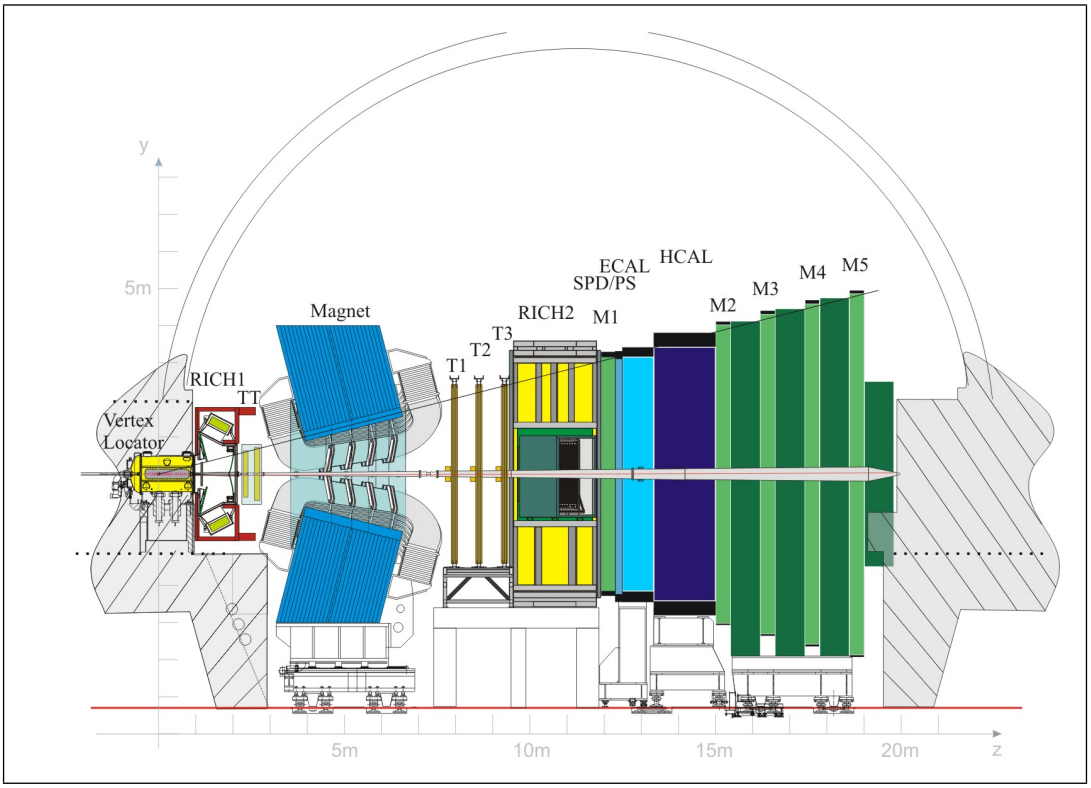

Figure 1: The LHCb detector

- ECAL/HCAL : the Electromagnetic and Hadronic CALorimeters are the last parts of the calorimetry system. They provides measurements of the energy of some particles.
- M1 to M5 : this is the muon system. Its purpose is to detect muons in the events.

1.2.2 The data acquisition system : DAQ

While running, the detector will register around 10 million collisions every second. Each one of those collisions produces 1 Mo of data. Nowadays, there is no technology able to deal that fast with such a big amount of data.
Nevertheless, only a very small portion of those 10 millions collisions are interesting for the physicists. Thus, the LHCb developed a system able to select potentially interesting events : the Trigger System. The LHCb trigger system operates on two levels.

The first level (called L0 for Level 0) is an electronic real-time selection made from data

coming from the VELO, the calorimeter and the muon-system. This level trigger works incredibly fast, making its decision in just four millionths of a second, and reduces the amount of events from 10 to a single million.

The remaining data are then sent to a set of 300 cards designed by CERN (called TELL1, see figure 2). We can say that this set of card is the entry point to the computing farm, to which we will send well formatted fragments of events through a complex IP network.

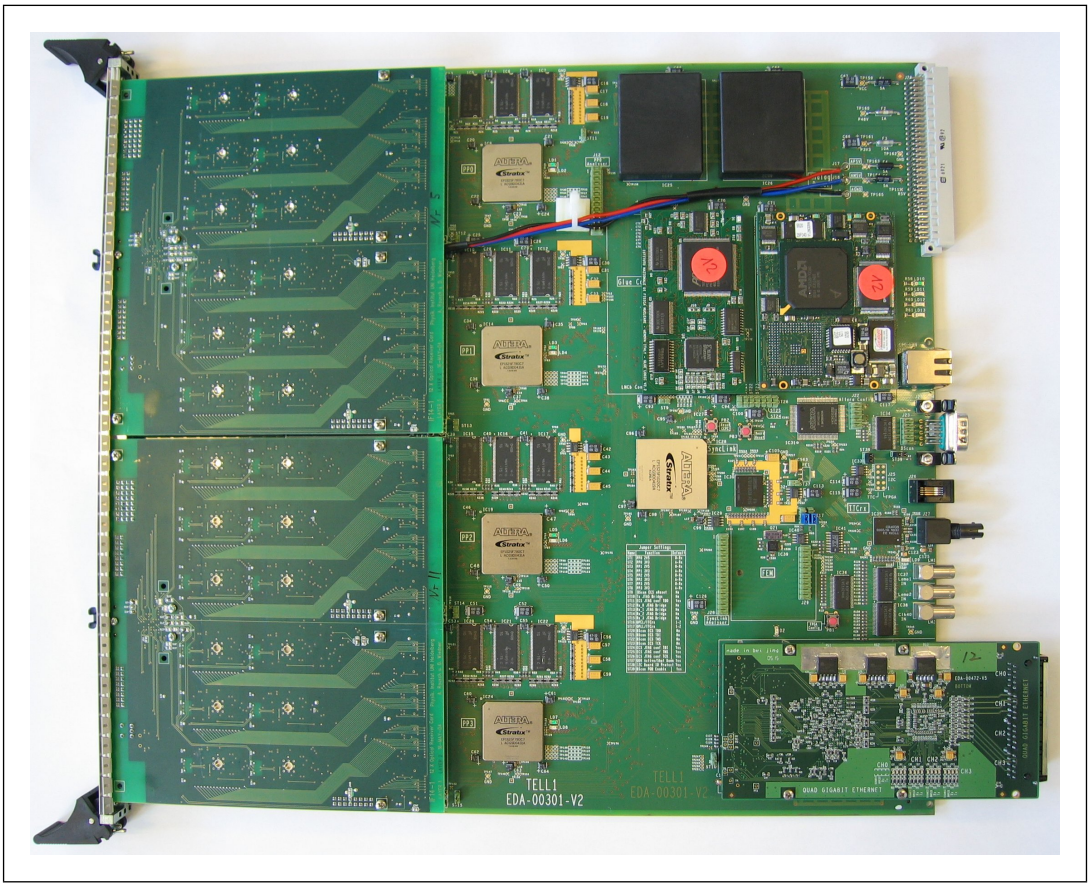

Figure 2: The TELL1 card

The computing farm is the HLT, which stands for High Level Trigger, the second level. This level is composed by an average of 2000 computing servers and 400 control servers. Its purposes are to reassemble the data coming from TELL1 in order to represent the whole event, but also to make a more accurate selection of the events considered as interesting. This level reduces the amount of events from 1 million to only 2 000, which is a more man-

ageable quantity.

Thus, in the LHCb experiment, we differentiate the LHCb Online team, which takes care of all the task processed during the acquisition of the data, and the LHCb Offline, which deals with the stored data.

1.3 Presentation of the project

1.3.1 Reasons for this project

I was affected to the LHCb Online team to realize my project. As we saw it previously, this team has to deal with an impressive amount of servers spreaded out over a large and complex network. This network is composed by those numerous servers, but also by a big number of layer 3 switches that interconnect the equipments.

Dealing with such a wide infrastructure requires several people to take care daily about it. Unfortunately, people may do mistakes. It may also happen that a network administrator don't have time to carefully configure the equipment : important is that it works. But a small misconfiguration on a single equipment can avoid the whole network to work properly.

Thus, a software providing a global overview of a network is very interesting tool for network administrator : if I only show you a screw of a complex machine, you have no way to know how the machine is working or what it is doing. But if I show you the whole machine with information on it, you can understand how it works, and you could even find where the problem comes from if the machine faces issues.

LHCb network is not a unique case, and even administrator of small network can find useful to have dynamic global overview of their network at anytime.

1.3.2 Specifications of the project

As I said it previously, such a tool can be interesting for many people in many different places. So even though the priority was to make it work on the LHCb network, a lower priority requirement was to make the software general enough to be used anywhere.
In spite of its small priority, we will see in the second part that this consideration leaded all my project.

The "customs" of CERN made me develop the software in Python. We will shortly discuss this choice in the second part.

Because a picture is worth a thousand words, drawing an interactive map of the network was one of the requirement. Indeed, there is no better global overview of a network than its map.

The software was also supposed to gather information about the equipment. Here is an exhaustive list of the features that we want to collect (some of the terms will be explained later in this report) :

- Type of equipment (switch, router, server)
- Name
- Amount of ports
- MAC address
- IP address
- GVRP capability
- ARP table
- Routing table
- Forwarding database
- LACP configuration
- VLAN configuration

We also expect to gather more precise information about each port of the equipments:

- Port number as defined in the equipment
- Speed
- MTU
- MAC address
- IP address
- Amount of packets received

- Amount of packets sent
- LACP status

The information is supposed to be stored in a database.
Finally, the software was asked to go through the network and collect information thanks to the SNMP and LLDP protocols, on which we will spend some time in the second part.

At the bottom of the requirement tasks, we find monitoring process. This is a secondary task for two reasons :

- some other tools already make this job really well
- the monitoring action we want at LHCb is a bit atypical : we don't want any change on the network. This means that no equipment should be added, moved or removed. Such a requirement does not make sense on most of the network because the software would trigger an alert every time a user turn off or turn on his computer.

1.3.3 Schedule

The schedule of the project is basically to spend one month on the application and database designs, algorithm research; one month of coding; and then make tests and improve the project.

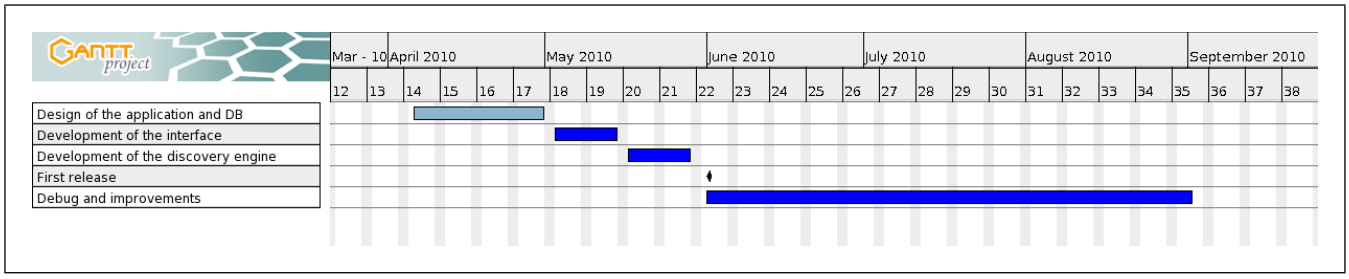

Figure 3: The expected schedule of my project

Now that we defined our needs, we can have a look at the technical points.

2 Technical aspects

As we saw it previously, making the software to work everywhere is a consideration that must be kept in mind during all the process. Thus, during all this technical analysis, we will

always take this requirement in account, and explain the consequences it has.

2.1 The technologies used

In this paragraph, I will present you the context in which I developed the software, meaning the different technologies and tools on which my software is based.

2.1.1 SNMP

SNMP stands for *Simple Network Management Protocol*. It is a UDP-based network protocol, defined by the *Internet Engineering Task Force (IETF)*, used mostly in network management system. SNMP is actually more than a simple protocol, because it also defines a set of standard like a database schema, and a set of data object, that we are going to describe now.

The global working of SNMP is quite simple (see figure 4) :

- we have a computer called *manager* supposed to monitor or manage a group of hosts (computers, servers, switches, printers ...). All the managed devices run a software named *agent*.
- the manager sends request (or order) to the agent of a managed device.
- the agent answers the manager (or execute the order)

So this protocol is exactly what we need to gather the information we want. We will now look how SNMP is actually working.

As you can expect, the questions we can ask to a managed device, and the way to do it is precisely defined. We could say that every managed device has a list of predefined questions to which he can answer. This list is actually a database called *MIB*, which stands for *Management information base*. This database uses a hierarchical namespace, whose notation is defined by the ASN.1 standard. Simply said, each question understandable by a device has a unique name (called *Object Identifiers (OID)*), and those name are sorted in a tree (see figure 5)

Thus, to precisely identify a question, the manager musts name it by concatenation of all the node names, from the root to the leaf. But a leaf can be very deep in the MIB tree. To avoid to write several line to identify the question, each node is defined by a name, but also a number, which makes the OID shorter to write.

