Analysis, specification and prototyping of a new approach to graphical user interfaces for CERN’s accelerator controls

Olivier Da Silva Alves

WORKING VERSION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rosaldo Rossetti
Second Supervisor: Grzegorz Kruk

July 24, 2017
Analysis, specification and prototyping of a new approach to graphical user interfaces for CERN’s accelerator controls

Olivier Da Silva Alves

Mestrado Integrado em Engenharia Informática e Computação
Abstract

Maintaining an infrastructure as complex and advanced as the different accelerators at the European Organization for Nuclear Research (CERN) is an incredibly challenging task. The purpose of several software created by the Controls Group is to monitor the different accelerators present at CERN, and make sure everything runs as intended. As of today this task is handled by one of CERN’s departments, namely the Beams Department. This entity is fragmented into groups and each of them has a specific purpose.

The group in display in this dissertation is the Controls Group:

The Controls Group is responsible for the specification, design, procurement, integration, installation, commissioning and operation of the controls infrastructure for all CERN Accelerators, their transfer lines and the Experimental Areas.

The Controls Group is currently running more than 800 applications, out of those more than 600 are using a graphical user interface, made in Swing. The point is that Swing cannot be invested in anymore, it is aging, less and less people are interested in it and due to the actual CERN employment policy, it has become harder and harder to find a suitable candidate to develop in this technology.

The future way for GUIs in the Controls Group is not straight forward. Nowadays there are many technologies that offer a powerful way to create applications; one can think of Web and all the frameworks that exist for it.

In this work one will go through many stages, from choosing a technology to go forward, all the way to an actual implemented solution.

The task ahead is relatively simple. One needs to choose a direction, develop a structure and tools around it so that the transition for a developer coming from swing happens to be as smooth as possible.

By the end of this dissertation one will understand what has been chosen and why. One will have a full overview of the implemented solutions, and finally, one will understand more about the current development infrastructure of the Controls Group, its community and its specific requirements.
Abstract
Acknowledgements

I would like to sincerely thank my colleague Mark Buttner for his impressive dedication toward making this project happen. Grzegorz Kruk, who helped me during the whole process guiding me step-by-step, and finally, Vito Baggioolini for allowing me to join his section and work on this subject.

Olivier Da Silva Alves
“Innovation is serendipity, so you don’t know what people will make.”

Tim Berners-Lee
## Contents

1 Introduction  
1.1 Context and motivation ................................... 3  
1.2 Goals ....................................................... 3  
1.3 Expected contribution ................................... 4  
1.4 Structure of the document ................................. 4

2 Choosing a path  
2.1 Introduction .................................................. 5  
2.2 Today’s situation ............................................ 5  
  2.2.1 The controls community adopts Java .................. 5  
  2.2.2 Java and Swing ......................................... 6  
  2.2.3 From the 90s onward .................................... 6  
  2.2.4 The end of Swing ....................................... 7  
  2.2.5 The present situation .................................. 7  
  2.2.6 We are left with a choice .............................. 8  
2.3 Evaluating a technology .................................... 8  
  2.3.1 Richness and aesthetics ................................. 9  
  2.3.2 Recruitment ............................................. 11  
  2.3.3 Learning Curve ......................................... 12  
  2.3.4 Development ........................................... 13  
  2.3.5 Deployment ............................................. 15  
  2.3.6 Mobile and tablet capabilities ......................... 16  
  2.3.7 Long term stability and support ....................... 16  
  2.3.8 Comparison summary .................................. 18  
2.4 SWOT analysis ............................................... 18  
  2.4.1 Web Analysis ........................................... 18  
  2.4.2 Desktop Analysis ....................................... 19  
2.5 Should JavaFX and Web co-exist in the controls community  
  ? ............................................................... 20  
  2.5.1 Coexistence vs Web only ............................... 20  
  2.5.2 What developers say .................................... 20  
2.6 Conclusion .................................................. 21  
  2.6.1 What can be said after all .............................. 21  
  2.6.2 The strategy for now .................................... 21  
  2.6.3 What technology to choose from ....................... 21  
2.7 Summary .................................................... 22
# Contents

5.4.1 Model View Controller (MVC) ........................................ 40
5.4.2 Application Model .................................................. 40
5.5 Summary ............................................................... 41

6 The controls framework for Java/JavaFX applications ........... 43
6.1 Introduction ........................................................... 43
6.2 Application Base ....................................................... 43
6.3 LogStatusBar .......................................................... 47
6.4 Log Console ............................................................ 47
6.5 Logging ................................................................. 50
   6.5.1 Log4j and Log4j2 appenders .................................... 50
   6.5.2 ApplicationLogger ................................................. 53
6.6 Finding elements ...................................................... 53
   6.6.1 Finding frame components ....................................... 53
   6.6.2 Finding Stages ..................................................... 54
6.7 Preloader - Splash Screen ........................................... 55
6.8 Multi-stage applications ............................................. 56
6.9 Application task ....................................................... 57
6.10 Charting ............................................................... 61
   6.10.1 Overview .......................................................... 61
   6.10.2 Getting started ................................................... 61
   6.10.3 XYChartPane ...................................................... 62
   6.10.4 Chart Plugins .................................................... 64
   6.10.5 HeatMapChart .................................................... 68
   6.10.6 Displaying large data sets ..................................... 70
6.11 GUI Testing ............................................................. 71
   6.11.1 Testing simple elements with unit tests ..................... 71
   6.11.2 Testing an application with automated robots and unit tests 73
   6.11.3 Simplified testing ............................................... 75
6.12 Summary ............................................................... 76

7 The controls tools for Java/JavaFX applications ................. 77
7.1 Introduction ........................................................... 77
7.2 Plugin Description ..................................................... 77
   7.2.1 Installation .......................................................... 77
   7.2.2 How to access it ................................................... 78
   7.2.3 Creating a new project .......................................... 78
   7.2.4 How to create a new view ....................................... 79
   7.2.5 How to generate missing FXML fields and Handlers .......... 80
7.3 Third party ............................................................ 80
   7.3.1 ControlsFX ........................................................ 80
   7.3.2 Gluon Scene Builder .............................................. 80
7.4 Summary ............................................................... 81

8 Result Analysis ........................................................ 83
8.1 Introduction ........................................................... 83
8.2 Problem description ................................................... 83
8.3 Goals description ..................................................... 84
8.4 Evaluation of the provided solution ................................ 85
List of Figures

2.1 ACCSOFT Project dependencies 2017. ............................................. 6
4.1 A simple example of the MVC process. ........................................... 33
5.1 A diagram describing the MVC pattern. ......................................... 40
6.1 A simple example of an application encapsulated into an ApplicationBase. . . 45
6.2 A small example of the log console in action. ................................... 47
6.3 The preferences of the log console. .................................................. 48
6.4 The search interface of the log console. .......................................... 48
6.5 The log status bar appender in action. ............................................. 50
6.6 The Log console appender in action. ............................................. 50
6.7 An example of a logging message displayed in the status bar and the console. . 50
6.8 The default preloader in action. ................................................... 55
6.9 The different application tasks running in the task tab of the ApplicationBase. . . 57
6.10 Example of chart overlays. ....................................................... 62
6.11 An example of several plugins on a single chart. ............................. 67
6.12 A beam image represented by the HeatMapChart. ......................... 68
6.13 HeatMapChart with CategoryAxis. ............................................. 69
List of Tables

2.1 Comparison between JavaFX and Web. ........................................... 18
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>Accelerator Controls Community</td>
</tr>
<tr>
<td>ACSEC</td>
<td>Accelerator Controls Software Engineers Community</td>
</tr>
<tr>
<td>ACOC</td>
<td>Accelerator Controls Others Community</td>
</tr>
<tr>
<td>CERN</td>
<td>European Organization for Nuclear Research</td>
</tr>
<tr>
<td>LHC</td>
<td>Large Hadron Collider</td>
</tr>
<tr>
<td>LASER</td>
<td>LHC Alarm Services</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>What You See Is What You Get</td>
</tr>
<tr>
<td>TN</td>
<td>Technical Network</td>
</tr>
<tr>
<td>GPN</td>
<td>Global Purpose Network</td>
</tr>
</tbody>
</table>
Definitions

Some details need to be explained before further reading.

In the context of this study one must establish some conventions on how to call the different communities treated in this document.

The community in its whole is, everyone part of the CERN’s accelerator controls environment. This includes many different people having various background profiles, some are full time software engineers, some only develop up to 20 percent of their time, one can think of physicist, operators, machine experts and many others.

To simplify the thought process they will be categorized as follows:

**Controls Community**  This definition refers from this point onward, to every person part of the accelerator control infrastructure, whatever background they have.

**Controls Engineers**  This definition refers, from this point onward, to every full time developer part of the **Controls Community**. Those people have very advanced skills in software development and developing applications is their main work.

**Controls Diverse**  This definition refers, from this point onward, to every person that is not a full time software developer, one can think about physicist and operators for instance. This community is very diverse due to their different capabilities and needs, they have special requirements.
Definitions
Chapter 1

Introduction

1.1 Context and motivation

The CERN controls community has been using Java since its release back in 1995. As its main graphical user interface framework, the controls community has been using Swing, also since it was released. Since then many different events happened in the technology world, Java has evolved and is still evolving. Swing also did, but recently it has become deprecated. It is a very important change for the controls community, investing in it, is no longer an option. Less and less qualified people are willing to use swing for their project, and, the Swing framework is not expected to receive any update from Oracle, as JavaFX has become an entire part of the Java SDK since Java 8.

In this context the controls community is in need of a change, the question is which way to follow, and more importantly, which tools or concept to provide. This community is very peculiar, as one will see ahead it is composed of many different people having many different backgrounds. Not all of them are software engineers, some only work a fraction of their time on graphical user interface creation. The combination of all of those different factors makes the project challenging, as one need to pay a special attention to all the different requirement the community has.

