Eclipse plugins for LHCb software development.

Engineer report

3rd year internship

Embedded Systems option (F1)

Volume I of II

Presented by: Grégoire Astruc

ISIMA SUPERVISOR: EMMANUEL MESNARD ACADEMIC DEFENSE ON SEPTEMBER 9, 2011 CERN LHCb SUPERVISOR: MARCO CLEMENCIC
Acknowledgments

I would like to thank my CERN supervisor, Mr Marco Clemencic for his time and his experience as well as the confidence he put in me.

I would also like to acknowledge Mr Hubert Degaudenzi for his input on the project.

I would also like to pay my regards to both LHCb offline and online teams for their warm welcoming.

Finally, I would like to thank Mr Emmanuel Mesnard for his visit.
FIGURES

Illustration 1: CERN Logo................................................................. 2
Illustration 2: LHCb logo............................................................... 2
Illustration 3: Projects structure................................................... 4
Illustration 4: Solution overview.................................................. 7
Illustration 5: Study schedule........................................................ 7
Illustration 6: Eclipse concepts classes....................................... 9
Illustration 7: Eclipse Architecture.............................................. 10
Illustration 8: Python script to Java bytecode.............................. 11
Illustration 9: Package Setup class diagram............................... 13
Illustration 10: Plugins organization......................................... 14
Illustration 11: SetupProject interface....................................... 19
Illustration 12: Project builders class diagram........................... 21
Illustration 13: CMTPROJECTPATH construction........................ 23
Illustration 14: GetPack package selection.................................. 24
Illustration 15: GetPack version selection................................... 25
Illustration 16: New package wizard........................................... 28
Illustration 17: New component wizard..................................... 31
Illustration 18: Plugins architecture.......................................... 34
Abstract

CERN LHCb Offline team, working on experiments around the LHC particule collider, had a number of tools to assist developers. LHCb projects use a domain-specific structure.

Tools are always evolving and some team members started to look for ways to integrate those programs (most of them being CLIs) into more recent programming software like the Eclipse IDE.

During this internship, a set of Eclipse plugins was created. They link the LHCb projects concepts to the ones in the IDE.

Résumé

L'équipe du CERN LHCb Offline, travaillant sur des expériences avec l'accélérateur de particules LHC, dispose d'outils pour assister les développeurs. Les projets au LHCb utilisent une structure spécifique à leur domaine.

Les outils sont en évolution constante et certains membres de l'équipe ont commencé à chercher des moyens d'intégrer ces programmes (la plupart étant des CLIs) au sein de progiciel de développement comme l'IDE Eclipse.

Au cours de ce stage, un ensemble de plugins Eclipse ont été créés. Ils font le lien entre les concepts régissant les projects au LHCb avec ceux de l'IDE.
# Table of Contents

Acknowledgments ........................................................................................................................................ 1  
Figures .................................................................................................................................................... 1  
Abstract .................................................................................................................................................. 1  
Résumé .................................................................................................................................................. 1  
Introduction .......................................................................................................................................... 1  
I. Study introduction ................................................................................................................................. 2  
   1- Company presentation ....................................................................................................................... 2  
      a) CERN ........................................................................................................................................ 2  
      b) LHCb ....................................................................................................................................... 2  
   2- Existing .......................................................................................................................................... 3  
      a) Projects organization .................................................................................................................... 3  
      b) Set of CLIs ................................................................................................................................. 4  
      c) Emacs enhancements .................................................................................................................... 5  
      d) Eclipse IDE ............................................................................................................................... 5  
      e) Wiki documentation .................................................................................................................... 5  
   3- Goals ............................................................................................................................................. 5  
   4- Solutions ....................................................................................................................................... 6  
      a) Analysis ....................................................................................................................................... 6  
      b) Chosen solution .......................................................................................................................... 6  
      c) Schedule ..................................................................................................................................... 7  
II. Technical aspects ................................................................................................................................. 8  
   1- Eclipse IDE ...................................................................................................................................... 8  
      a) Concepts ..................................................................................................................................... 8  
      b) Plugins architecture ..................................................................................................................... 9  
   2- Other technologies .......................................................................................................................... 10  
      a) Python integration ...................................................................................................................... 10  
      b) StringTemplate ......................................................................................................................... 12  
   3- Architecture .................................................................................................................................. 12  
      a) Solution designs ........................................................................................................................ 12  
      b) Synthetic overview .................................................................................................................... 13  
   4- Plugins ........................................................................................................................................ 14  
      a) LHCb ......................................................................................................................................... 14  
      b) CMT ......................................................................................................................................... 15  
         i. Package setup .......................................................................................................................... 15  
         ii. CMT project configuration ................................................................................................. 16  
         iii. Editors ................................................................................................................................. 16  
      c) SetupProject ............................................................................................................................ 17  
         i. Listing projects ...................................................................................................................... 17  
         ii. Running SetupProject CLI ............................................................................................... 18  
         iii. Builders ............................................................................................................................... 20
d) GetPack..........................................................................................................................23
  i.Listing packages...........................................................................................................24
  ii.Getting versions...........................................................................................................25
  iii.Fetching packages.................................................................................................26
  e) Package.....................................................................................................................27
  f) Components...............................................................................................................28
     i.Generators..............................................................................................................29
     ii.Builders................................................................................................................30
     iii.Directors...............................................................................................................30
     iv.Factory..................................................................................................................30
     v.Result.....................................................................................................................31
III.Results – Discussion.................................................................................................33
  1- Realization................................................................................................................33
     a) Architecture............................................................................................................33
     b) Tools.......................................................................................................................34
     c) Achievements..........................................................................................................35
  2- Operation....................................................................................................................36
  3- Exploitation................................................................................................................36
  4- Extensions..................................................................................................................37
Conclusion.......................................................................................................................39
Glossary...........................................................................................................................39
Bibliography....................................................................................................................39
Introduction