The structure of the MIB tree is precisely defined by several RFC standards, but we can

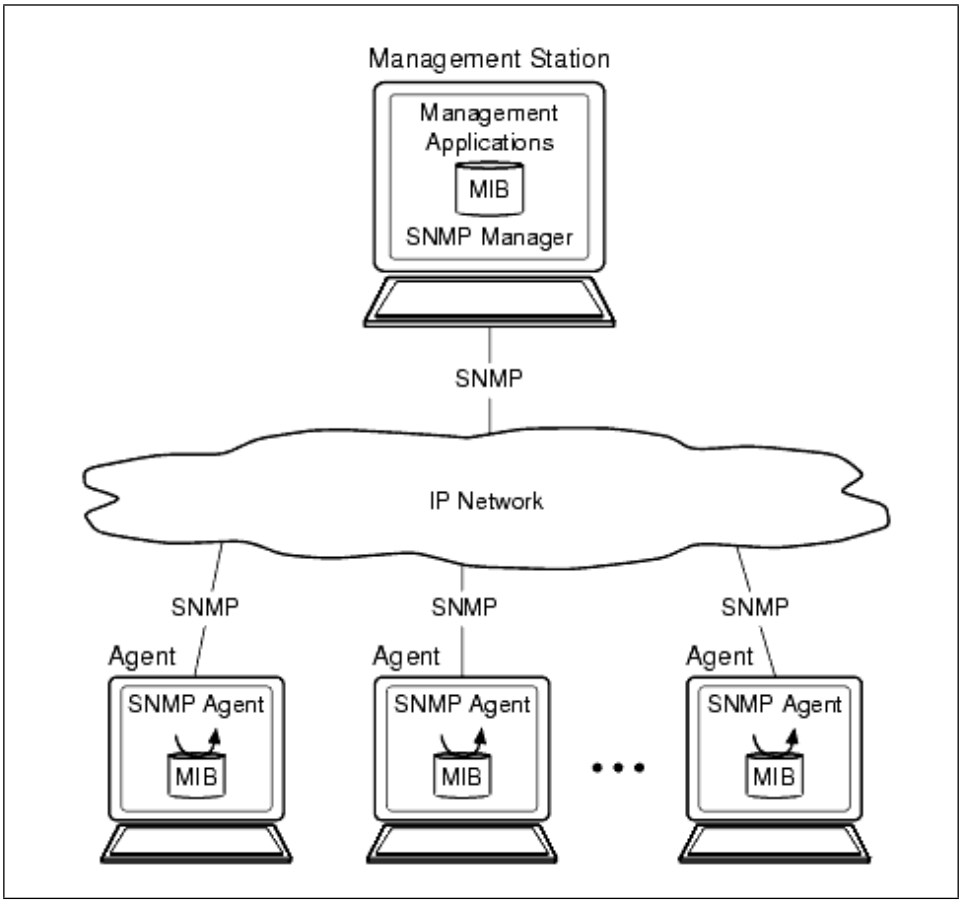

Figure 4: An SNMP description schema

make a crucial distinction between all the OID : either they are standard OID, or their are vendor specific. This means that either a question can be asked to any brand of device, or only to a specific vendor. We use to speak about *Standard MIB* and *Vendor MIB*. As you probably already guessed, for the needs of my project, I wanted to use only Standard MIB, but we will see in the third part of this report that it is not an easy task. (see 3.1.1)

SNMP has more options and details, but they are not accurate in this report [2].

[2]See the bibliography for the list of RFC about this subject

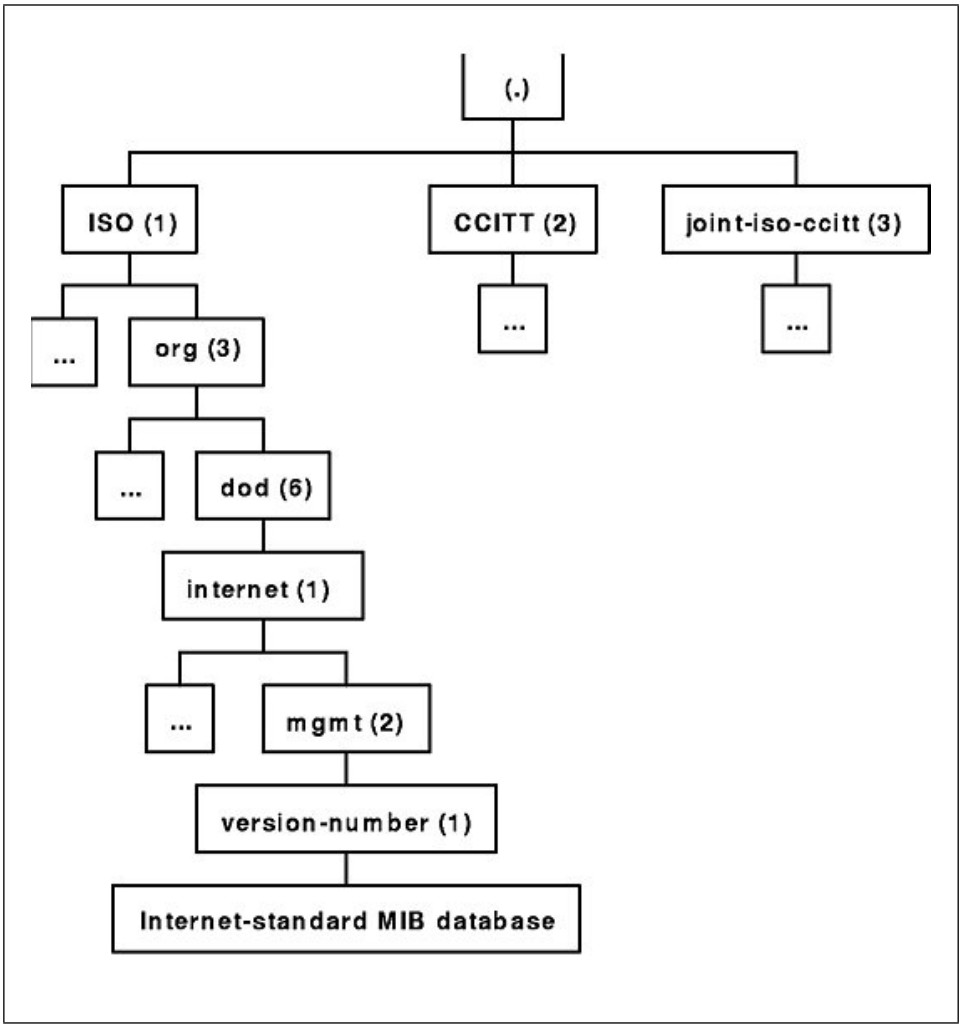

Figure 5: Example section of a MIB Tree

2.1.2 LLDP

The *Link Layer Discovery Protocol (LLDP)* (also called *IEEE 802.1AB*) is a vendor-neutral protocol, similar to proprietary protocol such as *Cisco Discovery Protocol*. Basically, LLDP is a protocol that is used by a device to advertises its direct neighborhood about its features : identity, capabilities, and interconnection on a IEEE 802 LAN network. This last information is definitely the most interesting for us : simply said, this allows a device to tell us to what other equipments it is directly linked.

LLDP is an automatic and transparent process for us : once you have it enabled on your equipments, the devices take themselves care about advertising their neighbors, or about updating their information about the remote devices.
But what is really interesting for us is that the information collected by a device about its neighborhood are stored in the MIB of the device : this means that we can get this information as any other by making SNMP query.

As a conclusion, we can say that we do not have to care neither know how LLDP works in our software. What is important, and even a crucial requirement, is that LLDP is enabled in every network devices.

2.1.3 Python

According to the CERN customs, which are not mandatory rules, I was asked to develop the software in Python. It is a high-level programming language that supports multiple programming paradigms. As python is a cross-platform languages, it is a good choice for the portability we want in our software. By choosing such a programming language, we ensure the portability from a network point of view, but from the execution platform point of view as well.
In my opinion, Python was maybe not the best choice because of its low speed : this is a big disadvantage when we have to run long algorithm, such as drawing a map. Nevertheless, I saw there a good occasion to learn Python, and the good coupling between Python and C++ finished to convince me.

2.1.4 Running configuration

In the section, I simply want to detail a little bit the configuration in which the software is currently running :

- Python version is 2.4.3
- The database used is MySQL. In a near future, we may skip to Oracle database
- The software is running on a virtual machine in the LHCb network, cut from Internet.
- The software runs on a *Scientific Linux CERN SLC release 5.4*, a Linux system developed at CERN
- LHCb network uses the switches of two vendors : Hewlett-Packard and Force 10.

A final test will maybe be launched in the Rome (Italia) hospital in the following weeks.

2.1.5 LACP, VLAN and GVRP

From all the information we want to gather about the equipment, those three terms deserve explanations, because they are less trivial network features.

LACP stands for *Link Aggregation Control Protocol*, and is an IEEE specification. It allows you to group several physical ports in a single virtual one. It is used mainly for redundancy reasons or performances increasing. An LACP can be either dynamically or statically configured, and the physical ports may act either in a "passive" or in an "active" mode [3].

VLAN stands for *Virtual LAN*. It is used to provide a virtual segmentation of a network, as you can see on figure 6. In order to make this segmentation, you assign in your equipment a port to a VLAN. We will see that it is not as easy as it seems to be later in this report, in a dedicated section. (see 2.4.3)

GVRP stands for *Generic VLAN Registration Protocol*, and is a protocol that somehow automatizes the VLAN configuration of a device, by exchanging information with its neighbors.

2.2 The database

In a first step, the software will discover the equipments of the network, and gather information about them. Those data need to be stored in a database in order to be used in a second step.

I send you back to the section 1.3.2 for an exhaustive list of the information we want.

You can see on the figure 7 the complete schema of the database, but I will summarize its structure.

- A network is composed by devices (Router, Switches, servers...), linked together trough their ports : this is represented by the tables *Equipment*, *Port* and *Link*
- Almost every equipment has a Routing table, an ARP table and a Forwarding table : so we have homonym tables
- As we saw it previously, configuration of VLAN consists in assigning a port to a defined group : this is the use of the tables *VLAN* and *VLANLink*

[3] Precise explanations about those modes is not interesting in this report. For more information, please read the IEEE Std 802.1AX-2008 document

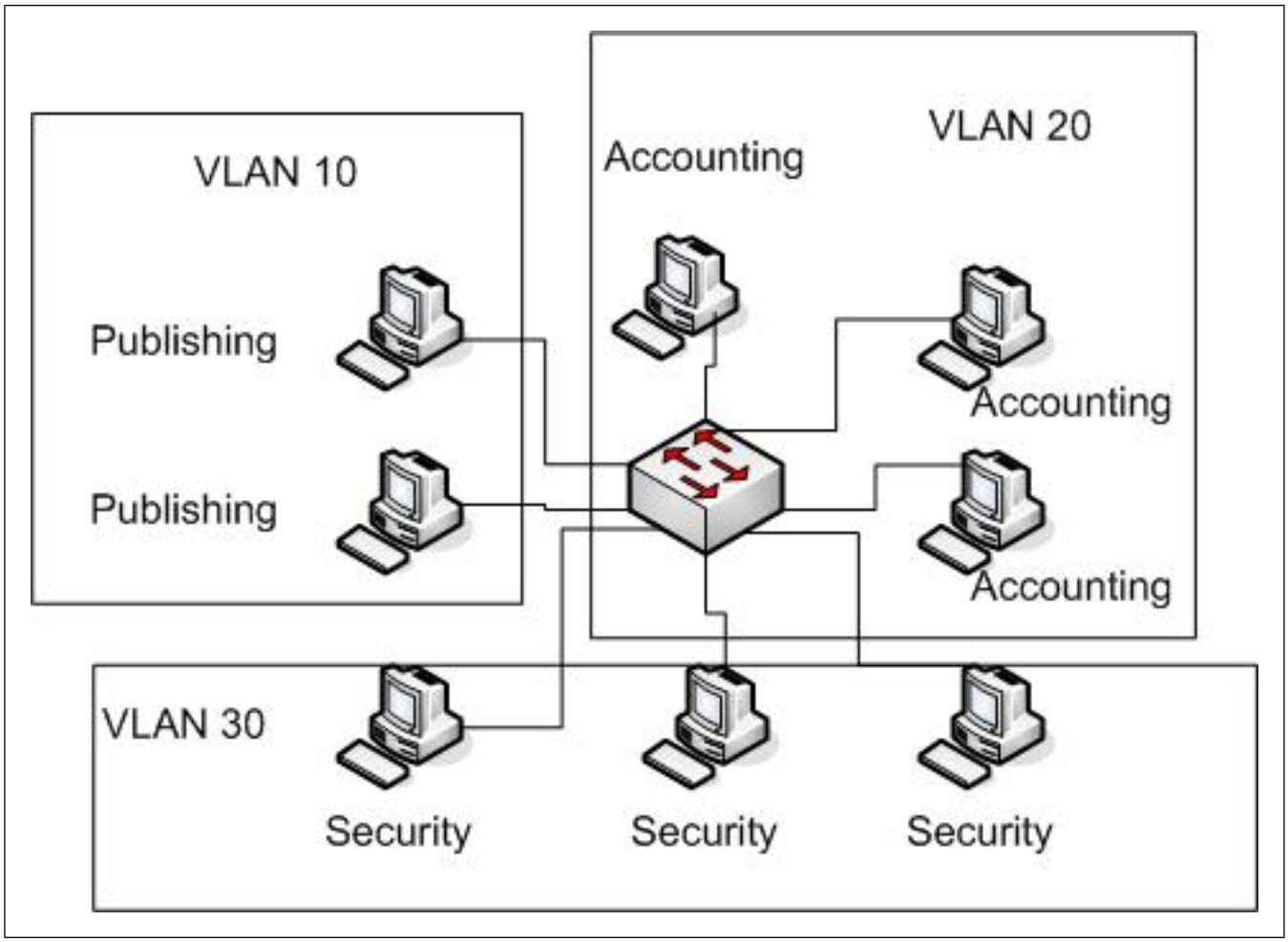

Figure 6: Illustration of the usage of a VLAN

- As for VLAN, LACP follows the same principle : assign a port to a group. That is why we have the *LACP*, and *LACPStatus* tables

For a while, I wanted to design a meta database. This means a database with a very variable structure. After discussing this proposition with Enrico BONACCORSI, we agreed that it was a lot of work for very little gain : the information we want about devices is precisely set, and is not subject to any changes, at least in LHCb. So we could imagine changing the database structure for a meta model if we want to make the information gathered variable.