1.2 Goals

The goal of this project is to provide an efficient set of tools and guidelines, to allow the CERN controls community to successfully transition from an aging graphical user interface solution, to a more up-to-date one. Analyzing the needs of the studied population, to understand which technology should be chosen and how to start the transition.

One should not expect a full transition by the end of this project, transitioning more then 600 graphical user interfaces is not feasible within six months. However, by the end of this project, the CERN controls community should understand which technology to invest in, know why this way has been chosen, and finally, have a set of tools to allow its users a smooth and easier transition towards a newer technology.
1.3 Expected contribution

Three artifacts are expected by the end of this project.

The first one is a set of guidelines, those will be used to create a homogeneous project ecosystem, allowing the controls community developers to switch from a project to another and still understand the core structure and its different elements.

The second artifact is the creation of a framework, made using the selected newer technology. The framework is going to be designed to reduce the amount of general code a user has to type, and also, provide new custom made elements that fit the needs of the controls community.

Finally, to ensure a much smoother transition and also a faster development an eclipse plugin should be developed. This plugin is going to provide an easier way for a developer to create projects, UI elements, by providing a template mechanism, that can be enriched by the user himself. The developer would then be able to reuse someone else’s interface components, with just a few clicks. The plugin should take full advantage of the framework and also follow the different guidelines to keep a fully consistent solution.

At the very end the project should be released to the open source community, to allow external developer to take advantage of features they might find relevant

1.4 Structure of the document

This document is organized for the reader to follow the entire decision process of the writer. One will start at the moment when the controls community wants to change but does not know where to begin with. Following the decision process one will see the different steps of the development, from the guidelines to the framework and finally to the plugin description. At the very end a result analysis will follow.
Chapter 2

Choosing a path

2.1 Introduction

The current situation of the controls community is changing. This year, is a year of transition. There is a need to move on from Swing, a deprecated Java graphical user interface framework. It does not represent a good investment choice for many reasons, detailed in this chapter. The other question that arises from such need is, which direction to follow. The following chapter is dedicated to answer those primordial questions.

2.2 Today’s situation

CERN’s accelerator controls environment is a very big and intricate web of software. It is composed of many different projects, using various libraries and frameworks, as one can see in figure 2.1. Each dot represents a project, and each line a project dependency. All of those are necessary, to control the different aspects of the accelerators infrastructure present at CERN [CER]. To have a better understanding, a quick history review is necessary.

2.2.1 The controls community adopts Java

In the mid 90s, a new programming language is released, Java [Orab]. The design goals of Java are very important for the controls community at that time. Java promises: "The development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks". Java is also, from the ground up, fully multi-platform. By then, running an application on any given machine, independently of its architecture, was a new paradigm in the software development world [Orac]. All of those features, were exactly what a complex infrastructure like the accelerator controls needed, hence why it was adopted at the end of the 90s [BCE99].
Choosing a path

2.2.2 Java and Swing

At the early stage of Java, back in 1998, Sun decided to release the second version of Java (JDK 1.2) [Cod]. It was a very anticipated version, as it featured a GUI toolkit, known as "The project swing". This toolkit was a set of components, such as text fields, labels and panels, that any Java developer could use, in order to build its own graphical user interfaces. [Pol00] On the opposite side of the spectrum, web was already present. Back in those days, web was very recent (approximately 10 years old). It was seen, by the controls community, as a technology, that did not reach its maturity yet. It was not suited, nor needed, for day by day operations. Granted, that it was invented by CERN[BCGP10], and, it was starting to be used by the industry, and also, by some CERN developers. Back then, it was used to display information, and did not have the capabilities to be used in very demanding applications. Typically applications requiring complicated elements, interacting with each other, and, with the user.

2.2.3 From the 90s onward

The controls community adopted Java back at the end of the 90s. With the release of Swing there was a will to develop new improved software, among the controls community. Quickly those
Choosing a path

new applications, were used as replacement for the old ones using different languages. Over the 20 years in the CERN accelerator controls infrastructure more than 800 artifacts were built using Java, out of those, more than 600 are using Swing components for their user interface. (Those numbers date back to January 2017, please take a look at figure 2.1). The controls community in the accelerator’s infrastructure is very different from the staff of a software company. Some people, are indeed, full time developers or engineers, but, a big part of people developing software are not. Those are operators or physicist, that develop during part of their working hours (from 10% to 30% of it). Those two communities were defined at the beginning of this dissertation in the definition section. General applications, frameworks and libraries, are developed by controls engineers. Very specific applications, and expert tools, are being developed by controls diverse. The distinction of an application, depend on the knowledge required to develop it. There are applications, a software engineer would not do without user resources, due to a lack of knowledge in a specific area. Software is seen differently among the controls diverse community. This population, consider it as a tool, that they use, to conclude their work. They have specific needs, and problems, that one needs to consider, when studying which technology should be chosen to go ahead.

2.2.4 The end of Swing

The 27th of January 2010, sun was acquired by Oracle[Ora10]. In the early 2010s Swing became deprecated, Oracle announced that a new UI library, called JavaFX, would from now on be the new standard.[Orae] During years, the controls community were developing using Swing, they have the knowledge, they master the tools, but now, they have to adapt to a new technology... But which one ? In the technology world, there are many other great tools, the biggest alternative to Java, would be, web. With all its bells and whistles, it looks great, it has been evolving to become more mature, and has tools that are getting easier to apprehend. It is strongly considered by the controls community.

2.2.5 The present situation

As of today, twenty years after the controls community decided that Java was going to be used, the face of informatics has completely changed. Everything is different, industry has transformed the way we use our machines, the way we program them, the way we build them. Java and web, are still here, and they are still being used, despite their change during all those years. The controls community, still uses Java, and Swing, to develop applications. Some however, are also being developed and used on the web. It is a very important factor one needs to consider. Web is now more mature, and is able to fit in some use cases. Web has evolved in 20 years, one can think of new programming languages, new frameworks and libraries, development made easier over time, etc... It remains a one of a kind and versatile technology. The world talks about eras when it comes to web. Web 1.0, directed at showing simple static HTML pages on a screen. Web 2.0, and its implementation of the ability to contribute between web users. We are currently in the web 3.0, with the development of new search interfaces, such as smart bots. They allow one to talk to their
Choosing a path

devices and receive an answer depending on a context. [Get] Many new eras are still ahead of us, and tons of article will explain that the web is still growing. Web applications are more dynamic than ever, almost any application can be created on the web, but as every technology it has its strength and flaws.

2.2.6 We are left with a choice

Some questions then pops up into the mind of different members of the controls community:

- "Should we switch all of our developers to web ?"
- "Should we have an hybrid approach combining Web and Java/JavaFX ?"

This first chapter, will help understand the risks of different approaches and guide the controls community to a possible solution.

2.3 Evaluating a technology

After considering all of those issues, a list of criteria was established to ease the path selection process for future applications, or, for current ones that needs an upgrade. A word about the context is important, the controls community has some specific needs, which differs from general needs. In this part every aspect will therefore be split into general community (people in general) and specific community (controls community).

The list is as follows:

- **Richness and aesthetics**: Here it is taken into account, how customizable an interface can be, how difficult can it be to find specific elements, etc...
- **Community**: This item reflects the external community one can expect upon choosing a technology. Such community can be of great help during developement.
- **Recruitment**: It concerns the possibility for a recruiter to find a new hire, that has knowledge in the selected area.
- **Learning Curve**: This points covers the time and effort necessary to get up to speed with the chosen technology.
- **Development**: How easy, or difficult, is it, to develop with a specific technology.
- **Deployment**: This item covers, how easily can one release and deploy a product.
- **Mobile and tablet capabilities**: This aspect covers the potential of each technology to be used on a mobile device.
- **Long term stability and support**: One of the most important criteria, the ability for a given product to be used, and maintained, for a very long time.
2.3.1 Richness and aesthetics

This criteria, will help understand more thoroughly, different aspects of interface design. One wants to focus on how consistent the user interface needs to be. Which technology offers the biggest amount of customization, and finally, does the developer community really needs very custom components, if so, how complicated it would be to find/create them. The answer to those points is important for any team developing software.

**JavaFX**

Here is a short list, that sum up, the different aspects of JavaFX for this criteria:

- **Out of the box standard widgets**: JavaFX comes with a very big list of standard components. Components like: panels, combo boxes, menus, and many others. Those components, can already cover controls community’s needs, in term of graphical user interfaces. [Redc]

- **Few 3rd party widget libraries**: There are not many available libraries, or frameworks, to extend the capabilities of the out of the box offer. Libraries, tend to offer custom components, built upon the generic one existing in JavaFX. As of framework, some have good features, some do not. [Rim] The 3rd party availability will be discussed thoroughly in the next chapter of this dissertation.

- **Standardized interactions**: The way JavaFX works, is entirely based on the model view controller pattern. JavaFX allows the possibility to have a very good separation between the interface definition, the way it is controlled, and finally, the data layer. [Hom]

- **Default theme available**: JavaFX comes with a default theme, one does not need to customize the interface, and can use what is provided. It has a relatively modern look and is suitable for office interfaces. [Pot]

- **CSS Customization**: A way to customize the interface in JavaFX, is to use CSS. It allows to customize, for instance, a label’s color, or a cell background in a table. One can achieve a very high level of customization if needed. [JG]

- **GUI builder available**: A tool, called Scene Builder exists. It is suited for a production environment. It allows developers to create complicated interfaces rather quickly, even if it is not their main activity. It is free, open source, and can be integrated in any Java IDE, or work as a stand alone application. [Glud]

- **WebView available**: In some cases, where a developer would need to use an already made web application, or, for some reasons, needs the capabilities offered by the web, he could still use the available WebView component present in the JavaFX library. [Reda]

**Web**

Here is a short list, that sum up, the different aspects of web for this criteria:

- **Standard HTML**: One of the big advantage, is that web uses of course standard HTML. It is able to run in every available browser, and, it is well known by most developers.
Choosing a path

• **Vast variety of libraries and frameworks**: One cannot imagine web development without thinking about all the frameworks and libraries available. Products like AngularJS, Ember, Bootstrap, D3.js, JQuery, etc... Are an inevitable part of today’s web. Development in pure HTML/CSS and JS is extremely rare, every advanced developer would look for an external third party library, that would cover some of his needs. Those tools, allow the developer to have a very advanced level of customization. One can find very specific component required to display very specific data.