My last year internship at ISIMA took place at CERN LHCb from April 2011 to September 2011.

For developing research applications they designed several tools to help non computer scientists, such as physicists. Those interfaces enable them focus on the problem rather than which options need to be passed to the compiler, or how one should use this or that commandline program.

However now these tools are aging and more powerful ones are available. Programs like IDEs assist developers, and allow them to focus even more on their core task.

Mr Marco Clemencic, my supervisor, put some effort to provide an IDE preset with a number of settings to ease the development process. But this was not enough as LHCb projects concepts differ a bit from the one acknowledged by the programming software.

Thus the subject of my work being finding ways to link the LHCb tools with an IDE to have both their benefits and as little caveats as possible.

After presenting the organization I worked for, I will provide an outline of the problem. Then I will go through implementation and technical details before dealing with the results.
I. Study introduction

1- Company presentation

a) CERN

CERN stands for Conseil Européen pour la Recherche Nucléaire in french, which translates to European Organization for Nuclear Research. As its name suggests it is an european organism dedicated to physics nuclear research and composed of two main branches. Though originally regrouping only european countries it is now composed by researchers from all over the world.

On the first hand one can see the CERN as a large project management facility. For instance it handled in 2011 the construction of a particle detector driven internationally which was successfully installed on the International Space Station (ISS).

On the other hand the CERN is a scientific institution. It provides substantially powerful computing tools as well as physics detectors for the research field. It is famous for its particle accelerator and collider, the Large Hadron Collider (LHC), 100 meters underground Switzerland and France.

Around it revolve four main experiments all looking for answers about the very foundations of our universe. I was working for the LHCb, which looks into violations of the matter/antimatter symmetry.

b) LHCb

The LHC beauty experiment is amongst the smallest detectors around the LHC. Still it regroups
about 700 scientists from 52 different universities and laboratories [LHCb collaboration]. It was meant to last ten years from its beginning which means it should end somewhere around 2018.

It is divided in two main groups: online and offline. The online group is responsible for the behaviour of the detector in its operation while the offline handles data analysis. My work was for the offline team. It was supervised by Marco Clemencic from april 2011 to september 2011.

2- Existing

a) Projects organization

In LHCb, there are around thirty projects. Each of them being itself composed of several packages.

A package is a set of files, organised according to some directory structure, which provides some well-defined, circumscribed functionality. It is the basic unit of software development and most LHCb packages contain code that can be compiled into one or more libraries.

A project is a set of packages that are grouped together according to some functionality, for example all Gaudi packages (Gaudi project), most public LHCb header files and base class libraries (LHCb project), all reconstruction packages (Rec project), all packages specific to the simulation application (Gauss project). All packages within a given project are released as a single unit: changing just one package in a project implies a releasing a new version of the whole project. Choosing a given version of a project automatically selects a single version of each of the packages in the project.
As with every collaborative development they also rely on some tools to manage the different versions of projects and packages. Another aspect is that most of the developers are not computer scientists but physicists and researchers. Therefore they must not focus on underlying problems regarding the build system nor about how they should retrieve a project or a package.

Moreover, there is also a lot of complications due to the vast community working around those project. Multiple versions are shared and maintained. Those issues will be detailed later in this document.