You could also notice that some of the fields in the database are redundant. This is just

to simplify the request, and because those fields are not to be updated at any time.

As a conclusion about the database, we could say that it is well adapted for the information we want, but if we want to make the software even more general, by extending the information it gathers, we would have to redesign the database structure.

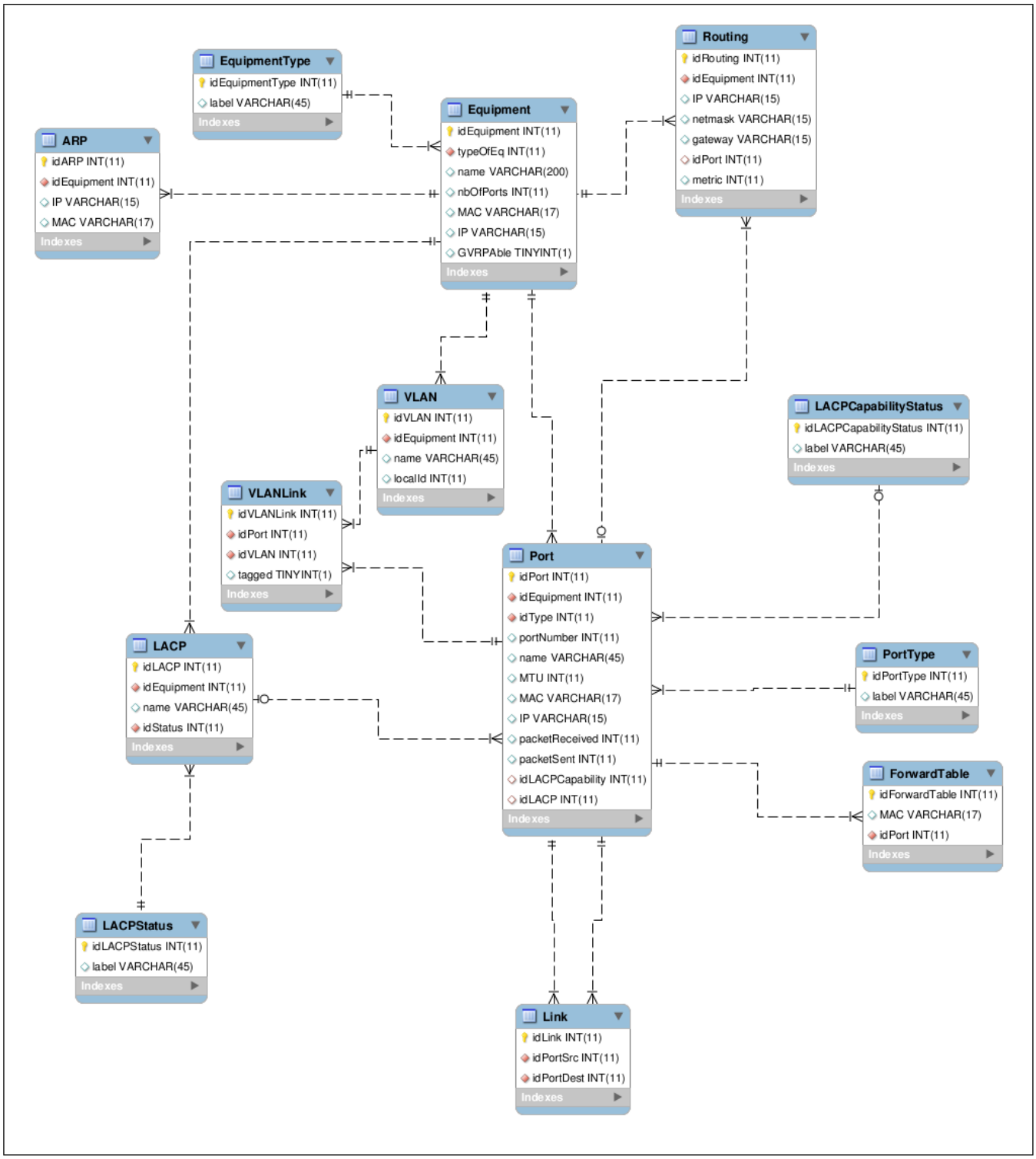

Figure 7: The database design

2.3 The application design

The design of the application is a fundamental point. If we really want to make the software general, easily modifiable and extendable, and easy to maintain, we have to be careful to the structure it has.

Thus, I spent almost one month on its design, the database design, and algorithms research (about which we will discuss later).

2.3.1 The architecture pattern MVC

The global software architecture I decided to use is a *Model-View-Controller (MVC)* pattern, whose schema can be found on figure 8. This pattern decomposes an application in three main parts :

- The model : it manages the information and data. The model is making all the computation, algorithms on them. When the model changes its state, it notifies the view(s).
- The view : the view is a human readable presentation of the model, and the way for a user to interact with it. Typically, it is the user interface.
- The controller : it accepts inputs from the user and instructs the model to perform actions.

The advantage of this architecture is that presentation and computation on the data are weakly coupled : one can change the user interface without changing the algorithms, and vice versa.

In the following description, I will mostly focus on the model part where all the difficulties of this project are concentrated.

2.3.2 The design pattern DAO

As we want to use a database, and keeping the generality of the software in mind, adding a layer between the database and the core of the application seems a good idea.
One part of this intermediate layer is a *Data Access Object (DAO)* pattern. Basically, this layer manages the mapping between the data as they are stored and the objects used by the application.

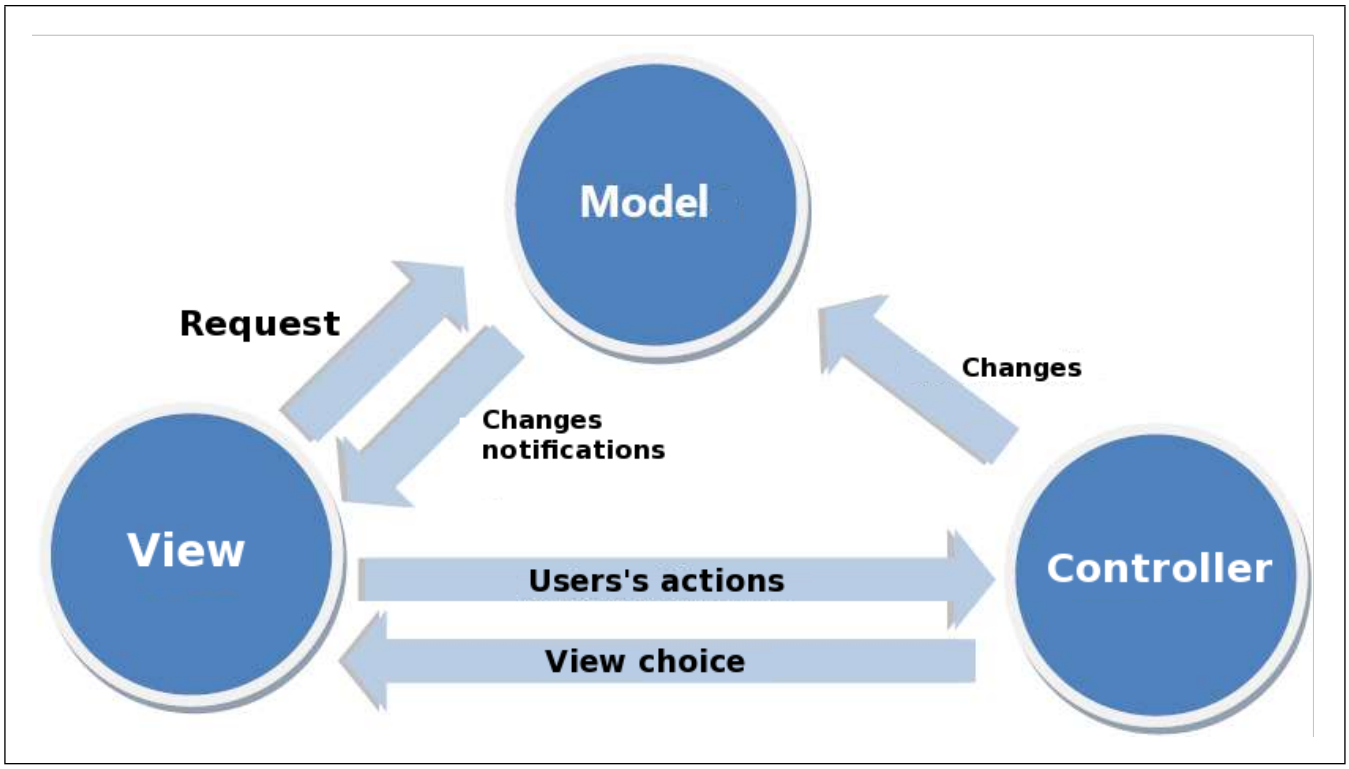

Figure 8: Schema of the MVC architecture pattern

In our specifications, we wanted to use database, so the DAO I implemented maps the objects of the application with the lines in the relational database. But if we decided to store the data in XML files for example, we would just need to change the DAO source code.

I said that the DAO is only part of the layer I introduced between database and application, because to access the database from Python, we need to use intermediate layer called *driver*, which is database dependent. Because the SQL syntax used in the DAO is pretty basic, the code would work with any *Database Management System (DBMS)* (like Oracle, MySQL...), but we need to use the specific driver. Thus, I added an abstract layer, so that the type of database is transparent for the DAO. The figure 9 represents the separation between the database and the model.

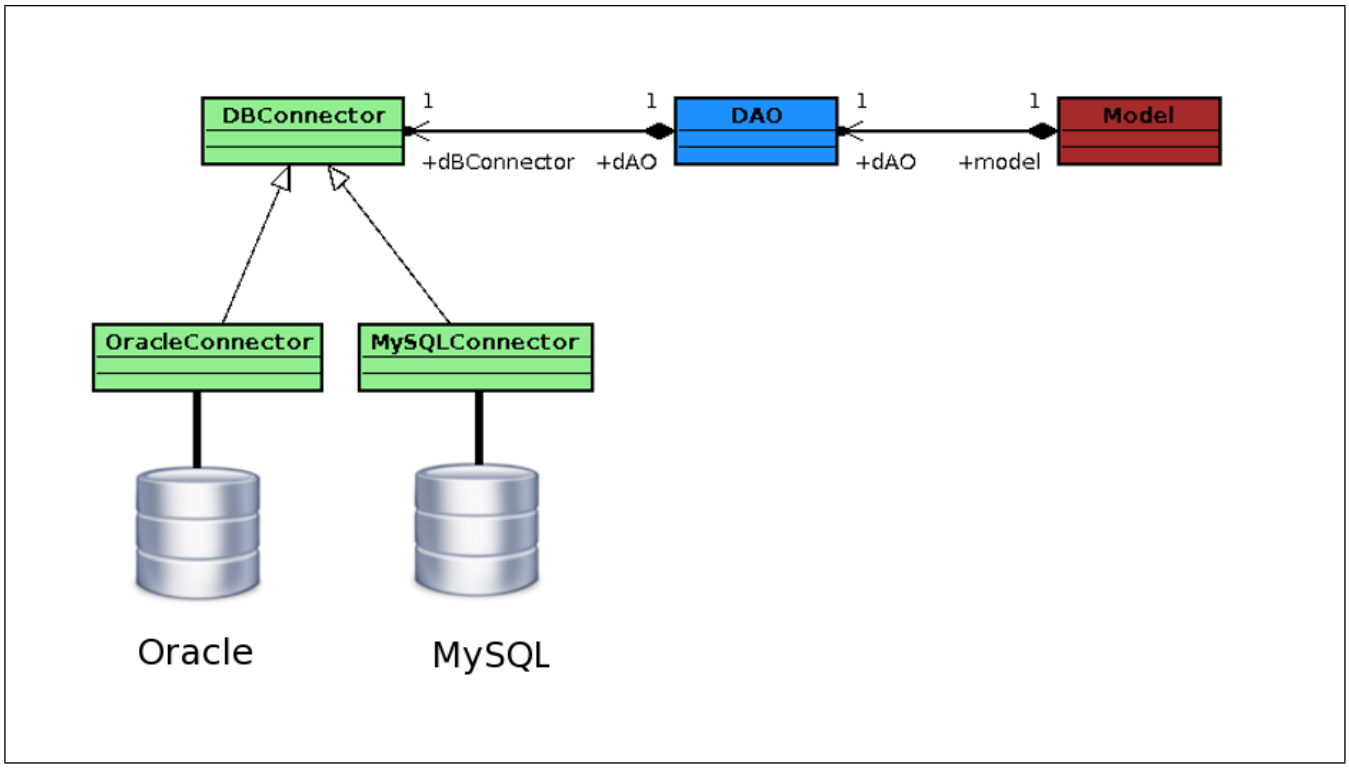

Figure 9: Schema of the DAO and driver layers

2.3.3 The data model

Inside the application, our data are represented by objects. A logical approach could be to have an object per entity we want to represent : one for equipment, one for Port, one for routing table and so forth. But if we take in consideration what we want to do with those data, we can make a step further.