• **Library-dependent interactions**: They way each library works is different. D3.js will not work like JQuery, Ember will not be the same as AngularJS. All of those libraries and frameworks have their own calls, functions and interactions. Every library added to the project will help the developer, but will also require some additional work time to get acquainted to it.

• **No default theme available**: HTML code does not come with some kind of styling, the buttons, lists, controls are raw by design. One must add CSS to customize it, many framework provide such default styles, for instance Bootstrap.

• **CSS/JS customization**: Pretty much everything can be achieved using the capabilities of CSS and JS. The level of customization is high, which is a good thing. It allows any developer to build, what ever he has in mind, even if extremely complex.

• **GUI builders not suitable for production**: After surveying current web developers in the controls community, it appears, that web GUI builders (typically WYSIWYG type of tool) are not at all suited for a production environment, creating a lot of undesired code.

**In general**: One can say, to sum up, that JavaFX requires less thinking with its out of the box capabilities and functionality. On the opposite side, Web, gives a wide selection of possible add-ons (all of the different libraries and frameworks available). The selection of add-ons requires an evaluation to be done before choosing a specific one. Also, some sort of "internal standardization" needs to be created. Both options however, allow for customization. Finally, JavaFX allows the developer to take advantage of the web capabilities thanks to the WebView.

**Specifically**: Controls community’s applications usually do not require very specific components. A standard set is required and it is sufficient for most products developed internally. The actual controls community web team, has already defined a set of "internal standard" and also possible add-ons to use. In fact, there is no clear winner in this category. All applications can be done in either of them, and, there is no blocking problem in each technology, however complex the final product might be.
Choosing a path

2.3.2 Recruitment

This part will cover one of the most important aspect of the selection process. Recruitment is a central point at CERN, due to its nature, a big amount of people working at CERN has a limited contract. There is a high turnover of developers on limited duration, usually one to three years. Those developers are recruited right after graduating, or even being still a student. They are called fellows and technical students, for instance. Project developed by those people are going to be handled, afterward, to others, with indefinite contracts. It is primordial to consider new hires when choosing a technology. Usually a project leader will have more trouble in recruiting someone young (freshly graduated) that knows an old technology, than recruiting someone that knows a brand new one. Not every technology is considered deprecated despite their age, languages like Java or C/C++ have been around for a very long time. They are still taught at school and many companies are in need of such developers... Let’s analyze this aspect in more details.

JavaFX  Here is a short list, that sum up, the different aspects of JavaFX for this criteria:

- **Taught at school**: Still today, Java (even if the first release was in 1995) is taught at school. More so, the most usual way to teach it, is through GUI development [FEU]. Until very recently, Swing was taught, but things are changing since the inclusion of JavaFX in the Java SDK, and the deprecation of Swing.

- **Not trendy**: JavaFX is not considered today as being very trendy[Trea], people use it when it fits better.

- **Less attractive**: For young developers, JavaFX is not attractive, due to a lack of industry need. A smaller amount of possible jobs are available at the end of their degree if one would to choose to follow JavaFX.

Web  Here is a short list, that sum up, the different aspects of Web for this criteria:

- **Taught at school**: All the basics of web are taught at school, HTML/CSS and JS are present among any software development related degree. The only downside is that schools do not teach any specific framework or libraries, and mostly, let the student decide what to use.

- **Very trendy**: Web is what all the industry is thinking about right now. Every applications, every startup in the IT sector are thinking and needing a web related product.[Treb]

- **Very attractive**: Since the industry is, all in for the web, of course new grads are attracted to it. It means a higher possibility to have a job.

In general:  Web clearly wins here.
Choosing a path

**Specifically:** The controls community finds it hard to encounter someone that has experience in JavaFX. Fellows and students prefer developing for the web, but agree to work with JavaFX if it is not a 100% of their work. Recruiters tend to select temporary workers based on their needs. Again, at the end of the temporary contract, the long term personal needs to take over the work created.

### 2.3.3 Learning Curve

One needs to take into account that most of the controls engineers are current Java/Swing developers, only a few are Web developers. Another part of the developers (the controls diverse) have specific needs, and are not trained to adapt to new languages. In any adoption of a new technology there is a required time to adapt and master it. This time is called a learning curve, it may be very steep and very difficult to climb, or, relatively easy. This is why this aspect is important, due to the duality of the current controls community.

**JavaFX** Here is the main aspect of JavaFX for this criteria:

- **Shallow curve:** To be ready to work with JavaFX, as a current Java developer, one needs to study the new JavaFX UI library and understand how it works. Also one needs to know the basics of CSS, and optionally, a new language called FXML. An XML file that describes the user interface.[JW14]

**Web** Here is the main aspect of Web for this criteria:

- **Steep curve:** To be ready to work as a web developer many technologies are required. The minimum every developer needs to know is HTML/CSS/JS. Using just those languages will not really help a developer create very complex applications. In fact, he should learn a lot more than that, REST, SASS or LESS, different kinds of frameworks, packages manager, JS transpiling languages (such as TypeScript for instance) and many more. The differences between browsers also need to be understood if not covered by a framework.[Con]

**In general:** As we saw for current Java/Swing developer, JavaFX is much easier to acquire. A Non-Java developer will spend the the biggest part of the time to learn Java, plus the rest discussed previously. It is nonetheless, still easier than the whole web counterpart.

**Specifically:** The majority of the controls community developers are already familiar with swing, however, some (few) do have web development background. It would require a lot of time to get everybody up to speed with web standards, and of course, a lot of assistance and support is going to be required. The controls community does not have an unlimited amount of time and/or resources to help developers switch to web completely.
2.3.4 Development

Development is another important aspect of this study, it is after all, the main activity of a software developer. This is why this section was sub divided into two main points, IDE support and complexity.

2.3.4.1 IDE Support

The current controls community’s web team, has established a set of tool any web developer, of the community, should use to develop for the web. On the JavaFX side, as the language stays the same, one will just need to use the standard Java IDE of the controls community, Eclipse.

JavaFX Here is the main aspect of JavaFX for this criteria:

- **Any Java IDE**: To develop a JavaFX application, a Java IDE is helpful, Eclipse is currently the most used in the controls community.

Web Here is a short list, that sum up, the different aspects of Web for this criteria:

- **IntelliJ (Ultimate)**: According to the current controls community’s web team leader, IntelliJ is, as of today, the tool that is the most convenient for web development.

- **Browser tools**: In order to analyze several aspect of the application, one needs to use the different developer tools available in the browser, in order for instance, to analyze if the web page was rendered correctly, etc...

- **Protractor test framework**: The controls community web team specified internal standards. One of them is the use of AngularJS. In order to test thoroughly, any web application they make using AngularJS, they have decided to use a test framework called protractor.

In general: Two different technologies require two different set of tools. There is no fight nor winner in this aspect.

Specifically: Most of the controls community is currently using Eclipse. In fact, an structure was built around it, a tailored distribution, specific plugins and add-ons, etc... Changing this IDE in the eventuality that all developers must create web applications, would create a considerable amount of work. Developers would have to get acquainted to a new IDE, and, a new structure would have to be put in place for IntelliJ.

2.3.4.2 Complexity

When the talk about complexity came out, the controls community had different opinions, but one thought stood up. Web is appealing, but looks complex. The variety of different languages and
Choosing a path

software one must use during the development, makes developers reticent. Staying with Java and adding only the JavaFX library looks like a much simpler step to take.

JavaFX  Here is a short list, that sum up, the different aspects of JavaFX for this criteria:

- **Java**: Java is the main programming language currently used.
- **Standard java environment**: When one uses JavaFX it is sufficient to use the standard Java development environment, in this case Eclipse.
- **UI library**: Having a new UI library typically helps a developer create his software faster. Once the learning curve has passed the complexity of development is therefore lowered. It is made possible by introducing new languages, with more powerful features, and, newer concepts, that usually reduces the amount of code necessary to realize specific functions.
- **Direct external communications**: The control systems exposes java-based APIs (Remote Methode Incovation, Java Message Service, ...) that a desktop application can call directly.
- **Java debugger**: To debug a Java program, one uses the debugger bundled with an IDE, it is a very simple process.
- **GUI builder available**: The GUI builder (Scene Builder) makes developing interfaces much easier and much faster, than having to programmatically create it.

Web  Here is a short list, that sum up, the different aspects of web for this criteria:

- **Standard Java environment + web environment**: In the event of a switch to the web, the server-side of a controls community application will remain Java. A developer therefore needs to know more languages and tools, thus increasing the development complexity.
- **Range of web libraries**: The amount of web libraries is very big. Each one may have a different structure, and different ways to operate. The more libraries are used, the more complex it becomes to develop an application.
- **Non-direct external communications**: Different communication protocols exists in the web, the problem however is that one cannot call directly Java APIs from JavaScript, one must always pass by a server that is used as a "proxy".
- **Java Debugger + Browser Debugger + HTTP Debugger**: In order to test thoroughly an application, a web developer will have to switch between different debugging tools. Making the debugging process more complicated.
- **No GUI Builder**: To this day there is not a GUI builder that can be used in production. Some WYSIWYG editors do exists, but the code they generate, is not enough for the controls community standards.
Choosing a path

**In general:** Skilled developers are used to the technology and the tools required to do their job. On both sides people will say that the complexity is acceptable. For a non-web developer however going to JavaFX is much easier.