Regarding this a number of tools were developed internally to help them focus on their task. But computer sciences evolve and Marco Clemencic wanted a way for people to work in a more efficient way.

b) Set of CLIs

People are able to set up a project environment and retrieve a package through two CLIs. Though already an improvement in comparison to running more complex ones this still has a some caveats. For example, if a user does not know the name of a package he would like to retrieve, the complete list is presented on screen without any search field. One must also download packages from the repository one by one, creating a lot of overhead due to the program invocations.
c) Emacs enhancements

Those features assist the user into creating their own sources following the LHCb guidelines. Some projects define interfaces, or algorithms, which allow researchers to interact with the data in LHCb databases. Emacs set up all the boilerplates needed for a given algorithm. Also, though providing syntax highlighting, autocompletion and some other neat features, emacs is not as fully-featured as an IDE.

d) Eclipse IDE

Marco Clemencic set up a set of plugins useful for LHCb developments. Though they allow developers to work efficiently, they do not know about the LHCb projects structure. Therefore, a large effort must be done using the CLIs prior to be able to use Eclipse. This extra amount of work is likely to discourage most users even before they start feeling the advantages of using an IDE.

e) Wiki documentation

To help newcomers, and as a handbook, the LHCb software team wrote wiki pages documenting the whole process. They provide a good starting point but it forces the user to switch back and forth between her console and her browser. The best practices being to put as many inline help as possible so the user do not have to read a manual.

3- Goals

The objectives of this internship were to improve the projects' developments for the LHCb Offline group.

Most of the existing tools are Command-Line Interfaces (CLI). It means that the user must open a terminal and type a few command in a not-so-friendly interface. The main requirement was a better integration
with modern development tools such as an Integrated Development Environment (IDE) which assists the programmer in its task by providing features such as autocompletion, syntax highlighting, an outline of the source file and so on.

Subsequent improvements were a more user-friendly interface for any given CLI. Instead of having to remember more or less complex command line options, users should be guided throughout the process of setting up their project.

All of this lead to the core analysis of the problem: how can it be improved while not rewriting every single tool from scratch?

4- Solutions

The analysis was mainly done by my supervisor prior to the internship. Therefore I will describe it briefly before giving the choosen solution and moving onto the technical details.

a) Analysis

According to what was said in the previous section, Mr Clemencic started using the Eclipse IDE for his own developments and tried to figure a way to make it more efficient. He came up with the idea of developing a series of plugins which would connect together some scripts and concepts from LHCb to the IDE.

b) Choosen solution

Thanks to the powerful plug-in system powered by Eclipse, the choosen solution was fairly straightforward. It was decided that several plugins would be developed, each representing a feature already existing through either CLIs or Emacs. The order in which they were developed depended on their importance.
The priority was for users to be able to set up a project without having to leave Eclipse.

c) Schedule

Illustration 4: Solution overview

Illustration 5: Study schedule
II. Technical aspects

Throughout this chapter technical details regarding this project will be provided. The designs for the solution will first help for the comprehension of this part. Then, the architecture will be detailed for a deeper view. Tools and softwares used for the good development of this internship will be detailed as they appear.

1- Eclipse IDE

a) Concepts

Eclipse works like any regular IDE except it is very flexible. It is an opensource platform developed by a large community and supported by IBM. The key concepts are the Platform, the Workspace and the Resources.

The Platform is the core framework, managing almost everything. The Workspace is responsible for managing resources such as projects, folders and files. In a given workspace, which represents a folder on disk, the user may create several projects with or without relations between each other. The Resources is a concept for everything existing in the workspace ranging from projects, to folders, regular files, links, etc.
b) Plugins architecture

In the Eclipse IDE, everything is a plugin. The core loads them on demand at the time they are needed. A plugin can provide several extension points, and implements as many extensions as needed.

An extension point describes an interface which other plugins can implement through extensions. It allows a plugin to delegate an implementation to another plugin. Several extension points are already defined by the eclipse platform: to create an editor for a given type of files, to develop new Wizards for importing files into the workspace and so on.

Other plugins can then extend existing ones by implementing those extension points. This is called an extension.
On Erreur : source de la référence non trouvée, each plugin appears as a box with rounded corners. They are plugged in the platform.

### 2- Other technologies

#### a) Python integration

Since at LHCb Offline they have a lot more expertise with python than with Java, my tutor asked if there was a way to develop plugins directly in this language. It was not a priority at first, and I was planning to check on that in the end of my internship. However, some time along the way I realized that most of what was written in Java already existed in the LHCb Offline code base.
Python can integrate with a Java VM through Jython as shown on Illustration 7. It basically compiles the python script into Java bytecode directly executable by the Java VM with little overhead. But for developing user interfaces, which must respond as fast as possible, this overhead was a bit too much. It was then decided to develop only small functionalities in python, while leaving the heavy lifting to Java itself.