When the data are in the database, we want to be able to :

- Display the configuration of an equipment and all what is linked to it, like its ports configuration, the routing table and so on.
- Display a map of the network, where the equipments and the link appear.

For the first task, we need to have in the object representing the data all the information we have about an equipment. For the second task, the consideration are mainly geometric :

where should I plot this equipment, between which equipment should I draw a line to symbolize a link... We don't need there to have any information about arp table, or port speed there, we would have uselessly big objects. And of course, to display the configuration of an equipment, we don't need its plot coordinates.

Thus, we come to the following data architecture : we have a family of object called *Data_Element* (see figure 10) used to hold the whole configuration, and a family called *Graph_Element* (see figure 11) used to hold the geometric data. And those family are indeed organized as our gut told us first : one class per entity we want to represent.

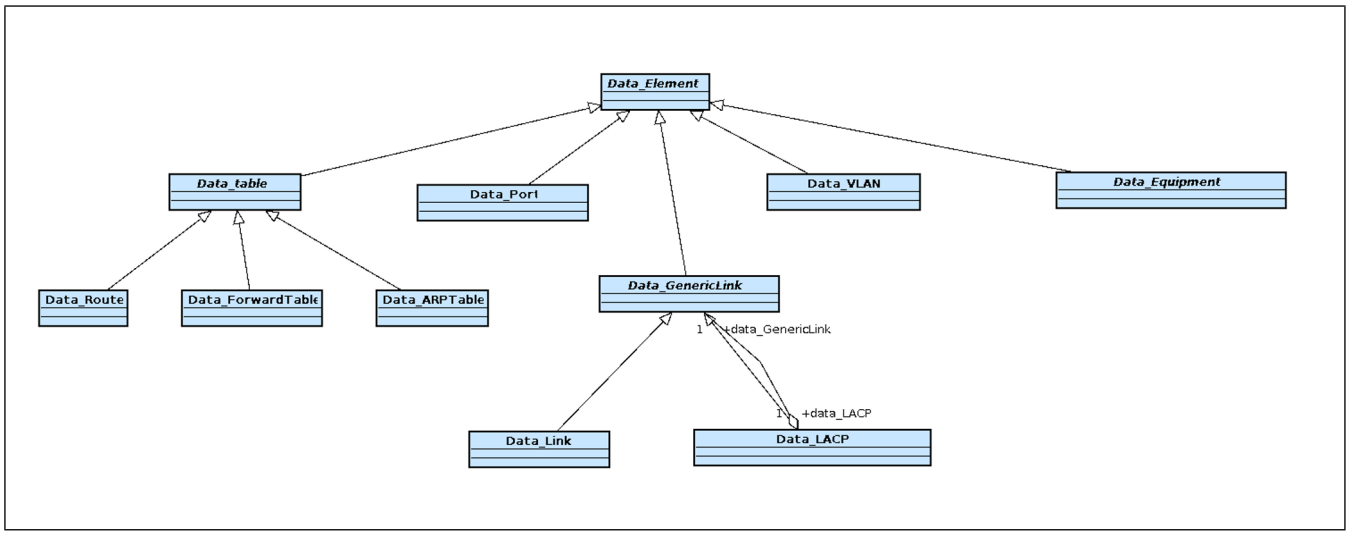

Figure 10: Schema of the family of object Data_Element

Nevertheless, there is third family of object that I had to implement : the *Request_Element* (see figure 12) that are used to gather the configuration of the equipment. Indeed, the presentation of the data that would be suitable for a human is not always the best to deal with inside the application.

The consequence of this division in families is that we have as much DAO as family. Even though classic DAO are mapping in two ways, from object to database and from database to object, the DAO I developed make a one way job. There are several reasons for this choice. Among them is the fact that user don't modify the data, we decouple the input and output of database, and it is more convenient to work in different piece of code when the approaches of the data are different. You can see a schematization of the DAO system of my project on the figure 13

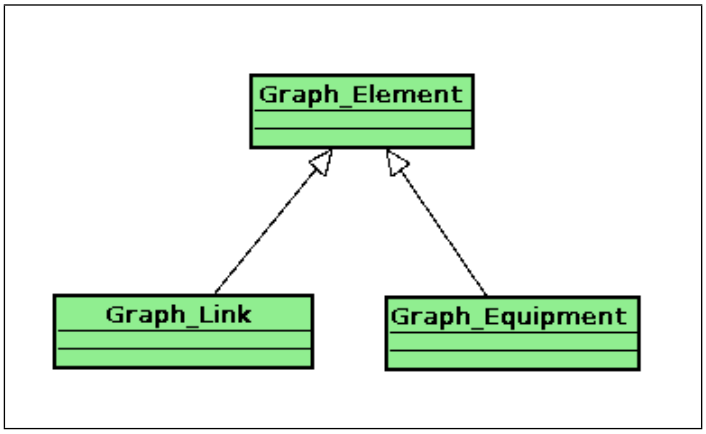

Figure 11: Schema of the family of object Graph_Element

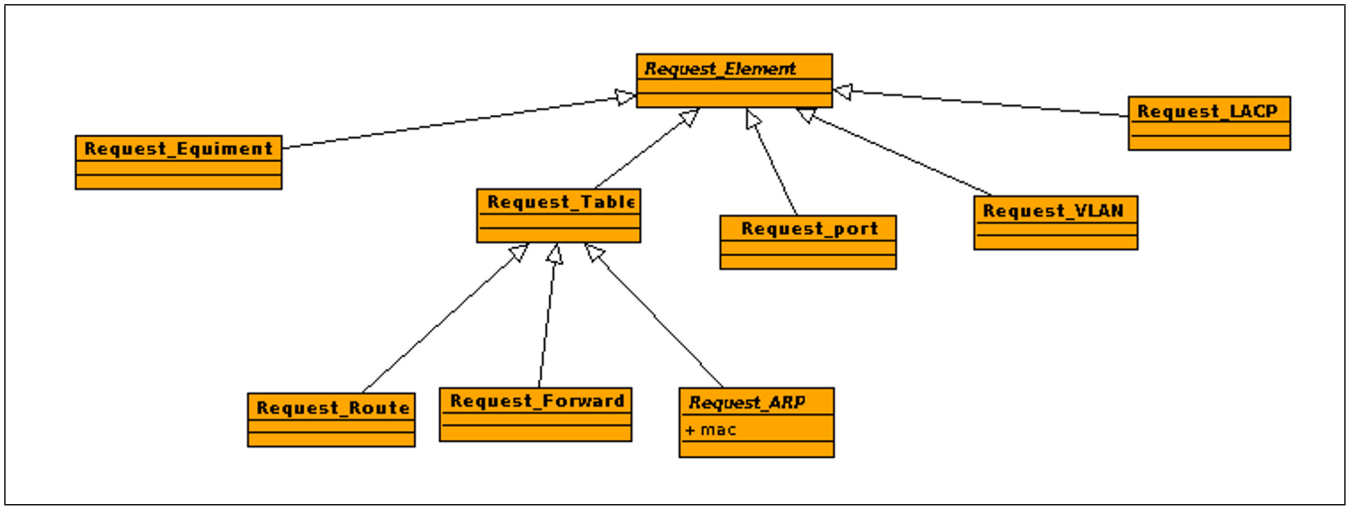

Figure 12: Schema of the family of object Request_Element

2.3.4 The request model

The structure of the part of the software supposed to gather the configuration of the equipment is the last sophisticated architecture. The specifications of the project set that the information must be gathered with SNMP request. But we can imagine that a company, with a very specific type of material, would like to use any vendor protocol to do the job. This is why I decided to implement a *factory pattern*.

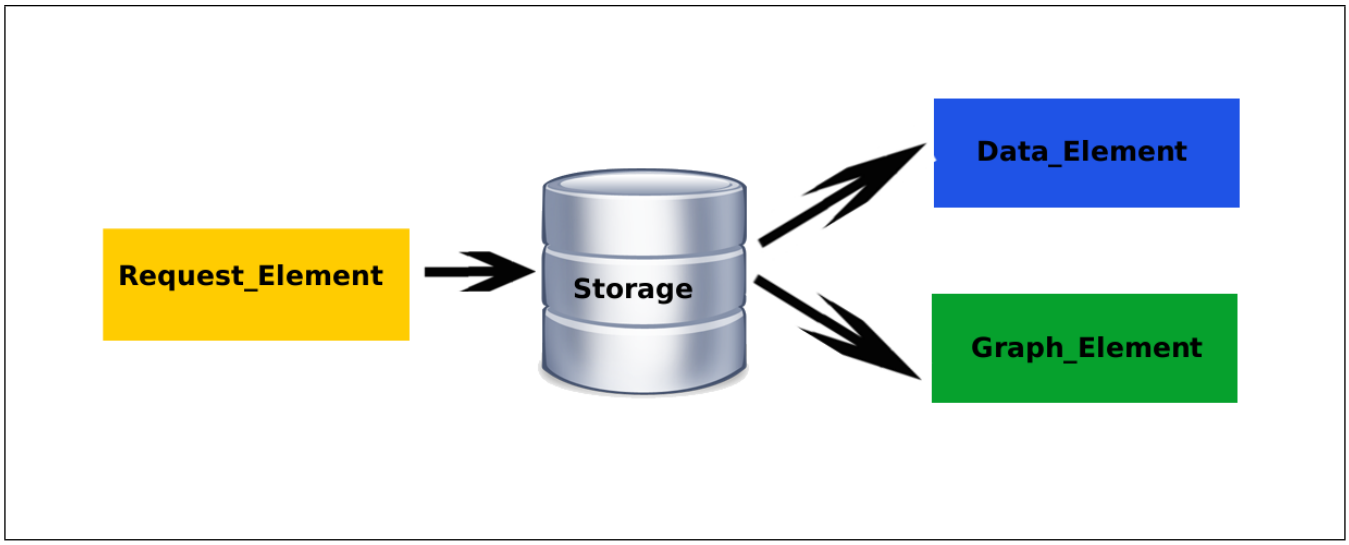

Figure 13: The DAO system

According to the famous *Gang of Four*, the purpose of the factory pattern (see figure 14) is to "define an interface for creating an object, but let the subclasses decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses."

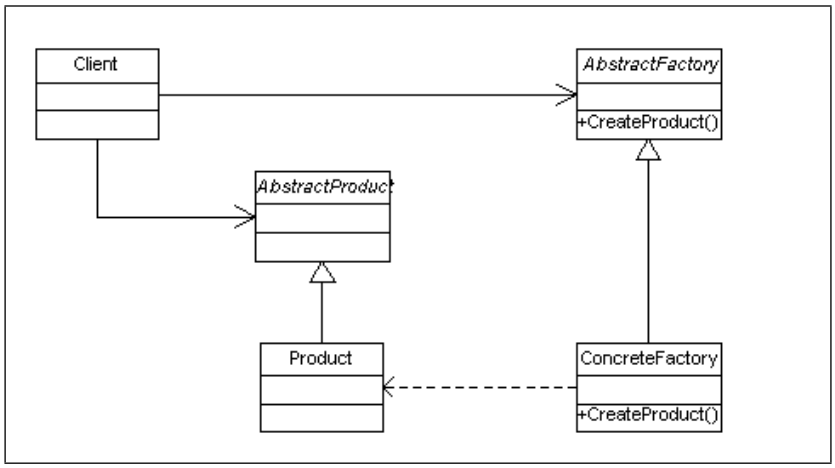

Figure 14: The factory method

We will spend more time on it later on this report, but the global idea to collect the data is

the following: we have a main platform, called *DiscoveryEngine*, that holds the algorithm to discover the network. To do so, it interrogates every equipment one by one, and crosscheck the information. To collect the data, the DiscoveryEngine creates one object called *Request* per equipment.
This Request object is like a direct interface between the DiscoveryEngine and the equipment, just as the remote control is an interface between you and your TV. Whatever TV or remote control you use, you always push exactly the same buttons, but you do not care about how the action is executed inside, only the results matter to you : to replace Jean-Pierre Pernaut by Laurence Ferrari.
This is exactly the same for the DiscoveryEngine : it is not aware whether the request is made with SNMP, or through files or whatever, it just expects answers to its questions, which are always the same. Thus, if a company wants to use vendor protocol, they would just need to create a new type of Request, but the algorithm would still work. The resulting design can be seen on figure 15