**Specifically:** Out of all developers that are now currently facing this question choosing either road is not easy, the controls diverse already had difficulties to start developing application in Java and Swing. Switching now, to web, would be for them orders of magnitude more complicated than switching to JavaFX. In addition, many of the currently running applications are connected to hardware, using the web difficults the connection process to the unit, due to a non-direct external communication mechanism. One may require, an intermediate middleware java program, to handle the hardware communication, thus creating an extra layer in the process. To sum up going web leads indeed to a more complex development.

### 2.3.5 Deployment

The developer needs to be able to offer an application to the client in the simplest possible way. Web and Java, have a different approach to that operation.

**JavaFX** Here is a short list, that sum up, the different aspects of JavaFX for this criteria:

- **Manual operations required:** When a new version of an application comes out, and the user does not have an automatic upgrade mechanism, he will have to upgrade manually.

- **Security concerns:** Running a desktop application has security issues, a vulnerability may be found in Java allowing someone to break in for instance.

**Web** Here is a short list, that sum up, the different aspects of web for this criteria:

- **Transparency:** When a user is working with a web based application, he does not have to do updates of any kind, the server can be updated transparently.

- **Security concerns:** A web application has security issues like any other web application. One can think of denial of service attacks or injection flaws for instance.

**In general:** From an external point of view the web has a very clear win, the user does not have to do a manual operation to update an application, also, deployments are made simpler due to centralization of the application. (i.e.: updating only the server vs updating all the desktop applications running on each machine).

**Specifically:** Within the controls group an infrastructure was created to allow any developer to deploy its application in a very efficient and simple way, even easier than the web. Using an in-house tool combined with JNLP[Orad]. This method removes different aspect of web based applications, like a dual server setup. In some cases however, a desktop application will have to be manually restarted.
Choosing a path

2.3.6 Mobile and tablet capabilities

With the capabilities and mobility provided by smartphones and tablets, it is important to evaluate if the chosen technology is able to run on such devices. Another point to evaluate is the need for such devices, should controls applications be available on a mobile device? Here is a comparison.

JavaFX Here is a short list, that sum up, the different aspects of JavaFX for this criteria:

- **Some third parties exists to port the code**: in fact a third party product called JavaFXPort allows a developer to make an application run on a mobile device. It is supported by a company called Gluon[Glub]. Before choosing it, one needs to be aware that the company can, at any moment, drop the project. The life expectancy of such product is not assured, due to people using web or native applications for mobile.

- **Non standard solution**: it is not straight out of the box, a third party library is required to port the code.

- **No long term perspectives**: One cannot say if this kind of library are there to stay. With the fast evolution of smart phones, this library might not be up to date all the time and may end up canceled by the community developing it.

Web Here is a short list, that sum up, the different aspects of web for this criteria:

- **Runs everywhere**: it is a fact any smart phone now can render a web page, no matter how complex it is.

- **Should be responsive**[Mar]: if one wants a good experience using an application developed for web, it should be responsive, to guarantee comfort for its user.

In general Any developer would tell you that the bigger the accessibility the better, and in this aspect JavaFX is not the optimal solution, web is.

Specifically: It mostly depends which use case one wants to cover, some specific tasks require the use of mobile applications. In all other use cases however, it is not necessary or even desired, for security reasons for instance.

2.3.7 Long term stability and support

Long term stability and support, is a critical aspect one needs to consider upon thinking about control systems. Let see how both technologies compare.
Choosing a path

JavaFX  Here is a short list, that sum up, the different aspects of JavaFX for this criteria:

- **Official successor of Swing**: Swing is deprecated, JavaFX was announced by Oracle as the successor.[Orae]

- **Integrated in the JDK**: Since the 8th version of the JDK, JavaFX is included. No need to use an external library anymore. Regardless of the future of the SDK, all the changes are backward compatible.

- **No statement for its future from Oracle**: No declaration from Oracle has been made considering the future of JavaFX. The community is therefore divided, some think it will be frozen, some do not[Alm].

Web  Here is a short list, that sum up, the different aspects of web for this criteria:

- **Basic languages are here to stay**: HTML/CSS and JS are not going to be abandoned any time soon.

- **Relatively short life cycle of libraries and frameworks**: Besides standards such as HTML/CSS and JS, any library, framework or tool today, might not be the used in three years. This point of view is based on the passed of web, who has seen several phases, and many technologies created, and barely used.

- **Unstable architecture**: today’s standard web application architecture, heavily reliant on the framework used, might need to be altered or completely re created due to the unstable nature of the web.

In general  It is in fact difficult to predict the future, and this goes for both technologies. Nevertheless, one can analyze that JavaFX will not disappear for years, as it is in the current SDK. Java has a strong backward compatibility, it has been like this since the language’s creation. In the web, life cycles are much shorter, this is proved by the recent evolution of web frameworks and libraries.

Specifically:  The controls community needs stability and very long life cycles. Some applications written at the beginning of Java are still running, maintained and used. The controls community cannot afford to rewrite an application every 5 years or so. At the end JavaFX is simply more reassuring.
Choosing a path

### 2.3.8 Comparison summary

<table>
<thead>
<tr>
<th>Aspects</th>
<th>JavaFX</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richness and aesthetics</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Community</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Recruitment</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Learning curve</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Development</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Adaptability to mobile</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Long term support</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison between JavaFX and Web.

### 2.4 SWOT analysis

#### 2.4.1 Web Analysis

- **Strengths**
  - Runs on any device that has a browser.
  - Can be made accessible inside and outside CERN.
  - Easier to update/upgrade outside TN (CERN’s Technical Network).
  - Easy to find new hires.

- **Weaknesses**
  - Vast majority of the controls community developers without web background.
  - Steep learning curve, big step for a Java developer.
  - Large variety of frameworks/libraries/tools. (seen as a weakness due to the increased complexity in finding a good one and then maintaining them, due to the very large variety of the offer)
  - Development more complex than with pure Java.
  - Short life cycle of frameworks and libraries.

- **Opportunities**
  - Web is trendy, it is unlikely to change.
  - Currently used libraries may become dominant.
  - Web development becomes easier with time.
  - Fast progress in possibilities.
Choosing a path

- If an industry Web standard emerges, it is easier to migrate to it from a Web application than from a JavaFX application.

**Threads**
- Used libraries might be obsolete soon.
- Applications need to be rewritten often, in order to be up to date.
- Many of the controls community developers might oppose the switch.
- High risk of failure when trying to move everybody to the web.

### 2.4.2 Desktop Analysis

**Strengths**
- Easy to learn for current java/swing developers.
- Out of the box functionality cover most of the controls community needs.
- Easier development process.
- GUI Builder suited for production.
- Swing developers keen to switch.
- Better perspectives for long-term stability.
- Small migration risks.

**Weaknesses**
- Much smaller external community than Web.
- Limited number of 3rd party libraries.
- Need to implement some missing features.
- Not appropriate for mobile devices.
- Not appropriate for applications to be used outside CERN.

**Opportunities**
- Options to use JavaFX on Web under development.
- Might still gain momentum.
- it is Open Source.

**Threads**
- Finding new hires is increasingly difficult.
- Community might be shrinking.
- Might be frozen by Oracle in the future.
Choosing a path

2.5 Should JavaFX and Web co-exist in the controls community?

2.5.1 Coexistence vs Web only

There is two scenarios, a complete switch or a much smoother transition. Would not it be a viable way to have both web and JavaFX?

2.5.1.1 Pros and cons

Pros:

- **Choice of the appropriate technology**: One would have the possibility to choose what technology suits best the application, and, the different uses cases, that the developers needs to cover.

- **Cheaper and less risky**: By using a coexisting approach, one does not need to throw away all the work done in the current infrastructure. In fact one can reuse it. Less risks are involved

- **Web not ready for the controls community**: As of today web cannot cover all the use cases required. A service that requires multiple real time subscriptions is today more complicated to do in web than in Java.

- **Safer for gradual migration**: Web will grow and get more mature, this will allow developers to gradually switch their applications from JavaFX to web if needed.

Cons:

- **Split of knowledge**: having both technology at the same time would create a separation between the two kinds of developers.

- **Less flexibility**: if the controls community has two kinds of technology, it will be harder to move developers between teams and project between teams.

- **Potential duplication of effort**: Some applications may require a web client and a desktop version. Many controls-specific GUI components would have to be developed twice, to accommodate the web, and the desktop version, of an application.

2.5.2 What developers say

After meeting with some current developers of the controls community, and asking, what do they think about a switch to web, some interesting information came out. One one side there are some that would like to learn Web, but most do not, as they have no time, and find the web stack too big and complex.
Choosing a path

2.6 Conclusion

2.6.1 What can be said after all?

Web is indeed progressing very fast, it has become very important. However it is still more complex than creating a JavaFX application. The changes in the web industry are fast, often, changes are not backward compatible... Web is for full time software engineers which are not the majority of the controls community developers.

Going to the web today would require a relatively big effort and resources, and also skilled web developers to support all the other developers of the controls community.

As of today, JavaFX suits the controls community much better. It is here to stay but unfortunately will remain a niche product.

2.6.2 The strategy for now

JavaFX has its flaws, a small community and an unclear future. Web is unstable, but perhaps, the situation will change in some years from now. Maybe Oracle will freeze JavaFX, or, perhaps web development will become as simple as creating a desktop application. If the situation changes, the controls community should re-evaluate the selected strategy. As of today JavaFX is preferred for next developments.