Just like Python requires a PYTHONPATH in which it will look for modules, Jython requires the JYTHONPATH environment variable. However, due the great diversity of environments (personnal computers, CERN employee computers, shared servers...), it was decided to leave the environment as is, without adding this new variable. But the problem remained about how telling to the Jython interpreter where to look. Actually, the JYTHONPATH adds entries to the sys.path variable. By adding manually these entry, I was able to remove the need for the JYTHONPATH. The Code 1 abstract shows how the engine is initialized.

```java
PySystemState engineSys = new PySystemState();
String libPath = "/Lib";
engineSys.path.insert(0, Py.newString("__pyclasspath__" + libPath));
// Get the PYTHONPATH from the environment.
String pythonpath = System.getenv("PYTHONPATH");
// Get the PATH separator (; for Linux, ; for Windows).
String separator = System.getProperty("path.separator");
// Get entries from pythonpath : split it using the separator.
List<String> pathEntries = pythonpath.split(separator);
// Insert each entry in the engine's sys.path
for (String path : pathEntries )
    engineSys.path.append(Py.newString(path));

Code 1: PythonEngine initialization
```
b) **StringTemplate**

StringTemplate is a Java library for creating templates. With it, we can clearly separate the data from their representation. Templates are files with a specific syntax. Data are pushed to the template through a controller code in Java.

Using StringTemplate enables my tutor and the LHCb team to quickly change a template without bothering with the Java code.

---

**3- Architecture**

a) **Solution designs**

The designs were made for flexibility and ease of maintenance. Every Eclipse plugin must be developed in the Java programming language. However, most users at LHCb including Marco Clemencic are not familiar with it. This is why a large amount of effort was put into making those plugin as easily modifiable as possible. Therefore a good share of design patterns were used: they are language-agnostic since they are concepts and allow programmers to have a better understanding of the software by sharing a common structure.

The ground step was modeling the LHCb projects' organization to reflect this fairly complex structure. It was taken into account that the build system currently in use is CMT. However, there were plans to switch to CMake. My approach had to allow the possibility to use one system or the other, or even both.

Another issue was the complexity of setting up a project and a package. To make it easier to change according to the needs of LHCb and its evolution, the setup is dedicated to a chain of instances which will set up the directories, files etc. according to some parameters. This enforces a clean separation between the package data and its setup.
b) Synthetic overview

After this it was time to organize the different plugins. One way would have been to use a single large plugin which would include all the required features. But this is not a good practice, nor it is recommended by Eclipse to do so. Hence splitting into several plugins.

Since the tools were already existent the plugins actually map their CLI equivalent in a one-to-one relationship. Along the way a bit of code factoring occurred as a lot of them shared common code.

Before a more detailed approach, the following Erreur : source de la référence non trouvée shows how the different plugins are tighed
together.

Illustration 10: Plugins organization

4- Plugins

Now that we have laid down the basics and concepts needed, we will focus on the main task. The plugins developed are not given chronologically but instead ordered first from common projects, like LHCb and CMT to more task-specific ones, and then by usage.

a) LHCb

The LHCb plugins holds common concepts mentioned in Solution
designs (page 12), sharing them across other plugins.

It also defines two extension points: one to setup packages, and one as a mean to retrieve packages.

A package setup may be a very complex operation, and there is very little chances to know every aspects as well as predicting how it will evolve. Therefore, the LHCb plugin defines an extension point allowing new ways of configuring a package to be added in the future, as well as easily changing or removing existing ones. For instance, CMT needs to add some directories and files to the package. This is done through the extension point.

To retrieve packages as well, there is almost no way to predict evolution. As of this writing, packages are fetched from SVN repositories. The extension point defines an interface which allows interchangeability. The GetPack wizard, detailed page 23, is an example.

At last, this plugin manages Jython engines. They are used by other plugins to create their own instances of this python interpreter for Java through one single common interface.

This project holds very little logic, since its main goal is to factor code which are commonly used by every other plugins.

b) CMT

The CMT plugin handles everything revolving around CMT as its name suggests. It features a package setup, CMT project creation and two editors.

i. Package setup

To properly create a CMT package, it needs to have a cmt folder containing a requirements file. Its content depends on some package information such as its name, the programming languages used
(currently C++ and/or python) and so on. For this matter, I used a template file which have placeholders filled by a controller class.

**ii. CMT project configuration**

Whenever a user creates a new CMT project, a project.cmt file is created containing basic information. Just like the requirements for a package setup, it uses a template. However since we only create a file here, there is no dedicated class. The code responsible for creating this file is directly in the plugin main class.

**iii. Editors**

Marco Clemencic had already developed a tool using another Eclipse plugin. But this added an extra dependency and other efforts. Originally as an exercise, I decided to write an editor with syntax highlighting for requirements files. It proved to work quite well and we decided to keep it. Later, my supervisor asked me for the possibility to provide syntax highlighting for project.cmt files as well. Both filetypes share some elements of syntaxes since they are both of the CMT kind.