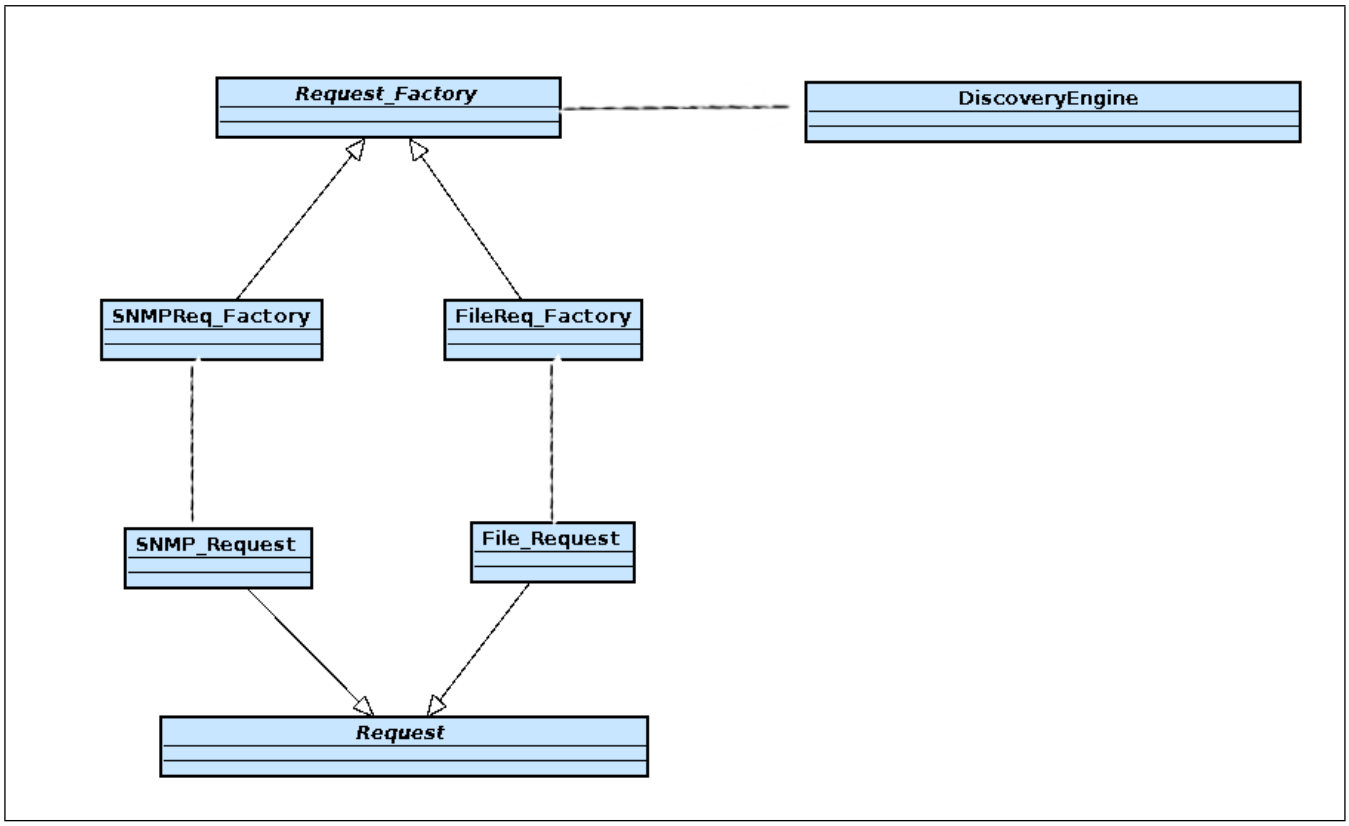

Figure 15: The factory method as used in the project

2.3.5 Miscellaneous

The software also includes configuration files, advanced logging system (with different log levels, maximum size of the file and so forth).

The architecture of the project is stuffed with diverse design patterns that makes the whole software a very adaptable application for the operating system that runs it, the database it uses, the way it collects the data, and even more.

All the key parts of the software can be change in an elegant way in the level of the code (such as the drawing algorithm, the log system...), but also at the user level. Indeed, the design of the application allows changes to the user such as the language of the software (so far only french and english translation), but also deeper changes in the configuration (such as the database configuration), or gimmick (such as drawing colors).

2.4 The Algorithms

Now that we have seen the design of the database and the application, let's have a closer look at the algorithms implemented in the software.

2.4.1 Force-based algorithms

Draw the map of a network is not as simple as it seems at the first look. Of course points and lines are good enough representation of equipments and cables, but the real difficulty is to place the elements. This problematic is actually a branch of graph theory called *graph drawing* or *graph layout* and this is the reason why this algorithm deserves a few lines in this report.

There are quite a lot of algorithms to process such a task, but the results is not always the same, and one would use certain graph layout for certain goal. I decided to use the *Force-based algorithm* to draw the map of the network. This choice is based on comparison of several pictures that show the result of different layout algorithms, and on the complexity of the algorithm. Indeed, my knowledges in graph theory are not sufficient to compare all those methods in detail.

The Force-based algorithm is a very clever method which assimilate a graph with a physical system. As you can see on the figure 16, a graph is composed by *nodes* and *edges*. With such a picture, it becomes obvious that we can compare nodes with equipment and edges with cables. The principle of the algorithm is to assign forces as if :

- edges were springs : we can use the Hooke's law to describe their behavior.
- nodes were electrically charged particles : Coulomb's law tells us how they interact.

Thus, nodes will repulse each others, and those which are linked with an edges are pulled together like a spring would do (see figure 17). The layout drawing is complete when the physical system is stable, that means that all the forces compensate themselves. Some examples of the draws obtained with Force-based algorithm can be seen on figure 18.

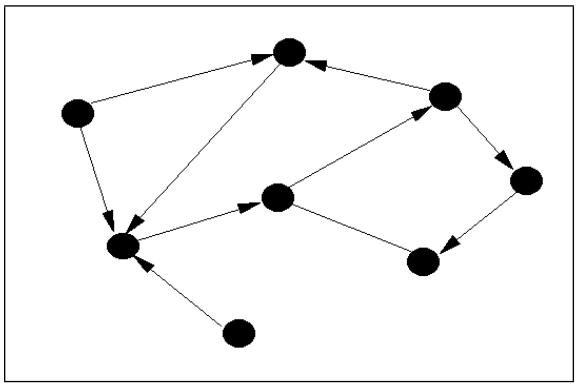

Figure 16: An example of a graph

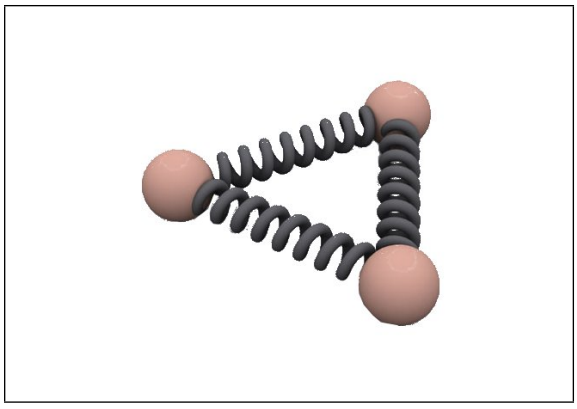

Figure 17: Comparison of a graph to a physical system

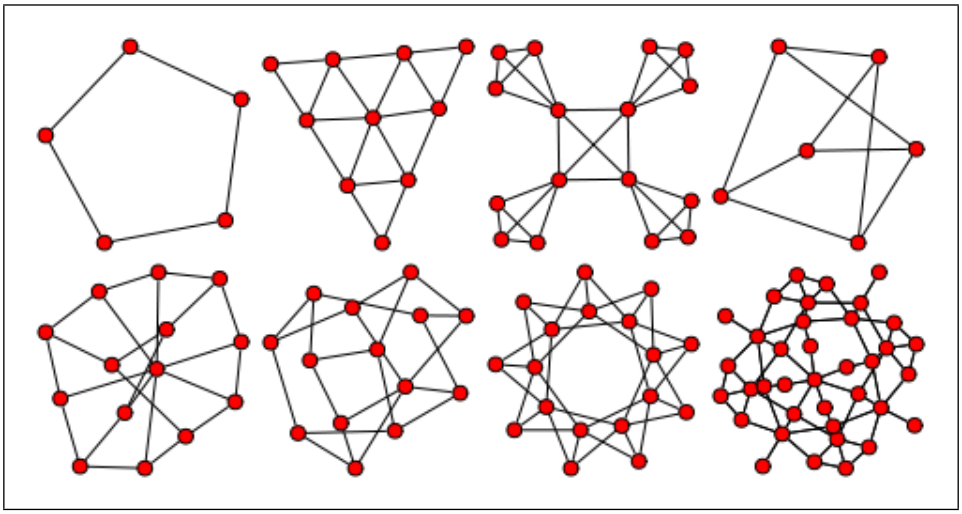

Figure 18: Examples of layout with Force-based algorithm

```
set up initial node velocities to (0,0);
set up initial node positions randomly ;
repeat
    totalKineticEnergy ← 0;
    foreach node do
        netForce ← (0,0);
        foreach otherNode do
            netForce ← netForce + CoulombRepulsion(node, otherNode)
        end
        foreach spring connected to node do
            netForce ← netForce + HookeAttraction(node, spring)
        end
        /* without damping it moves forever                              */
        node.velocity ← (node.velocity + timestep * netForce) * damping
        totalKineticEnergy ← totalKineticEnergy + node.mass * (node.velocity)^2
    end
until /* α is a very small number                                        */
until totalKineticEnergy ≤ α;
```

Algorithm 0.1: Force-based algorithm

The Force-based algorithm is known for producing good quality results, being intuitive, and very adaptable. Unfortunately, it has a high running time (3.1.4)

2.4.2 The network discovery

This is the biggest algorithm of the software, and one of the most interesting part of the project. Let's repeat our need once again : we want to discover every equipment (switches, routers, servers...) that belong to the same network, and gather information about them. Collecting the information is not what interests us here, because once we have found the equipment, we simply query it. The exciting part is "how do we find the equipments?".

First of all, remember that we must have LLDP and SNMP enable on all our equipments. SNMP allows us to query the equipments, and thanks to LLDP each equipment knows his neighbors. Thus, the solution seems trivial :

1. We choose the first equipment we want to interrogate
2. We gather its neighbors name
3. We interrogate every equipment of the neighborhood, collect their own neighbors, and so on

But it is not that easy for 2 reasons :

- We have to pay attention to infinite loop : let's say that I am querying the equipment A, and I learn that it has the equipment B as a neighbor. So I will want to interrogate B. But B tells me that it has A as neighbor, and I will want to interrogate A, and so on.
- We want more precise information than "equipment A is linked to equipment B" : we want to know through which of their ports they are linked. Even though LLDP protocol permits equipments to know the remote port to which they are connected, this information is hardly useful : first of all, the information is not always known, so it is not reliable; and then, the remote port will be designed with a name that correspond to the "remote equipment local port name", that is to say an alias name for the port set on the remote equipment. As alias name are not trustable, we need to work with port Id, that are not known except for the local equipment.

Nevertheless, there is an elegant solution that solves those two problems : let's call it the *link buffer*. The principle is the following : when we gather the neighbors's names, we don't create any object for the remote equipment, neither do we create object for the link between the two equipments. The solution is to create temporary data, that correspond to "half link",

and to create the link object when we have the two half of a link. I will illustrate this principle on the following example.

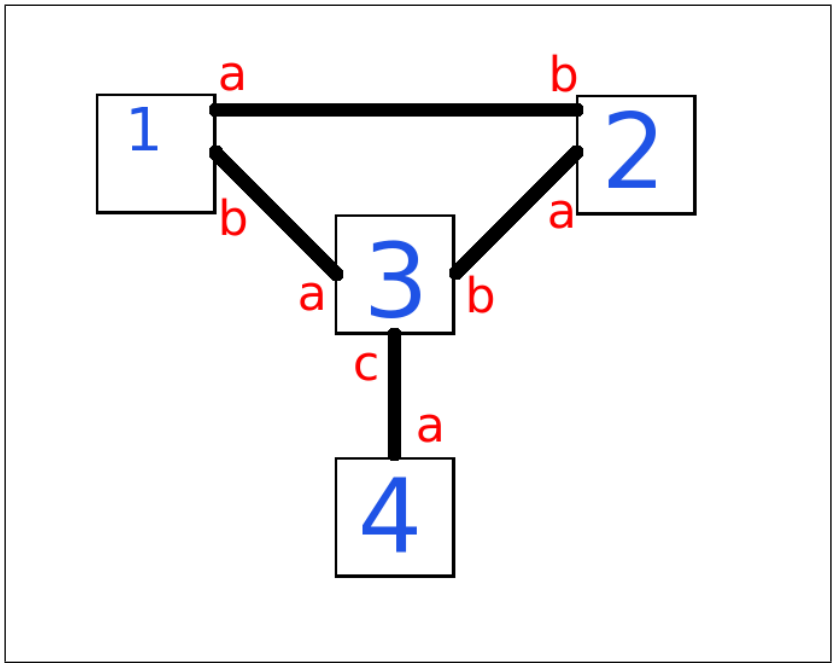

Figure 19: An example of a neighbor discovery

The figure 19 shows you a very basic network : we have 4 equipments (1, 2, 3 and 4) linked together as described on the picture (Equipment 1-port a is linked to the equipment 2 port b, ...). Let's assume that we start our algorithm on the equipment 3.
First we gather the neighbors of 3, and we get the following half links [4] :

Equipment 1	port 1	Equipment 2	Port 2
3	a	1	?
3	b	2	?
3	c	4	?

None of the line is totally filled in, so we don't create any link object. We now take the first equipment on the link buffer : it is the equipment 1. This equipment will inform us that

[4] It is important to know that the 2nd and the 4th columns are the unique id of the port as they are stored in the database. It does not appear on the schema not to make it too complicated

it is connected to equipment 2 through port a, and to equipment 3 through port b, so we add this information in the link buffer, and we get the following table :

Equipment 1	port 1	Equipment 2	Port 2
3	a	1	b
3	b	2	?
3	c	4	?
1	a	2	?

As you can see, the first line is now complete, so we can create the corresponding object, and delete the line from the link buffer. Thus, I will not come back on the equipment 3. The next steps are the following :

- We interrogate equipment 2 :

Equipment 1	port 1	Equipment 2	Port 2
3	b	2	a
3	c	4	?
1	a	2	b

  So we can add two more link in the database and erase them from the buffer

- We interrogate equipment 4 :

Equipment 1	port 1	Equipment 2	Port 2
3	c	4	a

  So we can add one more link in the database and then the discovery is over.