2.6.3 What technology to choose from

It clearly depends on the use case:

Use web when

- Application to be accessible in the CERN’s General Public Netowrk (GPN) or outside CERN
- Application to be used on mobile devices
- Application is not meant for accelerator control directly. (does not act on a physical machine)
- Application doesn’t require very complex UI interactions
- One has in his team at least 1-2 experienced Web developers
- One will have resources to rewrite it after relatively few years (according to web technologies life cycles)

Otherwise use JavaFX
Choosing a path

2.7 Summary

As it has been seen, the choice of one path or another is not straightforward. Nevertheless after consideration of all the different factors, a choice has been made. The controls community will adopt JavaFX as their main graphical user interface framework, web will be delegated to very specific interfaces requiring web capabilities.
Chapter 3

What is available for JavaFX

Disclaimer: For the purpose of this chapter "today" has an importance, it refers to the 3rd of February 2017. On that day, this study was conducted, in order to see, which projects are still in use and followed by the community.

3.1 Introduction

Now that JavaFX is the selected platform to develop applications within the controls community, one must establish what useful tools already exists to help developers. Nobody has to reinvent the wheel when it comes to programming, which is why this chapter is relevant.

Many frameworks, libraries and tools will be discussed during this chapter. The point is, to give a brief idea of their use, and, how they could be useful, or not, during the development of a JavaFX application.

All the relevant links are available, for each of the discussed items, if one wants to go further into details.

3.2 Frameworks

Frameworks are a big part of every language, they are used to simplify some functionality or provide new ones. They make the act of coding simpler, giving developers better tools. In this category one will see the different frameworks studied and see if one fits the current environment.

3.2.1 Afterburner.fx

The first framework on the list is a very simple one, it contains only 3 classes. Its purpose is to allow the developer to have everything organized. It forces the use of a special naming convention to create a cohesion between the different parts of the interface. It also allows the user to take advantage of simple dependency injections.[Fow04]

The simplicity of the framework makes it easy for a developer to get acquainted to it, in less than a day.
What is available for JavaFX

Using this framework would prevent the controls community to use their own naming convention and structure. A custom convention might be more suited for their special needs.

The project appears frozen, as the last commit was made the 27th of April 2016, but it important to note that there is nothing new to develop. More information can be found here: [Bie]

3.2.2 APX

This item is also a simple framework. The idea behind it, is to give to the developer a structure to follow. That structure binds together automatically the model, the view and the controller. In fact it is dedicated to the creation of MVC applications. The same pattern enforced by JavaFX. A simple utility file allows the developer to create easily the default MVC structure, with all three elements linked together.

The flaw, in that simple framework, is that it is still a work in progress, and, did not receive any update for more than three months. It does not make it very appealing. The idea behind it however, might be interesting to follow.

The project can be consulted here: [Dav]

3.2.3 Basilisk

Basilisk is an application framework, targeted at mobile and desktops. Its objective is to take advantage of the JavaFXPorts project, (more informations on that project later on this chapter), to bring the same codebase to the desktop and to the mobile platforms. It is inspired by the Griffon framework, in fact many features are shared between the two. Griffon will be discussed later in this chapter.

The project appears alive as the last commit was made the 1st of February of 2017. All the information is available here: [Bas]

3.2.4 EasyBind

This library provides another way to bind elements. It can be seen as a better way to do it, depending on the point of view. It also provides "monadic observable values", which means, that if one value is null it will not be considered as a "null" but as a "value not available". It is very similar to the "optional" type provided by Java.

To sum up, it allows features that are not really necessary, even if so, they could be replicated easily. This framework does not look very complicated, it could be apprehended quickly by a skilled Java developer.

The project appears finished, with the latest commit made on the 14th of January 2015. The homepage of the project is the following: [Mika]
What is available for JavaFX

### 3.2.5 Griffon

As the tag line says, Griffon is the "next generation desktop application framework platform for the JVM". Behind the marketing talk Griffon is inspired by Grails, a web application framework based on the Apache Groovy language. In order to use Griffon, a developer must be fluent in groovy, as it is a dominant part of the development.

One of the biggest flaws of Griffon, is, that it is primarily targeted at the Swing UI toolkit. It can however be used with JavaFX, considering that features are not as integrated as with swing. It is a problem for the controls community, as they want to use JavaFX as the main UI toolkit.

Besides this flaw, Griffon enforces the use of the model, view, controller pattern. It follows the Swing application framework, which also defines the application’s life cycle.

The project is alive as the last commit is from the 3rd of February 2017. All the information is available here [Gri]

### 3.2.6 Jacp

This framework is targeted at the creation of rich clients using JavaFX and an MVC approach. It enforces a specific structure to the projects, with a separation of files according to their types, like CSS, FXML and others. A full messaging mechanism is also provided which helps the communication between components. Jacp also provides tools to help with the creation of multi-threaded applications.

Provided features look interesting, but would require quite a bit of time for a developer to be used to such structure.

The project appears frozen, the latest update was done on the 24th of May 2016. All the relevant information can be found on the following address: [MS]

### 3.2.7 JRebirth

This framework is quite complicated, it provides many powerful features. One the most advertised features is the simplification of multi-threaded applications, and also, the help provided to avoid possible memory leaks. A developer might require a long time to get used to it. The framework specifies a 7 layer architecture, one needs to be acquainted with. The pattern used is called WB-CS-MVC, which stands for Wave Behavior - Command Service - Model View Controller. Another very important feature of JRebirth, is, its objective to create rich internet applications, however, those are not the target of the controls community.

The project is alive, its latest update was made on the 1st of February 2017. All the information is available on their website: [JRe]
What is available for JavaFX

3.2.8 TornadoFX

TornadoFX is built for the Kotlin programming language. Kotlin was created for the JVM, android and the browser. However interesting it might be, the controls community are not using it. The use of TornadoFX is therefore not an option.

Despite the fact that developers are not using Kotlin, it does not change the fact that TornadoFX introduces a lot of interesting features. Code simplifications, support of many GUI patterns like MVP and MVC, REST client with automatic JSON conversion are only a few interesting ideas present in TornadoFX. This framework also supports the use of very popular JavaFX libraries like ControlsFX and JFXtras. (Will be discussed in the next section).

The project is alive, the latest commit dating back to the 2nd of February 2017. All the information is available on their github page: [Sys]

3.3 Libraries

3.3.1 ControlsFX

Sometimes one may feel that some features and components are missing from JavaFX, ControlsFX tries to feel that void. It provides many high quality UI components, such as, a master/detail view or notifications. This library has proven its efficiency, it is used by many and has a big community following it. Many Swing developers interested about JavaFX in the controls community are already looking into it due to its notoriety.

The project is alive and has many contributions, the latest update was made on the 29th of January 2017. More details can be found here: [E[f]

3.3.2 FontAwesomeFX

It is a very popular set of icons available for free. The chances are that any web user already came across some of its icons, without even being aware of it.

A quick tour of the FontAwesome git repository, proves that this set of icon is here to stay. As of today it has 79 contributors and received 48171 stars. All the information is here: [Fon]

FontAwesomeFX brings this set of icons into JavaFX. The project is alive and has been updated for the last time the 31st of January 2017. This library also includes other sets of icons, like Material Design icons or WeatherIcons. Those can be of great help to any developer requiring very specific icons for an application.

All the information on FontAwesomeFX can be found on the following address:

[Detb]

An icon browser is also available right here:

[Detb]

It is important to note that all icons are available in a scalable format (SVG). They can be resized as one sees fit, also, all icons can be styled with CSS. It is definitely recommended for any JavaFX developer.
3.3.3 FXGraphics2D

This library allows a developer to draw graphs, on a canvas node, in JavaFX. It is a free implementation of Java's Graphic2D API ([Oraa])

It has been developed to be used with Orson Charts ([ors]) and JFreeChart ([jfr]), two other charting libraries.

The controls community requires charting and graphs capabilities. Charts are present in many applications and having the possibility to create charts easily is a big plus for a developer.

This library is alive, with the latest update released on the 2nd of February 2017. All the information is available here: [fxg]

The use of this library in the controls community context is recommended.

3.3.4 Gluon Maps

Gluon, the company behind some projects displayed here, like the Gluon Scene Builder (discussed in the next section), also provides a map library. It is very straightforward and allows a developer to include maps in an application.

The project looks relatively alive, its last update dates back to the 16th of November 2016. More information here: [Glua]

3.3.5 GMapsFX

Google Maps for JavaFX, this is the purpose of this library. The controls community may be required to display a map, and eventually, some overlays representing different machines. Using Google maps to do so is therefore a big plus for a developer. The necessity depends on the use case one tries to cover.

The project is alive, its latest update dating back to the 16th of December 2016. More information is available here: [GMa]

3.3.6 JFenixFX

Material design has been around for a relatively short time. This design language first introduced by Google (now Alphabet) in 2014, has seen its popularity grow due to its introduction to Android. Many applications provided by Alphabet like YouTube, Google Maps, Inbox, Google Docs and many other use those design guidelines and principles.

Using JFenixFX, any JavaFX developer can use material design, to bring back into the user a sense of déjà-vu. Any Android user is prepared to use a material design interface, and, is more likely to be pleased to see something he or she is used to.

This new theme provided by JFenixFX can be visually more appealing, but again, it is a subjective point of view. However, it is not really needed for the controls community softwares.

It is easy to use, one just needs to import the library on the Gluon Scene Builder (a project that will be discussed in this chapter).
What is available for JavaFX

The project is alive, the latest update was released the 1st of February 2017. More information available here: [JFe]

3.3.7  JFXtras

This library has the same objective as ControlsFX, it also aims to provide many UI components such as menus, date pickers and etc...

The project is also alive, the latest update dating back to the 27th of December 2016. All the informations can be found here: [JFX]

3.3.8  Medusa

Gauges, basically it is everything this library provides. In the controls community context, those gauges might be very interesting to display: speed, temperatures, time, battery state and many other physical data types.