To avoid duplicated code, the best decision was to factor common elements through inheritance.

**Syntaxes**

Syntaxes are described in JSON files. They are based on the official grammar description from the CMT website. This gives a flexible way for changing syntax elements without the need to change the plugin code.

At first there was only a single JSON file holding the requirements file syntax. With the request of a project.cmt editor, I needed to figure a way to avoid writing common parts twice. As a result shared grammar lies in one file, essentially holding delimiters for variables and litterals, while specific items are read from separate files.
c) SetupProject

SetupProject is the name of the CLI used when a user wants to setup a new LHCb project on her computer. It creates the project directories structure and configure environment variables for building. The CLI tool also does more than that but those two elements were the main requirements.

I will first describe how I displayed available projects to the user. Then how the SetupProject CLI is ran from Java and finally the process of creating a project in the workspace using the Builder pattern.

i. Listing projects

```python
# From LHCb Scripts
import LbConfiguration
# From Java
from ch.cern.lhcb import Project
from org.eclipse.core.runtime import SubMonitor

lbProjects = LbConfiguration.Project.getProjectList()
# monitor is external. Injected from Java (see SetupProject.java)
sub = SubMonitor.convert(monitor, "Creating list", len(lbProjects))

for p in lbProjects:
    # same thing with projects.
    projects.add(Project(p.Name(), p.NAME()))
sub.worked(1)
```

Code 2: Listing projects from python

There was no easy way for someone to easily get a list of projects. One would have to look at some webpage to get ahold of its name and then ask for it using SetupProject. In the LHCb scripts code there was a python module which is aware of the projects' list. This is actually when the need for a python integration rose. Using it, the process for retrieving this list was straightforward: a little python script (Code 2) is executed from the Java controller code (Code 3). Since the bindings work both ways, a list is initialized in Java and filled in the python script. At last, it is presented to the user in a combobox.
ii. Running SetupProject CLI

SetupProject CLI may require several options passed to it depending on the version of the project requested. For instance, one could want a stable release of a project, or the bleeding-edge version from the nightly builds, or almost anything in between. Moreover these options are order-dependent: for two given options, their order on the commandline will alter the output of SetupProject.

This implied two main constraints: regular users should have a way to request either stable versions, nightlies or the current development release whereas more advanced users should have the opportunity to customize options even more, with the ability to rearrange them and specify custom ones.

Commandline options

Basically, an option is a composite of two things: a switch and an argument. The switch represents the option name, such as build-env or nightly and the argument definition is self-explanatory.

They are arranged in a WritableList, an special Eclipse list, which can be bound to an interface.
Options are always added to this list. When SetupProject needs to be run, the list is passed to the SetupProject process. Option objects are serialized through the toString() method.

Nightlies

Like mentioned above, the nightly is a special option. Latest versions are built on CERN servers every day and every for all projects. This option actually takes two arguments: a slot, roughly a build configuration, and a day. The list of slots is read from an XML configuration file on the LHCb code repository. Since there was no need for the ability of writing new values to this list, it is presented through a standard combobox. It still enables the user to fill it with a custom value while avoiding some complexity on the programming side.

User interface

The user interface (Illustration 6) design had to be simple enough
for casual users not to be lost but also powerful enough to allow more advanced programmers to customize the commandline according to their needs.

The approach used was a tabbed design: two tabs are presented to the user.

A Basic tab where the user has a few choices presented with radio buttons. Whenever the radiobutton is changed the option list is first erased then written with the only value gave by the radio button setting. An Advanced tab gives a lot more choices, namely adding other options and arranging them. New option are then pushed to the end of the list.

This approach allow the rest of the application to use the same structure, which is reading from the list, whether a single Option is set through the Basic tab or more advanced settings are in play.

All of this is presented in a wizard page. When the user hits the Next button, SetupProject is run with the given options. The output is displayed on the next page in a list view, where each line produced by the CLI corresponds to one row in the list.

When the user press the Finish button, the project is created in the workspace through a set of builders.

iii. Builders
A `ProjectBuilder` interface describes one step during the project creation process. Using the Builder pattern, it is easy to add, remove, or reorder the different step. The Wizard acts as a director: it manages the list of steps and feeding them with the proper input. At this time, three builders are defined.

Illustration 10 shows the class diagram of the builders and Code 4 is an excerpt from the Wizard code.

```java
// Configure the different steps.
List<ProjectBuilder> builders = new ArrayList<ProjectBuilder>();
buffers.add(new CreateProject());
buffers.add(new NaturesBuilder());
buffers.add(new EnvironmentBuilder(pathEntries));
// Apply them in order to the project.
for (ProjectBuilder buildStep : builders) {
    buildStep.build(project);
}
```

*Code 4: Director code in SetupProjectWizard*

**Project creation**

This is the first mandatory step. It registers the project in the workspace, creates the Makefile as well as a couple other directories. It
also calls the CMT plugin to populate the project.cmt file.