You may have noticed that the solution of the link buffer also prevents from more complex loop than direct one (such as 1-3-2). We will see in the section 3.1.3 that this solution has other advantages.

If we implement this algorithm using SNMP, and try it, you will be nicely surprised to detect all the switches and routers of your network, but probably angry of not seeing any workstation or server. And the reason is easy to find : their is hardly any workstation which has SNMP nor LLDP enabled. The direct consequence is that the switch (or router) and the computer interconnected do not know that they are neighbors. The indirect consequence is that I have to find a way to discover those equipments that we will from now on call *end nodes*.

Guoming LIU just graduated his thesis at CERN, in the LHCb Online department, about network discovery and he had a solution to offer me. To understand it, we need to remember two things :

- The ARP table (ARP stands for *Address Resolution Protocol*) is basically a table that gives, for a given equipment, the correspondence it knows between the IP address (layer 3 of OSI model) of a remote equipment and the MAC address (layer 2) of this remote equipment. Because of configuration such as default gateway, the line present in an ARP table may not be the real MAC - IP association of the remote equipment.

- The forwarding database is a table used to store the MAC addresses that have been learned and which ports that MAC address was learned on. So we have a MAC-Port association.

The idea of Guoming LIU is to crosscheck this tables to have a Port-IP association, paying attention not to take in account the *uplink port*, which are the ports that link switches and routers together. To know that we are dealing with an uplink port, we check if this port appears in the list of port where we have a neighbors known by LLDP (as we assume that all switches and routers have LLDP enabled). The algorithm 0.2 is the algorithm to discover the end nodes, differing a little bit from the original form of Guoming LIU. It is made mention of logical-port, we will come back on it in section 3.1.2.

Thus, we can find all the switches and routers thanks to LLDP, and we can find all the end nodes by crossing the ARP table and the Forwarding database of each equipment : the network discovery algorithm is now complete.

2.4.3 The VLAN algorithm

As I explained you previously, a VLAN is a virtual segmentation of a network. To make the segmentation, we configure the switch to assign one port to a VLAN, also defined in the switch. A VLAN is uniquely identified with an ID. The difficulty comes from the fact that this ID, as we will see it, may not be the same across all the network.
In addition to the port to VLAN assignment, we must set each port either as 'tagged' or 'untagged' :

- tagged : this means that, when a packet is sent through this port, the switch "stamps" the packet with the id of the VLAN. When a packet arrives through this port, the switch expects a packed "stamped" with the id of the VLAN; if it is not, the switch simply discards the packet. A port can be tagged with several ids of VLAN. The port at the other end of the link must be tagged the same way if several VLANs are assigned to this port, or untagged with the same id if there is a single VLAN defined.

- untagged : this means that, when a packet arrives through this port, the switch expects an "unstamped" packet, that will be from now considered as belonging to the VLAN whose id is set on this port. We could say that anything plugged behind an untagged port is considered as belonging to the same VLAN, whose id is the one set on the

```
Data: globalARPTable
Topology discovery for switch S ;
UL ← uplink ports ;
ARP-Table ← arp-table;
FDB ← forwarding table;
foreach f in FDB do
    if f.port not in UL then
        if f.port is logical-port and physical-port in UL then
            break;
        end
    end
    if f.mac in ARP − Table then
        build the entry {f.port, IP(hostname)} ;
        Remove the entry from the ARP-Table;
    end
    else
        if f.mac in globalARPTable then
            build the entry {f.port, IP(hostname)} ;
        end
    end
end
foreach arp in ARP − Table do
    if arp.mac not in neighbors then
        Append arp to globalARPTable ;
    end
end
```

Algorithm 0.2: End-nodes discovery algorithm

untagged port. A port can be untagged with only a single id. The port at the other node of the link can be either tagged with the same id, either untagged with any id.

This tagging system is very useful for a network administrator hurrying up : he could add several machines to a VLAN, just by plugging a new unconfigured switch in an untagged port of a switch where the expected VLAN is set. But it then becomes difficult to have a global overview of the VLAN configuration over the network, because even though a switch is not configured at all, it can belong to a VLAN.

To makes things even more complicated : two switches linked together through untagged ports with id of VLAN e.g. 2 may both have a VLAN configured with id e.g. 3. Those two VLANs with id 3 will be totally independent. Thus, we arrive to situation such as on the figure 20. At a first look, according to the id of the vlans on the different ports, we would say that there is 3 VLANs grouped as following :

- A is alone
- B is alone
- C, D, E and F are together

But at the light of the previous explanations about tagging system, we arrive to the conclusion that there is indeed 3 VLANs, but order as following:

- A, B
- C, D
- E, F

Actually, at this point in time, I am still working on the algorithm that would permit me to have global overview of the VLAN configuration. The solution I have still suffers from bugs, whose origin is difficult to find. Indeed, the mistake may either come from the algorithm itself, or from wrong data that I collected (see 3.1.1)

3 Problems, solutions, and results

3.1 Problems and solutions

3.1.1 The Force 10 case

I met several problems that badly changed my schedule (see 1.3.3) : instead of the beginning of June deadline scheduled, I produced the first version by the end of July. Almost all

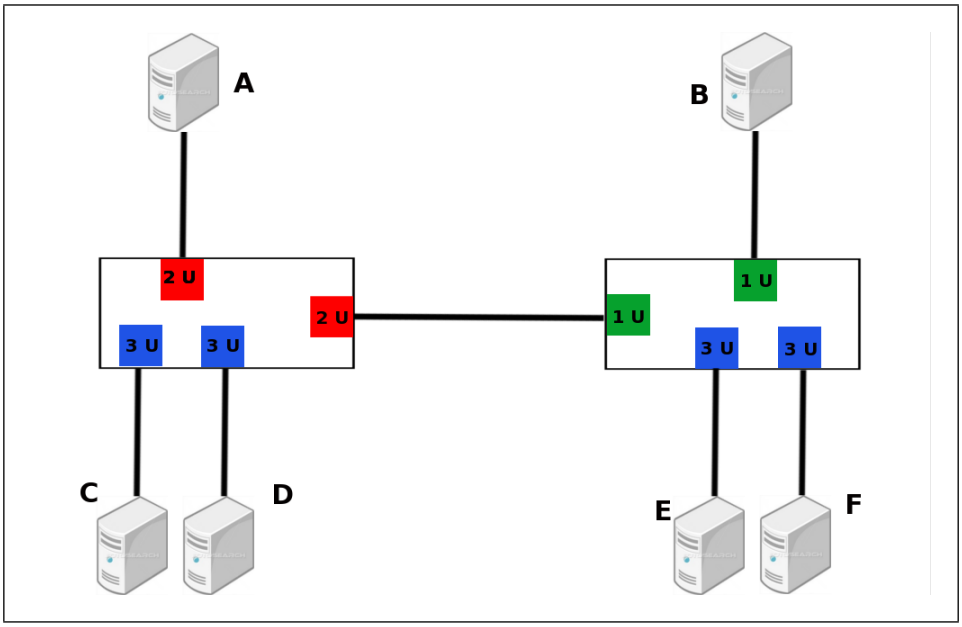

Figure 20: Tricky situation with VLAN

the issues I faced had a single origin, namely "Force 10 switches". I told you that LHCb network is based on plenty of Hewlett-Packard switches, and on a very few Force 10 switches.

For quite a long time, the discovery engine did not work properly, and I could not understand why :

- Infinite loops
- Different equipments discovered according to the start point
- Crash when dealing with some data
- Plenty of other issues

I decided to develop a specific and very precise log system to find out the reasons of all those weird events. Each run produced more than 100 000 lines of log, that had to be analyzed. I first had to deal with those log files manually, but once I understood the output scheme produced by the errors, I could create scripts that extract useful data. I could finally determine that all the problems come from the Force 10 switches :

- They do not answer to some standard SNMP requests

- They answer to some standard requests with a non-standard format of answer
- They answer to some standard requests something wrong
- There are few bugs in the firmware

Finding the reasons of those weird behaviors was long and difficult, but finding the solutions for this is at least as much time consuming.
It would have been very easy to make the test for every function if we are querying an HP or a F10 switch, but this would definitely not have been designed for generate purposes. Remember that the goal was to use only SNMP request of the standard MIB.

Another approach would have been to use the factory pattern I implemented for the request, and create a totally different Request depending on the equipment I interrogate. This solution would have been more general, but is very unpleasant, because then I would need to implement every single function once per brand of switch. So unless by using some tricks that would make the global design silly, this is not good solution neither.

In front of the very annoying bugs and mistakes of the F10 switches, I had to resign myself to use something else than standard MIBs.The difficulty was to include those special methods in an environment that is meant for being general. I finally found out a very elegant way, in addition to be very powerful.

Using the power of Python's introspection, I spent some times developing a "plugin system". Thus, I can write my whole application with very generic code, and when I meet a special case, I simply have to write my special function in a new file, and store this file in the appropriate directory. What is really interesting is that this plugin system is smart enough to deal with the plugins without any intervention of the user, and any modification in the original code. Even better : the way it have been developed allows people to create plugin for functions used when making request on equipment, but this also works for every single function or method of the software.

This system permitted me to use vendor mibs or ssh connection to retrieve the correct information from Force 10 switches, without making my main code dirty.

3.1.2 The virtual interfaces

A *Virtual network interface*, also called *Logical network interface* is a port that does not exist physically, but is used logically. For example, switches often create virtual interface for a

VLAN or an LACP : logically, the switch deals with it just as a normal interface, but physically, the treatments happen on the physical ports linked to that VLAN or LACP.
And this can be tricky : when gathering the interfaces list of an equipment, I care only about the physical one. But then, when I analyze the ARP table or the forwarding table, some lines may refer to a virtual interface, and as only the physical ones are meant to be added in the database, I do not know what to do with those special lines. You can imagine that things become complicated when we have an LACP that belongs to a VLAN.

The solution to solve this problem is quite easy : gather all the virtual interfaces, and "unroll" them until we have only physical ports. It is a little bit tricky because one can have infinite loop, with virtual interface including itself, so we must put an arbitrary maximum depth of unrolling. Even though the solution is obvious, the main difficulty was to find the appropriate place for this algorithm : after we have all the needed information, but before we need them.

3.1.3 The unreachable hosts

This problem, although very simple once we have the solution, disturbed me for a while. Some equipments where discovered, but I could not interrogate them. Finally, I found out that the hosts were unreachable because of the routing configurations. This problem cannot be solved at the software level, one must launch the process on a computer able to reach every other equipments.

Nevertheless, the consequences on the algorithm of an unreachable equipment is an infinite loop. Indeed, this equipment always appears in the list of equipments to interrogate. A solution would be to simply erase it from this 'todo list', but this would definitely not be elegant. Such solution would have as consequences to loose information : we know that there is an equipment, unreachable, and we know at least one of its link. It would be a shame to throw out such important data.

The link buffer (see 2.4.2) offers a very nice solution : the way it is done offers the possibility to create fake equipment, fake ports, and fake links in the database, to store this information we do not want to loose. Actually, this solution is available, whether the remote equipment is unreachable or simply does not speak SNMP.

3.1.4 The network map

The Force-Based algorithm produces very nice looking graph, but its biggest defect is its execution time when we have a lot of nodes and edges. The version I implemented was

working perfectly for graphs until a very few hundreds of elements, but it never came up with the whole LHCb network, provoking application crash. The algorithm itself is not the only responsible of this problem : Python is definitely too slow for such application.

Luckily, Python can interact very well with C or C++ code, and it occurs than the C++ library *Boost* implements the Force-Based algorithm. Actually, the exact algorithm is called *Kamada-Kaway*, which is a Force-Based algorithm with optimized physic laws to get the nicest graph possible.

However, even if the algorithm is now processed in C++, the drawing time is about 15 minutes. The Boost library offers few other graph layout algorithms, that I am, at this point in time, considering to try.

3.1.5 The RPM packaging

I was asked only a few days ago to package my software in an RPM package. An RPM is a file in charge to distribute a software in the most automatic way possible for the user. Nevertheless, Python is not really adapted for RPM distribution, and has its own distribution ways, which are less convenient (in particular, there is no efficient uninstallation system). In addition, the version of Scientific Linux used at CERN suffers from a bug, that appears almost randomly, and makes the packaging of Python sources a nightmare.
However, the packaging had to be done in RPM because it was meant to be used in *Quattor*, a system developed at CERN to manage very large clusters.

I finally decided to mix both solutions : package my software with Python tools, and package the result, with all the dependencies in an RPM package. This solution is very nice because it offers the uninstallation tool linked to RPM, and it comes with all the dependencies, which are installed only if they are not already present on the system (this part is managed by a script I added).

3.2 Results

Although the software is not totally finished now, we can already have a good idea of what it will look.

3.2.1 Time line

As you can see on figure 21, the real time line I followed was quite different than the one I expected in section 1.3.3. The first release came much later, but I could debug and improve

my software all along the development. The reason of those schedule modifications is the Force10, that we discussed in section 3.1.1.