The project is alive, its latest update being on the 3rd of February 2017. All the information is here: [Med]

3.3.9  RichTextFX

The name of this library says it all. It allows to have rich texts in a JavaFX application. One can think of a custom logging console, displaying an error in a red bold text and a warning in orange and italic. This kind of functionality can be greatly appreciated for specific applications, or may not be required at all.

The project is alive and the latest update was the 27th of January 2017. More information is available here: [Mikb]

3.3.10 SmartCSV

Exporting data to a CSV file, or, reading data from a CSV file, is an operation almost always required in the controls community environment. An example of its use is to export data from a tableview. This simple operation does not come out of the box, so, having a library that covers this functionality is very interesting. This library is definitely recommended.

The project is available on github, but appears frozen. Its latest update dates back from the 6th of October 2016. More details are available here: [Sma]

3.3.11 TestFX

Tests are a very important part of any software development. Any kind of code must be tested, graphical interfaces are no exception. TestFX provides exactly this, a way to test a graphical user interface. This library provides "robots", those can test a software mimicking a user, moving the mouse, clicking on buttons, exactly in the same way anyone would do. It is also capable of taking screenshots of a failed test, to know where exactly it has gone wrong. One of the richest feature
What is available for JavaFX

is the ability to run headlessly by using Monocle ([Mon]). It is definitely required for any JavaFX interface.

The project is alive, it has 20 contributors, and, was updated for the last time the 31st of January 2017. More information is available here: [FXa]

3.3.12 TilesFX

Displaying tiles on a screen, can be of interest to anybody that needs to display many informations, organized on a screen. If the interest of the developer, is for instance, to display the state of 20 machines on a screen, TilesFX may be an easy way to keep everything organized, and displayed properly.

The project appears alive, the latest commit was made on the 3rd of February 2017. All the relevant information is accessible here: [FXb]

3.4 Tools

An important number of tools are available to use with JavaFX. Here is a curated list that will be helpful to every developer.

3.4.1 E(fx)clipse

This item is not only a tool, but a collection of different ones. It perfectly integrates with Eclipse, one just needs to install the different plug-ins offered. Doing so, it enables many specific features, such as, FXML and CSS color coding and auto completion.

It is efficient but not entirely necessary, it is a nice to have rather than a must have.

The project is alive and available for every developer to use. All the required information is available on the following address: [E(f]

3.4.2 Gluon Scene Builder

This tool may be the most important of the list. It allows any developer to create easily an interface, using drag and drop of components. This tools also exports the interface to an FXML file, allowing a developer to create his interface without having to type a single line of code.

This tool is free and open source, it is also backed up by a company called Gluon (that supplies commercial support if one needs so).

It is relevant to state that this tool and the generated FXML code is suited for production level softwares. The use of this tool is not mandatory, but strongly recommended.

The Scene Builder project is alive, its latest release was published on the December 16th, and binaries are available on the Gluon website: [Gluc]
3.4.3 JavaFXPorts

This project is not going to be covered much in this dissertation. The mobile environment is not, at all, the target of the applications run by the controls community. Nevertheless, this project might be helpful for some developers, it allows to port JavaFX code to the mobile world like iOS or Android.

All the information is available here: [Glub]

3.4.4 Scenic View

This tools helps developer understand the current state of an application scene graph. It is similar to opening the developer tools on the browser, there you can adjust different kinds of properties, without having to compile and check your interface all the time. It does not only simplifies adjustments of interfaces but also improves a developer’s efficiency and speed.

The project is alive, there is even a version ready for the JDK 9.0. All the details are available on the following website: [Glue]

3.5 Summary

Out of all the frameworks available for JavaFX none of them has received a very warm welcoming from the controls community. Many features could be extracted and mold into a more "homemade" solution.

The different libraries are very likely to be used, depending on the use case. Some more generic ones like ControlFX or JFXtras will find a place in the controls community local repository for any developer to use.

The tools are strongly recommended, they offer features that are very important for any developer. The only one not needed is JavaFXPorts, developers of the controls community, are not targeting mobile devices with their applications.
Chapter 4

GUI Patterns Comparison

4.1 Introduction

This chapter covers several GUI patterns. Understanding those patterns, will help understand how JavaFX was built, and the concepts at its core. It is important to understand such informations, after that, one will be more aware of how to build an application using JavaFX.

The following text does not follow an historical order, no specific order has been applied. It is based on the work of Martin Fowler [Fow], a well known British software developer, specialized in object-oriented analysis, UML, and, patterns, among other topics. Also for this research a review on the Model-View-* (MV*) family of patterns have been made using the research conducted by Syromiatnikov and Weynsin. [SW14]

In the GUI environment, there is many different architectural patterns one can choose to follow [LK17] The same goes for non GUI oriented software development. One can think of the singleton pattern, the decorator pattern, the iterator pattern and others. Those patterns, are dedicated to a specific software area, they can be behavioral, structural, creational, etc... depending on what they do and how they are used.

In this study the focus will be on different GUI architectures that have been, or are still, in use. The conclusion will be focused on JavaFX.

Here are the different architecture that will be discussed:

- Forms and Controls
- Model View Controller
- VisualWorks Application Model
- Model View Presenter
- Humble View
4.2 Form and Controls

Forms and controls, is not a standard name, it refers to an architecture based on forms. This kind of architecture became popular due to the development of several client-server applications in the 90s. The main aspect of such architecture, is that a form is specific to an application, but, it uses generic controls. Controls are different buttons, sliders, pickers that are common in every user interface. A developer can build its own controls, and, those can be reused in other applications. The form has two main functionality:

- **Screen Layout**: It defines the arrangement of the different controls on the screen, and also, the hierarchical structure between the elements.

- **Form Logic**: It defines the behavior that cannot be easily programmed into the controls themselves. To clarify, operations such as: clicking on a list triggers a change in another textfield.

The form also contains and displays data, almost always, this data comes from another place. It can come from a remote database for instance. An application therefore contains three copies of the data.

- **Screen Data**: This data, is the one present on the different UI controls., It is the one actually seen by the user.

- **Session Data**: This data is the one temporarily stored in memory, until the user saves it or commit it, to a database for instance.

- **Record Data**: This data is the one present in an external structure.

The biggest challenge faced by this approach, is to keep the screen data synchronized with the session data. From this problem originated a solution called data binding.

**Data Binding**: When two elements are bound together, and a change happens to one of them, it is automatically propagated to the other, making both elements synchronized.

Data binding, is achieved using elements called properties. One will bind a control to a value, doing something like this: `textfield.bind(valueProperty)`. In forms and controls, properties are not objects per say, there is not yet the concept of "property object", that one can find in JavaFX. Data binding is very important and a core concept of JavaFX.

Unfortunately, data binding cannot do all the work. A quick example is: "how can one use data binding to register a click on a button ?" The notion of events was back then introduced to cover this situation. From this point on, each control has a list of events it can raise. OnKeyPressed, OnClick, OnTouch, etc... are all different events raised according to a specific interaction. One can compare it, to a remodelation of the observer pattern.

JavaFX also has this functionality built-in. One can stipulate that a specific function should be called when an action happens. It is very common and very useful.
4.3 Model View Controller

Model View Controller might be the most common and most heard of GUI architecture. The core of the MVC pattern, as the name implies, is a clear separation of the model, the view and the controller.

![Diagram of MVC process]

To make it more simple, here is a brief description of each part:

- **Model**: this part of the pattern corresponds to the data of an application.
- **View**: this part of the pattern corresponds to the interface seen by the user.
- **Controller**: this part of the pattern is responsible to read different inputs and figure out how to use them, typically interacting with both, the view, and, the model.

The core concepts of MVC, stipulates that the model should be fully GUI agnostic. Different GUIs can also use the same model. It is important to note that each UI elements has a pair of view-controller. The screen view, also as a whole, has one.
The observer pattern is also present in MVC, the GUI observes the model and if it changes, the GUI reacts accordingly. The controller does not change directly the view, it changes the model, that change, will be propagated through the use of the observer mechanism. It is different from the previous architectures, where data binding would solve this situation.

In pure MVC, the observer mechanism is standard, but there exist also another kind of synchronization, called, flow synchronization. This terms means, that the changes will be propagated to the different UIs, when the user will explicitly save or commit his changes.

There is many similarities between the JavaFX architecture and MVC:

- **Model:** In Java/JavaFX, as in any languages, data exists.
- **View:** A file called FXML, defines the controls, their location on the screen and the different events a control can raise.
- **Controller:** The concept of controller is present in JavaFX. It is a special java class that is linked to the view and that is used by the user.

### 4.4 VisualWorks Application Model

The objective of this pattern is to solve some problems raised by MVC. A quick example, as MVC assumes that the state of a view can be acquired from the model, how can one know which element of a list has been selected?

The first concept introduced is to create property objects. A reminder, properties are used for data binding. The goal is, to wrap another object inside a property object, so that many more values can be bound. It allows for instance, going back to the previous example, to have a selected property on a list view. Having this kind of tool, allows the developer a bigger control over an interface.

The second concept introduced is the application layer, which allows to separate the presentation logic and its state. The different parts of the interface do not look directly at their model or at each other but, they now look at the application layer. One can think of this as a three layer architecture where one has to pass by all intermediate layers to communicate to another one.

Some of the ideas were used to create JavaFX, the concept of property as objects is one of them. Every property in JavaFX is defined by an interface. It the next chapters it will also be described, how we use and recommend the application model.

### 4.5 Model View Presenter

After MVC and forms and controls, model-view-presenter (MVP) wants to combine the best of both architecture.[Pot96] Its aim is to provide the simplicity of forms and controls, and, keep the clear separation of the different MVC elements.