**Natures settings**

After the project is created in the workspace, natures are added to it. In Eclipse, every project may have one or more nature. A nature is sort of a label for a project, allow eclipse to adapt its interface depending on the project nature. Here, as we use a Makefile, a C++ nature must be added to the project to benefits from other plugins.

While adding natures, a C++ configuration is set. It relies on the famous CDT Eclipse plugin which handles everything oriented towards C/C++ development. Unfortunately, there is no easy API for adding new configurations to a project programmatically. Reading an unofficial [CDT FAQ] taught me a way to create one for a given project using classes marked as *internal* in CDT. It works fine, but there might be issues rising in the future if those classes were to change.

**Environment configuration**

At last the environment is configured. Using the configuration previously set, environment variables necessary for building the project properly are added or removed to the environment.

For CMT to work as expected, the `PWD` and `CWD` environment variables must be unset. Then a `CMTPROJECTPATH` variable is set. Its value depends on several items. First the project workspace. Then, the different path passed to SetupProject are added.

If one or more nightly configuration were used, specific entries are read from an XML configuration file specific to each nightly and appended to `CMTPROJECTPATH`. At last, `LHCBPROJECTPATH` is appended.
Each path entry in the `CMTPROJECTPATH` must be unique, the first encountered with a left-to-right reading being the one kept in case it appears multiple times. To fulfill this aim without parsing the entire generated string I used a LinkedHashSet. This standard Java container shares characteristics from both a LinkedList and a Set. It means that value are added in order just like a regular list, but only one occurrence is kept if an attempt to add the same value is made.

Once a project is setup, the usual way is to retrieve packages from the codebase before anything else.

d) GetPack

The program used to retrieve packages from LHCb repositories is called GetPack. Currently if the user does not know the package’s name she would like to get, GetPack will display a ncurses interface. There is no way to search for a package by typing in a few letters and the list is quite long.

Working with Mr Clemencic, we designed an update to this program to be able to retrieve the list of packages easily. Likewise, the current version does not allow a user to specify more than one package at once on the commandline. This issue was addressed and will be discussed in the latter part.
i. Listing packages

To list packages, an XML output was devised. Passing two arguments --list and --xml to getpack without specifying a package displays the full list of available packages. They are nested in a tree structure, first ordered by projects and then by hats. The list is then displayed to the user with a treeview.

![GetPack package selection](Illustration 14: GetPack package selection)

A search field is also displayed at the top allowing someone to type in a few characters of the name and the tree view will automatically display only items containing this text. This required more of a documentation than a programming one. This being a very common feature in UI designs, Eclipse already provided all the necessary infrastructure. Therefore it was mainly a matter of providing the right input to get it done.

When a user select packages, they are added to a list, which is passed to the next screen where she can select which versions she needs.
**ii. Getting versions**

For any given package, the version one should download depends on both the project and its version it is downloaded for. At the beginning of my internship, getpack was directory-dependent: it had to be run within the project directory to be able to guess the right version. This meant some programming extra-efforts at first, but Mr Clemencic kindly added yet another arguments so that getpack could be started from any directory while being able to correctly find a default version by specifying a project directory on the commandline.

I first tried to parse getpack standard output to read the available versions. But though being easily readable by a human, it was hard to parse in a nice and efficient way. As a result an XML output was designed to retrieve packages' versions.

Once parsed they are displayed in comboboxes for each package. The default version is pre-selected and the user may change it or type his own if he knows what he is doing. Like for every other plugins this gives a great deal of flexibility which was one of the strongest requirements.

![Illustration 15: GetPack version selection](image)

Now that the versions are set for each package, the user can press the Finish button to begin downloading them.
iii. Fetching packages

Packages are fetched whenever a user has selected packages and hit Finish without displaying the versions selection page, or he went through both pages. In the first case packages versions will be set with the one guessed by getpack.

Packages names and versions are fed one by line to getpack standard input (see Code 5). In a first iteration, the commandline tool was run in blocking way meaning the user could not do anything else but watch the progressbar progressing. Since checking out packages can take a long time, it was decided to run this as an Eclipse job.

Jobs in Eclipse first display a dialog with a progressbar. But one can decide to hide this box and start work on something else. The job then continues its work in the background and the user can continue his developments.