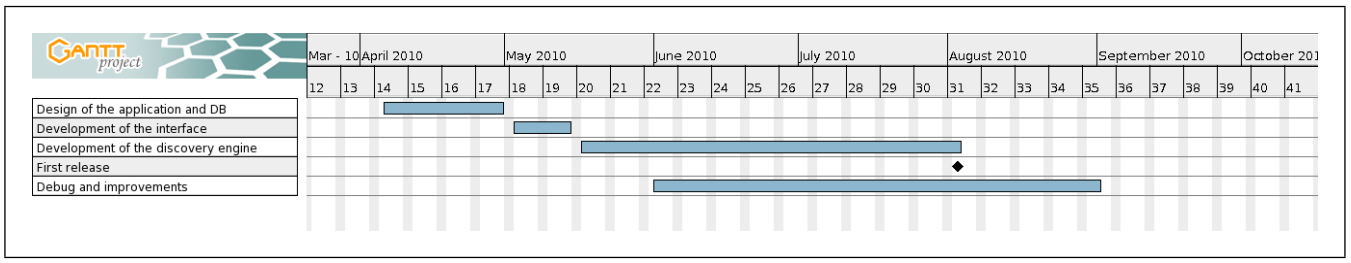

Figure 21: The real schedule of my project

3.2.2 Performances

The full discovery of the LHCb network takes about half an hour. A profiling test of the application shows that most of the time is spent doing the request and waiting for the response, which we could easily guess. We could save a very few minutes by decreasing the *timeout*, the amount of time after which we consider that the equipment does not answer. In addition to this, when we get an empty response, we interrogate the equipment once again to be sure that the response is really empty.

I am still working on the update algorithm, so I cannot give definitive conclusions about it. Nevertheless, even thought it is based on several threads (which we cannot do for discovery), its running times appears to be *at least* half an hour. The best time is reached when we use 4 threads. I did not have time yet to make a profiling precise enough to find out the real reason of this execution time.
However, I have serious suspicions that the reason is that we run the software on a virtual machine that emulates dual core processor.
In any case, as an entry in the ARP table or the Forwarding database lasts by default 2 minutes, the database will never be real time updated (except if we have one thread per equipment that keep running all the time, which brings other problems).

Some request in the database may also slow down the process, especially the update one. The database is already optimized with index on the table. If I have enough time before the end of my internship, I will do a benchmark to compare the performance between a MySQL database, and an Oracle one.

3.2.3 The interface

the software actually offers two interfaces : one in command line, and a graphical one developed with wxPython (visible on figure 22).

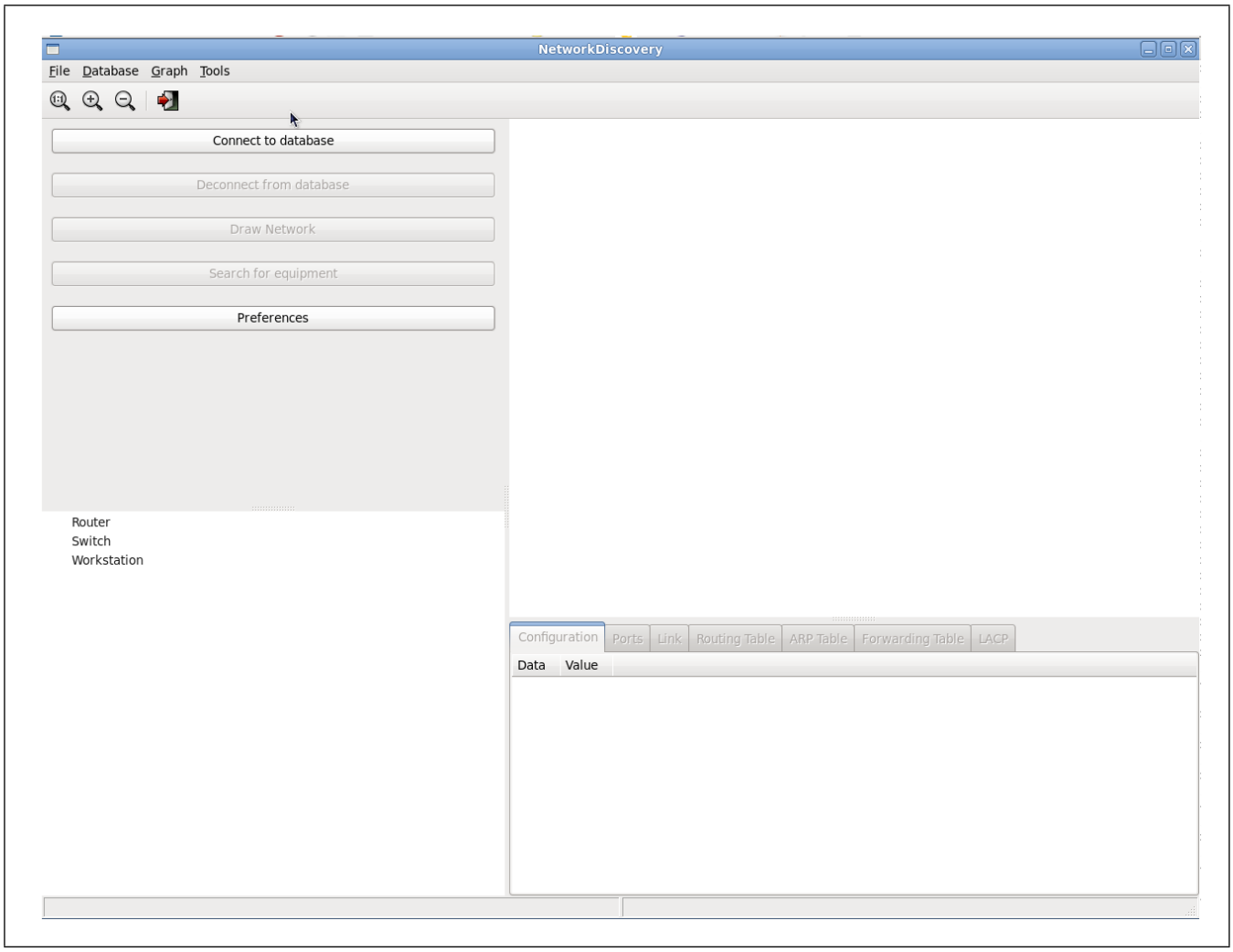

Figure 22: The graphical interface

The graphical interface offers a properties window, visible on figure 23, in which you can change log options, database configuration, the language of the interface, and some display properties (mainly color) of the graph.

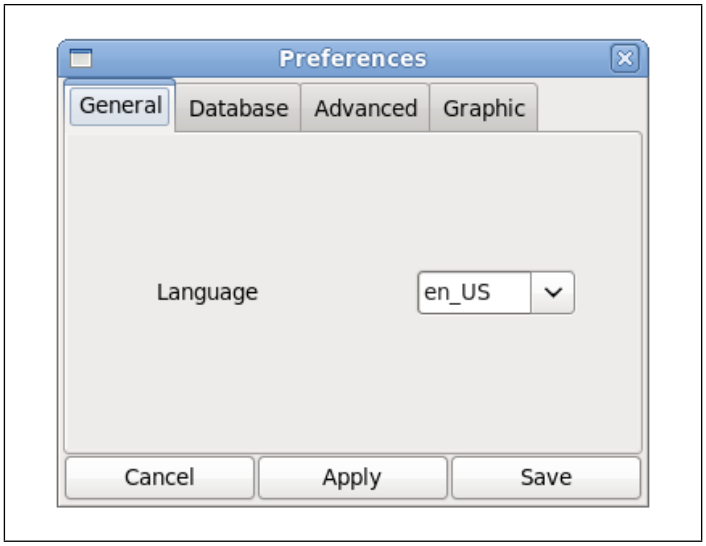

Figure 23: The properties window

The configuration of the selected equipment appears in the bottom of the Interface. As it is visible on figure 24, a tab system permits you to switch between the different parts of the equipment configuration : general, port, link, LACP, VLAN ...

The graph is displayed on the top right of the interface. As we discussed it in the section 3.1.4, the graph display still suffers from some issues, but you can see on the figure 25 the result of the algorithm with a smaller amount of nodes and edges. The map is interactive : we can zoom in and out, move the map, and information may be displayed (like equipment name on top of the corresponding point). Originally, the equipment were represented by icons. I replaced them with dots because of the computation time. When I will try to change the algorithm, I may display icons again.

Of course, the tree, the equipment configuration panel and the graph are linked together : an interaction with one may affect the others. For example, if you select an equipment on the map, it is highlighted on the tree, and its configuration is displayed.

A search tool is available. It allows you to search an equipment with name, mac, ip and type (switch, router, workstation). Of course, each of these fields can be combined, and we

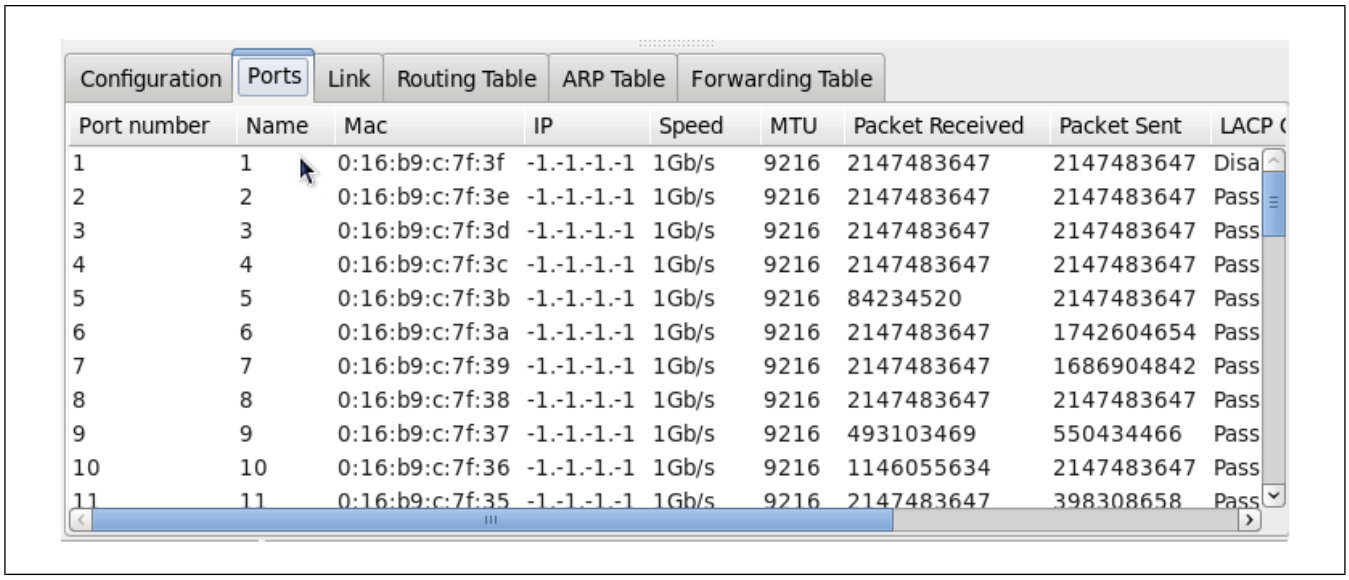

Figure 24: The configuration of the equipment

can use regular expression, based on the regular expression of your database system.

As you can expect, most of those features are available in the command line interface, but not all of them (it does not make any sense to choose displaying color in command line interface...).

At this point in time, a very few features are missing in the interface, mostly button to trigger the discovery algorithm, the update process, the VLAN computation, and the search windows that gives mac-ip correspondence.

4 Future extensions

The demonstration I made to my supervisors few days ago enthused them a lot. They saw a lot of potential in this tools, so that they already have several ideas of future extensions. I will most probably not have enough time before the end of my internship to work them out all by myself. Here is a non exhaustive list of the possible extensions :

- A layer 2 path : it would be very interesting to have the path followed by a packet from a given source MAC address to a give destination MAC address. This is not

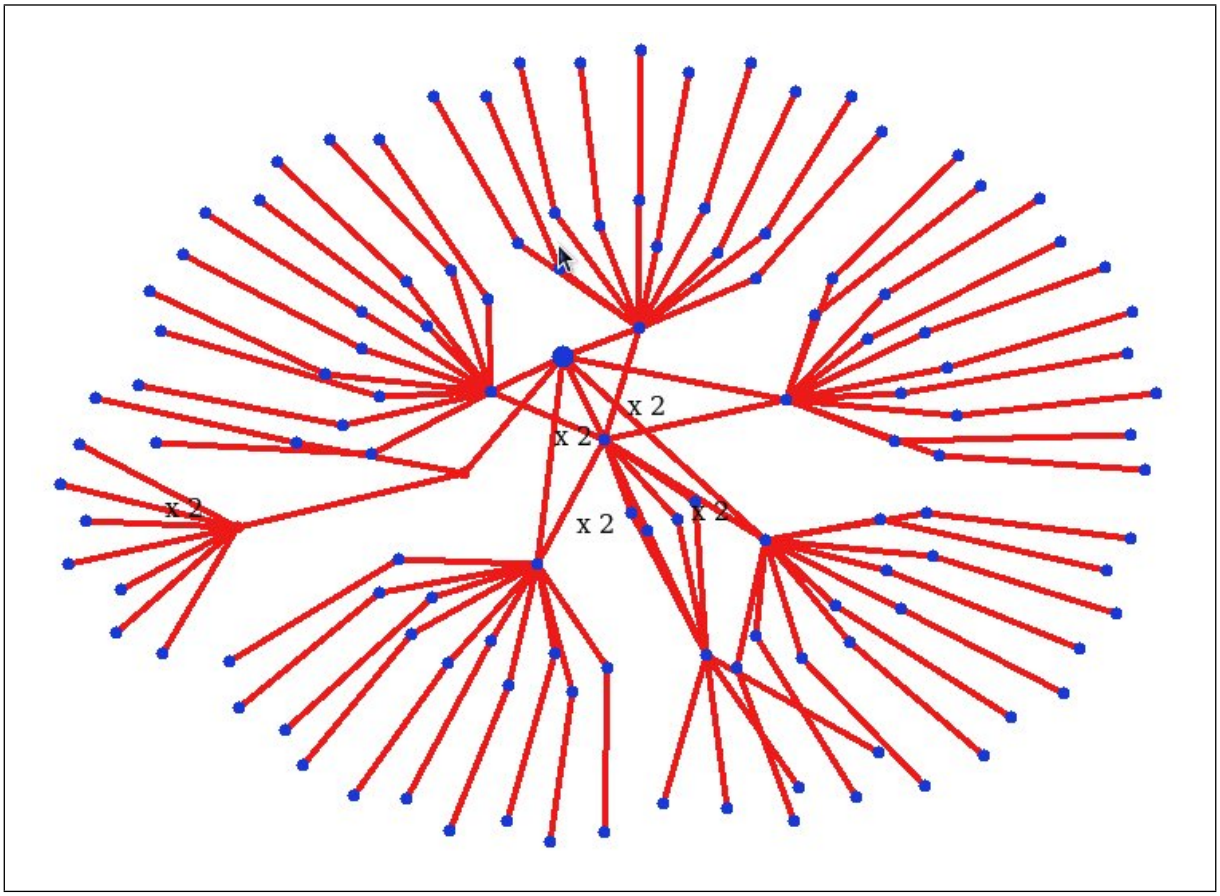

Figure 25: The map of the network

a very difficult algorithm, and all the data needed to use it are already stored in the database.