The different controls, in the forms and controls panel, are now called widgets. MVP treats the whole view as a structure of widgets. When the presenter updates the model, it follows through
the same observer mechanism as MVC. There is a strong similarity so far between MVP and the rest, but there is a difference in who does what. In fact two approaches exists:

- **The supervising controller:** this allows the view to handles some logic, and more complicated events are handled by the presenter.

- **The passive view:** this makes the view totally passive, meaning, that it does not handle logic at all. All operations are handled by the presenter. It has a strong advantage when it comes to testing, testing the presenter thoroughly allows to test all critical functions of the interface, without the need to test the interface itself.

Here are the differences between MVP and the other architectures seen so far:

- **Forms and Controls:** The presenter is expected to manipulate the model, with an observer synchronization mechanism, that will propagate the changes to the view.

- **MVC:** MVP uses a supervising controller, handling some more complex events. The widgets are not separated into views and controllers. Views, can be considered as simpler controller without user gesture handling.

- **Application Model:** views may update themselves from a common model.

MVP is not the core pattern of JavaFX, for instance the views in JavaFX, are observing the model directly, which is against MVP.

### 4.6 Passive/Humble View

It is important to go into further details on the passive/humble view. It starts from a very simple point of view. Testing UIs has always been seen, by many, as a difficult process. The objective of this architecture, is to give, to the complicated to test controls, the least amount of operations possible.

It introduces the concept of "humble object". Any object that is difficult to test, should have minimal behavior. By reducing the capabilities of such objects, one minimizes the chances of having undetected errors, raised by the lack of tests.

What can be considered as the presenter, now assumes a much more complex role. It decides, how to react to user events, and handles the data population of all the different UI widgets. All user events and display logic are routed to the presenter. This makes the UI totally passive, and being manipulated by the presenter, the view, no longer needs to have the visibility of the model. Testing the presenter, or the model, allows then to test most of the risks presented by the UI, without testing hard-to-test elements.

JavaFX however is against this architecture by design, the view watches the model directly. To avoid testing issues, libraries such as TestFX can be used.
4.7 JavaFX GUI Architecture

As seen JavaFX takes bits of several GUI architectures. It is important to remember the parts that composes the structure of JavaFX.

- JavaFX uses generic controls, that can be reused in different applications.
- JavaFX takes advantages of data binding, by using object properties.
- The use of the observer mechanism to keep everything synchronized, is necessary.
- Event handlers, should be specified in the view, except for more complicated cases.
- There is a clear separation of different elements. The model, the view, and the controller, have different functions and files.

4.8 Summary

This chapter concludes the first part of this dissertation. This brief description of different graphical user interface pattern, is important in order to understand the different decisions that have been made while creating the final solution. Now that all the theory and decision process has been explained, one can begin the creation of the series of guidelines and rules, the controls community must follow, for JavaFX applications, from this point onward.
Chapter 5

GUI guidelines and structure

5.1 Introduction

In order to begin the elaboration of a solution, thinking about requirements is a necessity. The first item on the list is, the elaboration of guidelines. Those, will explain the project structure, different naming conventions, and finally interface rules a user has to follow.

5.2 Project Structure

A well defined project structure is important for future developers. It helps them to be at ease when looking at someone else’s project. If everybody follows the different conventions, a new comer placed on a specific project, will not be in trouble to understand how the application was designed. Again, it is very important to remember CERN’s organization, many new people come and many others go. There is a big turn over and projects may have someone new every year. This is why having a consistency across projects is a must.

It is important that the structure described on the following lines is tailored to fit the controls community needs and current build style.

To start with, one needs to focus on where a developer should put his code. Every project must have a folder where the code goes, this folder is called "src", it is located at the root of the project and it is used as a source folder.

Under "src" each project has three sub folders (all of them also used as a source folder):

- "src/java": This folder is where all the java classes needs to be.
- "src/test": This folder is optional, and should contain all java based tests your application may require.
- "src/resources": This folder is optional, and should contain all the classes related to external resources.

Before continuing any further, a small introduction needs to be written, on an in house tool called CBNG.
GUI guidelines and structure

In order to build and release a project, related to the CERN accelerator infrastructure, an internal build and release management tool was created. It is named CBNG and stands for Common Build Next Generation. CBNG is the successor of "Common Build", a build tool used widely around the controls community. CBNG is based on Gradle, meant to support the process of developing Accelerator Control Software. It provides automated tasks for compiling, building, testing, managing dependencies, releasing and deploying software. [Sta]

This concludes the small introduction to CBNG.

CBNG requires two different files product.xml and project.properties, located at the root of the project in order to work properly:

- **product.xml**: This file specifies all the dependencies required by the new project. It also defines the current product name, its version, and its location in the accelerator controls SVN tree.

- **project.properties**: In this file, a developer can define properties and give them a value. For instance, one can define jnlp.enabled = true that will have an effect on the project, according to an application’s code.

As for the rest of the possible files and folders, the developers have total freedom to organize them however they see fit for their project.

5.3 Naming Conventions

Many new concepts are being put together to simplify the creation of Java and JavaFX applications within the controls community.

The most important part of this structure is the naming convention. Following those guidelines will help make better use of the tools provided (and detailed in a further chapter).

One may wonder why does the controls community requires a naming convention, a very small example is the following: if a developer has a controller, an FXML and a CSS file, that are following the naming convention, the tools are able to understand the link between them, and, create this link the developer. This, at the end, means less code and quicker development.

5.3.1 The project Names

Every project requires some informations, such as the project name or the main package name.

Four different informations are required upon the creation of a new project:

- **The project name**: it must have the following name structure: <Top Tier SVN Name>-<Project 1st Name>-<Project 2nd Name>-<...>

- **The main package name**: this is the main java package and it should have the following name structure: cern.<Top Tier SVN Name>.<Project 1st Name>-<Project 2nd Name>-<...>
GUI guidelines and structure

- **The launcher name**: this is the entry point of a JavaFX program and it should have the following name structure: `<Project Last Name>App`

- **The SVN location**: this is the SVN location, where the project will be stored. It should have the following name structure: `<Top Tier SVN Name>/<Project 1st Name>/<Project N-1 Name>/<Project Name>`

An example, of a project following the naming convention is, for instance:

- **Project name**: accsoft-sample-program
- **Main package name**: cern.accsoft.sample.program
- **Launcher name**: ProgramApp
- **SVN Location**: accsoft/sample/accsoft-sample-program

*Side note*: what is refered as the, "top tier SVN name", is one of the element that are present in the root of the controls community SVN repository. Those top tier level, can represent projects, accelerators and/or teams, and are pretty much frozen, meaning, it is extremely rare that they happen to change.

### 5.3.2 The FXML view names

This specific item, should not be compared to a view in a model view controller application. JavaFX uses, at it is core, MVC, one will have to create views to define a graphical interface. In the next chapters, when the new framework will be explained, one will understand that the concept of views has been extended, we call it FXML views. For now however, one just needs to know that in this specific case a view is a set of three files, a Java class called the controller, an FXML file and a CSS file.

The naming convention is very simple for such views:

- **The controller**: needs to be named `<view name>Controller.java`
- **The FXML file**: needs to be named `<view name>.fxml`
- **The CSS file**: needs to be named `<view name>.css`

Any JavaFX project developed following those guidelines should also have the controller, the css file and the fxml file in the same package. However, it is left to the developer’s preferences, if the css and fxml files, should be in src/java or in src/resources.

Once those three files have their names following the guidelines, and once they are in the same package, the developed tools are able to interconnect them easily.
5.3.3 The variable names

In this section there are not many rules to follow, only advices. For every FXML specific variable, (e.g. variables declared with the @FXML annotation), it is recommended to use variable names that follows the convention: <Any Name><Element name>. The <Element name> refers to the name of the FXML element one is considering (Button, Label, ProgressBar etc.). If one is referencing a Label, the associated variable name will be helloLabel, the "hello" part is entirely up to the developer.

5.4 Pattern Convention

5.4.1 Model View Controller (MVC)

The MVC pattern refers, to the model-view-controller pattern, and it is the core of JavaFX, as it has been described earlier.

A view is composed of an FXML file and a CSS file. An FXML file contains a description of an interface in an XML-based language, which allows to separate the interface description from the code logic. The controller, is a Java class, with references to the different view elements, and interacts with the model.

All of those files have a specific purpose:

- **The controller**: will be used to communicate with the model, and the view
- **The FXML file**: will be used to describe the interface in an XML format.
- **The CSS file**: will be used to stylize your interface. (Changing the color of a button, the background of a table cell, etc...).

It is important that every application, created from this point on, in the controls community, follows this pattern. The controller should not take on the FXML purpose and vice-versa.

5.4.2 Application Model

In the case of a communication between several controllers, it is strongly recommended to use a shared model between the controllers. The application model has been described in chapter 3, but, as a reminder, here is a small description of its characteristics. When a controller needs
GUI guidelines and structure

to communicate with another part of an application, there should not be any link between them. A controller will instead update the shared model, any other controller watching the different properties of the shared model will then be updated according to its needs. Using the powerful binding and property mechanism, provided by JavaFX, a controller can react automatically to an event happening in another controller, providing that he is watching the property that has been changed.

5.5 Summary

It is important to provide a project structure and different guidelines to be followed. If such rules would not be in place, each team, each developer would be creating projects according to what they see fit. Which would difficult resource transition between one project to another. In order to create a homogeneous ecosystem, consistency is key, this is why such rules have been implemented. Those guidelines and structures are also important, if the user wants to use the different features offered by the provided solution, described in the following chapters. As an example, the solution takes full advantage of the naming convention and the structure convention, to automatically connect the FXML file and its associated controller. It is therefore necessary that any developer, that wants to use the provided solution, applies those rules.
GUI guidelines and structure
Chapter 6

The controls framework for Java/JavaFX applications

6.1 Introduction

The controls community has been using Swing as its main GUI toolkit since it was invented. Many tools and framework have been developed in-house, to help the users create swing applications faster and better. [BPBJ11] However, today, such tools and framework do not exist for JavaFX and must be implemented. This chapter describes the framework created and tailored to the needs of the controls community applications. Besides the different methods aspects of the framework described in this chapter, part of it takes the form of a base application, where the user can attach his own application. It is composed of three important pillars, a status bar, where one can display any relevant message for the user, a log console, that will display all the log messages a developer added in his application, and finally, a task manager, in order for a user, to see what is currently running in the background of his application.