A console view is also attached to the process. The getpack standard output is tighed to this view reporting progress just like one would see it on a terminal window.
Now that the user set a project and got one or more packages, he can create a new package.

e) Package

The wizard to create packages was actually the first one develop. It was both a way for me to get started with the eclipse plugins architecture and for my supervisor to see what was feasible and conceivable.
It consists in a single page wizard where user can provide some information such as the package name, its initial version, which languages are used...

Most of the concepts were already explained in the LHCb plugin paragraph (page 14). The Wizard then adds package setups implementations to the package according to the settings from the wizard page.

After setting up a package, one can start add his own code.

f) Components

In LHCb, algorithms and tools developed by researchers are called components. Several projects such as Gaudi or DaVinci describes interfaces for those algorithms. There is a lot of boilerplates users should
not worry about. Instead, they should only focus on selecting the right kind of interface, give a name to their algorithm and start the implementation.

Currently there is a set of emacs extension to do all this. However, there is very little logic and it is not very flexible. For instance, for one kind of interface called a GaudiTool, one could actually specify more than one interface, which is not feasible with emacs. Likewise since those interfaces are C++ there are always two files created: a header and a source file for the implementation. With emacs, one needs to first create the header, customize it, then start emacs again to create the source file accordingly while paying attention to give it the same name.

In computer sciences there is a principle called Don't Repeat Yourself (DRY). This approach violates it.

Through the plugin I have developed, a user only specifies once his class name and the kind of interface he wishes to implement. Then, either both a header and a source file are created or just a header file if he is only extending an interface definition.

Since this aspect of the plugins set is most likely subject to evolutions I applied several patterns to reduce the amount of changes needed.

\textit{i. Generators}

Files are created using generators. A \texttt{FileGenerator} is an abstract whose attributes are set from the user inputs. It provides an abstract method \texttt{generate} which purpose is to actually generate a file in a given folder.

Since everything relies on templates, a \texttt{TemplateGenerator} was written. It can load a template file, populates the placeholders, and generate the file.

They can be linked together thanks to the \texttt{Decorator} pattern. It is a
way to transparently extend the behaviour of a class.

One generator class correspond to one kind of file. Generators are constructed with builders.

\textit{ii. Builders}

The Builders for the components follow the same design as for a project. One Builder can properly create one kind of generator. At the time of this writing there is only one implementation: the TemplateBuilder. Some parameters are set at construction-time regarding which template to use.

The abstract build method then returns a FileGenerator. This looks like an extra and obsolete layer of abstraction but it allows a better separation of concerns and reduces the changes needed for building a file.

\textit{iii. Directors}

The Directors manage steps necessary in building component files. They chain together builders, and, through iterative calls, construct the FileGenerator and return it.

\textit{iv. Factory}

The Factory manages directors. Each director is given a key string to identify it. Then upon request, it returns the correct Director according to the name specified as an argument to the method.
**v. Result**

The result of this architecture is that the controller code in the wizard requires zero change. To add a new type, one need to follow this procedure:

- Add a new type radio button to the wizard page. Helper methods are provided as well.

- Create required generators. For instance one could like to generate a file from a webservice.

- Create the builder for this generator. Here, the extra parameter could be the base of the URL.

- Add a new Director for this type.

- Register the Director in the Factory.

This is the fewest required changes I could come up with. If the

---

**II. Technical aspects**

2011 – Grégoire Astruc
files are template-based the third and fourth steps become unnecessary since both the generator and its matching builder already exists. Since my supervisor is not very familiar in Java, but still being a developer, this approach allows him to extend those aspects as easily as possible.

I have here presented the technical aspects of this internship. I will now discuss the results of this work.
III. Results – Discussion

In this chapter I will discuss about the outcomes of the project. After reminding its architecture and functionalities, I will describe the project in its operation, how the different pieces fit together. Then I will talk about how it runs in exploitation at LHCb. Finally I will describe how it can be improved and extended with ideas which were not implemented because of the lack of time.

1- Realization

In the previous chapter, I described the project architecture from a developer point of view. In the next figure is displayed how the different plugins are connected to each other.

a) Architecture
b) Tools

Throughout the realization of this project, a large number of tools were used.

Obviously everything was done using Eclipse. This fully-featured IDE provided a number of facilities which made the development easier and allowed me to be a lot more focused on the task at hand rather than bothering with other details.
The Eclipse Plug-in Development Environment (Eclipse PDE) consists of a set of plugins dedicated to the creation of plugins for this platform. Though useful most of the time, I sometimes ran into troubles because of files not correctly updated or rebuilt. This happened a few times and was always overcome with ease or with help from the Internet.

To share the project with my tutor we settled on Git. It is a powerful versioning system that allows developers to commit and push changes from their local copy of the repository to the common one. This way, all the code lies in a single place. Moreover, if this project is continued in the future, one could see all the history of changes and most likely why certain things were made the way they are, either because of implementation or deployment concerns.