- Develop an API : an API would allow a third-party program to use the database and the algorithm of my software very easily. This seems to be a very pleasant extension for the LHCb team. Actually, this is very simple to implement : the API just corresponds to a new *View* in my MVC pattern.
- Export the graph : an interesting feature could be to export the graph every time the database is updated. Thus, we could display it on a web page for example. The main difficulty here is the computation time needed to draw the algorithm.
- Split the software in client/server mode : the idea would be to run the discovery and

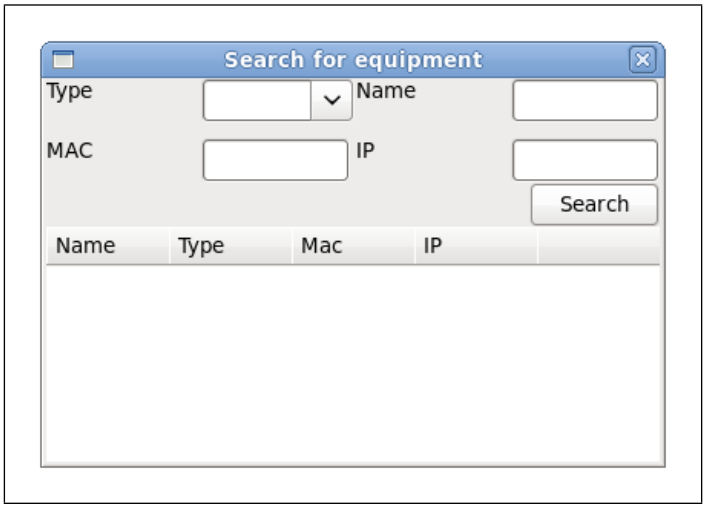

Figure 26: The search tool

update algorithms on a server, and allow client, installed on any computer, to connect to this server. Thus, all the power consumption is out of the user workstation. Once again, the MVC pattern already in place allows this solution : while the model will be on the server side, the view will be on the client site. At first glance, the controler may be in any side, but a more careful analyze would probably show up the best decision. In any case, in order to do this, we will need to adapt all 3 parts, so that they can communicate with each other across the network. This is an interesting extension, but really time consuming.

As you can see, the software is subject to a lot of improvements, but all those extensions are made possible thanks to the well prepared architecture design.

Conclusion

At this point in time, the discovery algorithm is in the final debugging test stage, and it seems like there are no major bugs to correct. The update process now produces a file that sums up the changes, the next step is to apply those changes in the database.

Until the end of the internship, I can focus on finishing the VLAN algorithm, which seems to be in very good way. I will then search an algorithm more efficient than the Force-based one to draw the map. Finally, I will try to make the API, to use my software from third-part tools.

Whereas the schedule I followed was far from the one I expected and that new issues always ruined my plans, those problems were always very educative, and the greatest lesson I can get is to always respect the defined standards.

As a conclusion, this internship has been a really positive experience : the CERN is the best place to have a preview on the research world, and a unique chance to deal with pioneering and huge systems. I could increase a lot my network knowledge, and learn a new programming language. Finally, it is really pleasant to think that the result of my work will be used in such a prestigious institution like CERN

Bibliography

[1] An architecture for describing simple network management protocol (snmp) management frameworks. RFC 3411, IEEE.

[2] Coexistence between version 1, version 2, and version 3 of the internet-standard network management framework. RFC 3584, IEEE.

[3] Introduction and applicability statements for internet standard management framework. RFC 3410, IEEE.

[4] Link aggregation. RFC 802.1AXTM-2008, IEEE.

[5] Management information base for network management of tcp/ip-based internets: Mib-ii. RFC 1213, IEEE.

[6] Management information base (mib) for the simple network management protocol (snmp). RFC 3418, IEEE.

[7] Message processing and dispatching for the simple network management protocol (snmp). RFC 3412, IEEE.

[8] Simple network management protocol (snmp) application. RFC 3413, IEEE.

[9] Structure and identification of management information for the tcp/ip-based internets. RFC 1155, IEEE.

[10] User-based security model (usm) for version 3 of the simple network management protocol (snmpv3). RFC 3414, IEEE.

[11] Version 2 of the protocol operations for the simple network management protocol (snmp). RFC 3416, IEEE.

[12] View-based access control model (vacm) for the simple network management protocol (snmp). RFC 3415, IEEE.

Annexes

Appendix A

The physic at LHCb

Even though this is not the main topic of this report, it is interesting to understand what CERN is doing, and what experiments require such an amount of people, money, and time.

CERN is a laboratory dedicated to particles physics. This branch of the physics studies the elementary subatomic constituents of matter and the interactive relationships between them.

Most of the elementary particles are not constituents of the nature we observe all around us. Thus, to produce and study them, we need *particle accelerators*. The basic principle of an accelerator is to propel charged particles to high speeds and to contain them in well-defined beams. The aim is to make beam collides, either with a target or another beam in opposite direction. The results of such collisions are those "non-daily" particles.

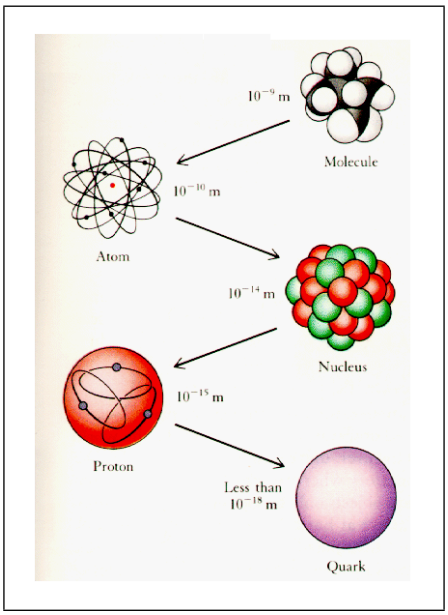

Figure A.1: The composition of the matter : electrons, quark up and down

The LHC is one of those accelerators, and is even the largest and highest energy collider. This machine is able to accelerate a beam at an energy of 7 teraelectronvolts [1].

[1]The electronvolt (eV) is a unit of energy equal to approximately $1.602 \cdot 10^{-19}$ J. By definition, it is equal to the amount of kinetic energy gained by a single unbound electron when it accelerates through an electric potential difference of one volt. We use it to describe energy, momentum, mass, time, distance and temperature

Currently, the state of classification of elementary particles is *the Standard Model*.It describes fundamental forces, using mediating *gauge bosons*. Those forces are :

- the strong force : it is the force that binds quarks together, and that binds protons and neutrons together to make the nuclei of an atom. The gauge boson associated is the *gluon*.
- the weak force : this force is responsible of the β decay. The gauge bosons associated are *W+, W- and Z*.
- the electromagnetic force : the force that causes the interaction between electrically charged particles. The gauge bosons associated is the *photon*. Actually, it is the force that produces all the daily phenomena, such as light.

The model also contains 24 fundamental particles, which are the constituents of matter. All those particles are summed up in the figure A.2

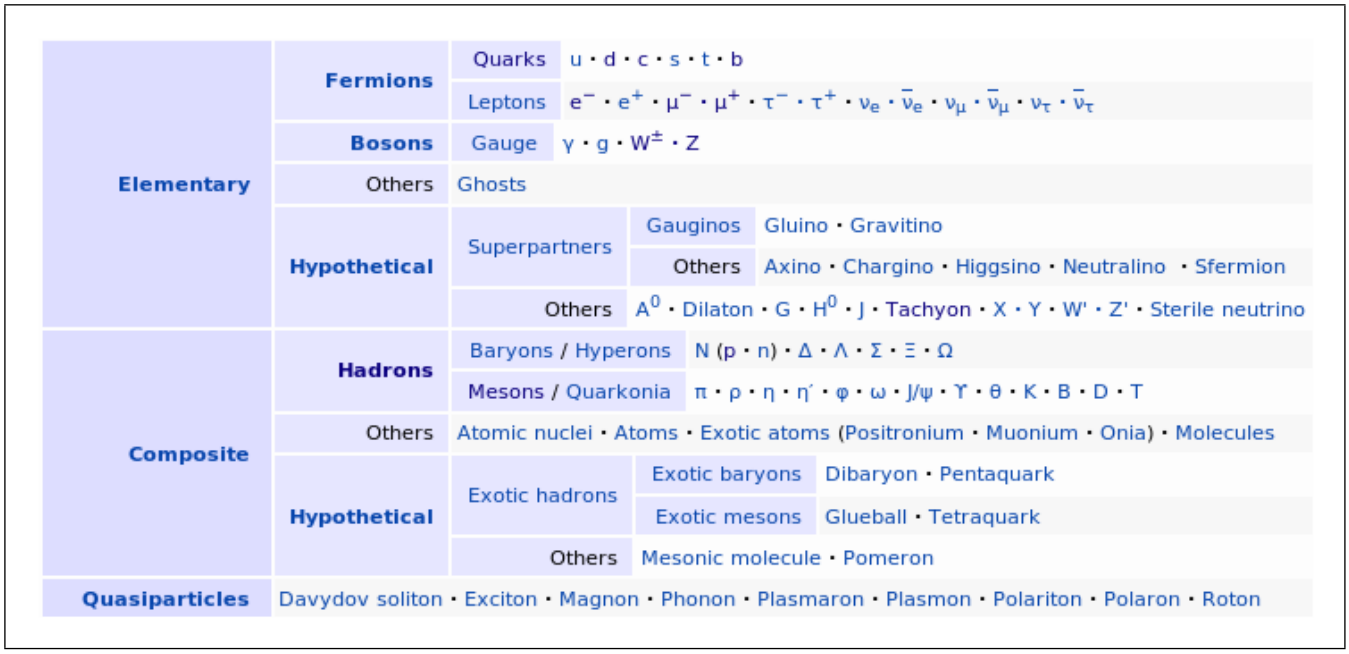

Figure A.2: The standard Model classification

But for each elementary particle of matter, we can find an antiparticle. A particle and its antiparticle put together annihilate in pure energy. The antiparticle is the symmetric of the particle, as if you were looking yourself in a mirror. Never the less, we don't understand everything about antimatter.

As you can notice, some of the particles listed are hypothetical, and the gravity is not in the elementary forces described by the standard model. Thus, the standard model is incomplete, and the LHC is meant to fill it in.

Whereas the quark present in the matter around us are *up and down quarks*, the LHCb focuses on another kind of composite particles made from another type of quark : *the beauty, or bottom quark*. Those composite particles are called *B-hadrons*. The LHCb has the following missions [2] :

- Measuring an upper limit on the branching ratio of the rare $B_s \rightarrow \mu^+\mu^-$ decay. The meson B (B_s) is composed by an antiquark bottom and quark up ($B+$), down ($B0$), strange or charm.

[2] I invite you to have a look at the figure A.2 to see in which column are the particles we are speaking about.

The Muons are elementary particles that are very difficult to stop; we can observe them coming from cosmic ray. The meson B can decay in pair of muon, but very hardly : LHCb has to measure this ratio.

- Measuring the forward-backward asymmetry of the muon pair in the flavour changing neutral current $B_d \rightarrow K^* \mu^+ \mu^-$ decay : normally, the reactions happening in one way can happen in the opposite way. But some of the decay present an asymmetry.
- Measuring the CP violating phase in the decay $B_s \rightarrow J/\psi\phi$, caused by interference between the decays with and without $B_s - \overline{B_s}$ oscillations : as I told you previously, we don't know everything about antimatter. As matter and antimatter were produced in equal quantity, and annihilate in contact one with each other, there should not be any of them left. But even though we don't find antimatter around us, there is a left over of matter : you, me, everything that you can see. That means that at a point in time, a disequilibrium happened, a small difference between matter and antimatter occurred. The *CP violation* is a small difference that physicists want to analyze.
- Measuring properties of radiative B decays