Bugs and features requests were gathered on a CERN Savannah project page. Savannah is a web application on which users can report bugs and ask for new features for a given project. This adds a trace to what was done and what still needs to be done.

Other third-party tools such as Pencil, to draw interfaces, and ArgoUML, for UML models, were used. These are free and open-source. Even if they were to die, somebody could still get an old version and be able to open the different document files which resides with the source code.

c) Achievements

The project objectives were to provide UI layers for CLIs as well as replacements for Emacs extensions.

The CLIs have been integrated or superseded with this set of developments with success. However, some features are still lacking. The CLIs still gives a lot more flexibility and some parts of the UIs should allow expert users a lot more customizations.

Regarding the Emacs replacements they have been integrated into the Eclipse platform. Now users can transparently use one or the other,
but Eclipse offers more than Emacs, enhancing the user experience and easing her developments.

2- **Operation**

The product was mainly tested by my supervisor, Mr Marco Clemencic. We worked on small iterations. Every time I had finished developing a feature, he would test while I would be working on the next item. Using this method, I was able to respond quickly to modification requests as well as fixing bugs while still having the ability to develop new features.

In the end of the internship, Mr Clemencic found a user to test the plugins. He was new to CERN and therefore had no a priori regarding the tools he should use. He reported some issues, which were resolved or are likely to be solved by the end of the internship.

The product in operation enables a user to setup a project, get packages, create new packages, and be assisted in created new LHCb components. He does no longer need to switch back and forth between the IDE and the commandline. The project works for most basic use cases as well as a few advanced cases. Actually Mr Clemencic started using them as soon as they were made.

3- **Exploitation**

Marco Clemencic deployed a shared Eclipse installation on which he installed the plugins I developed as well as other third-party plugins. Now they are part of this setup and can be used by anyone. Moreover, oneself could install them for his own private Eclipse installation since they are rendered public through an update site on the CERN servers.

Unfortunately, besides the tests by my supervisor and the
experiment from one user, there were no feedbacks about the product deployed. But the regular testing and usage by my tutor showed that the product is stable enough for a day-to-day use.

4- Extensions

There is still a lot work to do to be able to work efficiently and completely with Eclipse without having to switch to a terminal at one point or another.

The package creation is still simplistic. It does not cover all the use cases. For instance a package may be a shared library, a static one, a component, etc. Currently the plugin only create a requirements file with the basic structure and does not cover all of this. However, compared to the old way where one needed to copy this requirements file from a dummy package it is a large improvement. And hopefully, with the architecture set for this, it should be fairly straightforward to extend the behaviour of the wizard by adding new possibilities.

Setting up a project only, besides creating some folders and files, only configures the build environment for one default configuration. But projects use both a default and a debug configuration where the environment is slightly different. Thought this should be addressed by the end of the internship it still something that needs to be dealt with.

On the same scale, projects are run with specific CMT commands. A workaround was found to make it easy to run from Eclipse. But users are unlikely to be able to use all the debugging features provided by the IDE which are useful in the process of finalizing some application. The idea for the future would be to provide a CMT runner for Eclipse which would take care of setting up the runtime environment as well as interfacing with the Eclipse debugging system.

Finally the GetPack wizard works nicely with Eclipse, as well as SetupProject, but it does not fully represent the projects/hats/packages architecture of LHCb. The everlasting goal would be to create a Rich
Client Platform (RCP) which would cover all the aspects for LHCb developments.

Though this is likely to take a larger amount of time than a 6-month internship, the plugins developed here should be a good starting point.
Conclusion

To improve the developing experience for non computer scientists like physicists at CERN LHCb, team members started providing tools for internal use.

The tools were getting old, and more powerful ones were made available, assisting even more the developer in her task. My CERN supervisor, Mr Marco Clemencic, began releasing a shared Eclipse IDE installation with a preset selection of plugins to help users.

However this still did not reflect all the aspects of LHCb projects organization. Therefore my work lied around these aspects, integrating LHCb tools in the Eclipse IDE.

Most of the original ideas were successfully integrated while other arose during the internship. Some difficulties were encountered along the way, like finding the right technologies or changing some LHCb CLIs behaviour to be more easily integrated. Some were overcome without anymore problems, while others needed a little more work or reflexion.

The set of plugins created covers standard scenarios. More work would be needed to handle all of them and aiming towards a complete LHCb RCP.
Glossary

Command Line Interface. Interactive program run from a console........................................ CLI ......................................................................................................................................................, 39
Integrated Development Environment..................................................................................... IDE ........................................................................................................................................ 1, , 39
Large Hydron Collider. Particule Accelerator......................................................................... LHC ............................................................................................................................................... , 39
Bibliography