6.2 Application Base

The standard procedure to start a new JavaFX application, is to extend the Application class as shown in the ExampleStdApp.java code snippet below. The easiest way to make use of the facilities provided with this framework is by extending the ApplicationBase class (see ExampleApp.java example). The main difference, in this simple case, is that instead of creating GUI components into the start method, one will have to do this into the createRootNode() method. The start method, has been made final, in order to preserve the proper initialization steps for the frame components. This, implies that one will not have to manually create a new Scene, but only to create the root node of the layout, the Scene will be generated by the framework.

ExampleStdApp.java

```java
import javafx.application.Application;
import javafx.scene.Parent;
```
The controls framework for Java/JavaFX applications

```java
import javafx.scene.Scene;
import javafx.scene.control.TextArea;
import javafx.stage.Stage;

public class ExampleStdApp extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {
        Parent root = new TextArea("Some text here");
        Scene scene = new Scene(root, 800, 600);
        primaryStage.setTitle("Example Application");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

ExampleApp.java

```java
import cern.accsoft.gui.application.ApplicationBase;
import javafx.scene.Parent;
import javafx.scene.control.TextArea;

public class ExampleApp extends ApplicationBase {
    public ExampleApp() {
        super("Example Application", 800, 600);
    }

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    protected Parent createRootNode() throws Exception {
        return new TextArea("Some text here");
    }
}
```

Running these few lines of code will give the user an enriched frame, in which one can plug its content (a simple TextArea in the example shown on the figure 6.1).

The primary Stage of this application will be filled with a MasterDetailPane, (components from the ControlsFX library), in which there is two different nodes in a split pane, with the possibility to collapse the detail pane (bottom one).

44
The controls framework for Java/JavaFX applications

Figure 6.1: A simple example of an application encapsulated into an ApplicationBase.

By default, the detail pane will be collapsed, and will be composed of a tab view. The Console tab contains a LogConsole component, useful to display log messages, whereas the Tasks one is used to show progress and status of the tasks currently running (see ApplicationTask).

Under this pane a LogStatusBar is placed, ready to display text messages, status of running tasks, and, from there, one can also expand/collapse the details view.

The three methods implemented in the ExampleApp.java code snippet, allow the users to set up the application customizing some settings:

- **main(String[] args)**: this is the place where the launch method from JavaFX Application is normally invoked; unless strictly necessary, one should not add any extra initialization code in this method.

  Initialization should be done in the init method (overriding the default, empty, one).

- **ExampleApp()**: if one does not need to specify a title or a desired size (width and height) for the main Stage, one may not need to define a constructor, (the default one will create a window without title that will adapt to its content).

  If otherwise, in order to initialize the constructor, one has to invoke one of the super constructor of ApplicationBase. There are 3 versions available, allowing the user to specify title and sizes of the main Stage of the application or using empty-default values.

- **createRootNode()**: this is the correct place to create and manipulate GUI nodes. As one may have noticed, the start method cannot be overridden as it has been made final, and, usually, that is the place where one initializes its GUI.
The controls framework for Java/JavaFX applications

The `createRootNode` method is executed as part of the `start` method in `ApplicationBase`. Here, one should return the root node to be placed in the central area of the application, it will be then added as the root of the master pane (filling all the available space).

By default the primary `Stage`’s size fits to the scene (that, in turn, is modified according to the size that the user specifies, through the constructor, or through the provided root node). It is then centered on the screen and finally shown. If this is not the behavior desired, it is possible to customize these final steps by overriding the `startApplication` method.

**startApplication example**

```java
import cern.accsoft.gui.application.ApplicationBase;
import javafx.scene.Parent;
import javafx.scene.control.TextArea;
import javafx.stage.Stage;

public class ExampleApp extends ApplicationBase{
    public ExampleApp() {
        super("Example Application", 800, 600);
    }

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    protected Parent createRootNode() throws Exception {
        return new TextArea("Some text here");
    }

    @Override
    protected void startApplication(Stage primaryStage) throws Exception {
        //Default behavior is
        //primaryStage.sizeToScene();
        //primaryStage.centerOnScreen();
        //primaryStage.show();

        //Customized one
        primaryStage.setX(100);
        primaryStage.setY(200);
        primaryStage.setWidth(600);
        primaryStage.setHeight(500);
        primaryStage.show();
    }
}
```
6.3 LogStatusBar

The LogStatusBar component is a custom extension of the Controlsfx StatusBar, that adds a background color to the log messages according to the level of the message. This control is placed at the bottom of a window, and it is used to display various types of application status information. By default the status bar contains a label to display log messages, and a progress bar (see Progressbar) for long running tasks, that appears, when a task is started (see ApplicationTask).

Additional controls / nodes can be placed on the left and right sides of the status bar (see getLeftItems() and getRightItems()).

```java
//Add nodes to the left side
getApplicationPane().getLogStatusBar().getLeftItems().add(new Button("My Button"));
```

6.4 Log Console

This control is by default integrated in the ApplicationBase provided by this framework, but can be used as a standalone component, and, can be placed wherever needed. It will display the log messages, and allow the user to customize a set of preferences and also filter the amount of information shown (see figure 6.2 and 6.3):

- Message level filters.
- Show-hide time.
- Specify the visible depth of the stack traces logged.
- Specify the console buffer size (number of messages).

A dedicated dialog is provided to search for text in the published messages (see figure 6.4). All of those facilities can be accessed either programmatically or by right clicking on the console area.

Figure 6.2: A small example of the log console in action.
The controls framework for Java/JavaFX applications

Figure 6.3: The preferences of the log console.

Figure 6.4: The search interface of the log console.
The controls framework for Java/JavaFX applications

---

**Set stack trace depth**

```java
//Setting the stack trace depth
getApplicationPane().getLogConsole().setStackTraceDepth(4);
```

By default, the console will display the time in which a message was logged (hh:mm:ss), but one can set a custom `DateFormat`. If instead one prefers to modify the time format using a `log4j` appender, with a dedicated `PatternLayout` (e.g. `%ddd MMM HH:mm:ss.SSS`), one will need to disable the default time stamp in order to avoid having it duplicated.

```java
//Setting the date format
getApplicationPane().getLogConsole().setTimeFormat(new SimpleDateFormat("dd-M-yyyy hh:mm:ss - ");
//Disable automatic time stamp
getApplicationPane().getLogConsole().setShowTime(false);
```

By default the console is *collapsed* at startup, but this preference, and also the position of the *divider*, can be modified programmatically:

```java
//Set the console to be visible by default
getApplicationPane().setBottomPaneVisible(true);
//Set the divider position to 70% of the available height (0 -> divider on top, 1-> bottom)
getApplicationPane().setDividerPosition(0.7);
```
The controls framework for Java/JavaFX applications

6.5 Logging

This frame also supplies different ways to display logged messages into the provided console and status bar.

6.5.1 Log4j and Log4j2 appenders

3 different appenders are included for each version of log4j logging framework:

- **LogStatusBarAppender**: will display log messages to the primary stage status bar. The message level will be highlighted by the status bar color (the color will fade after a default interval of 10 seconds, customizable) (See figure 6.5).

  ![Figure 6.5: The log status bar appender in action.](image)

- **LogConsoleAppender**: will display log messages to the primary stage console. The message level will be highlighted by the status bar color (the color will fade after a default interval of 10 seconds, customizable). In this console it is possible to filter by message type, fix the number of visible messages, search for text etc. (See figure 6.6)

  ![Figure 6.6: The Log console appender in action.](image)

- **DualAppender**: will display log messages to both targets, status bar and console. As shown on figure 6.7

  ![Figure 6.7: An example of a logging message displayed in the status bar and the console.](image)
The controls framework for Java/JavaFX applications

In order to configure the appenders, one can use one of the configuration file provided in the following paragraph (depending on which version of log4j one is using):

**log4j.properties**

```properties
log4j.rootLogger=DEBUG, statusBarAppender, consoleAppender, stdout

log4j.appender.statusBarAppender=cern.accsoft.gui.logging.log4j.LogStatusBarAppender
log4j.appender.statusBarAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.statusBarAppender.layout.ConversionPattern=%d{HH:mm:ss} %c{1}: %m
log4j.appender.statusBarAppender.Threshold = INFO

log4j.appender.consoleAppender=cern.accsoft.gui.logging.log4j.LogConsoleAppender
log4j.appender.consoleAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.consoleAppender.layout.ConversionPattern=%d{HH:mm:ss.SSS} [%-10t] %-5p %-10c{1} ==> %m
log4j.appender.consoleAppender.Threshold = DEBUG

# Standard console appender
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{HH:mm:ss.SSS} [%-10t] %-5p %-10c{1} ==> %m%n
log4j.appender.stdout.Threshold = DEBUG
```

**log4j2.xml**

```xml
<Configuration packages="cern.accsoft.gui.logging.log4j2">
  <Appenders>
    <LogStatusBarAppender name="LogStatusBar">
      <PatternLayout pattern="%d{HH:mm:ss} %c{1}: %m"
      alwaysWriteExceptions="false" />
      <ThresholdFilter level="info" />
    </LogStatusBarAppender>
    <LogConsoleAppender name="LogConsole">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%-10t] %-5p %-10c{1} ==> %m"
      alwaysWriteExceptions="false" />
      <ThresholdFilter level="debug" />
    </LogConsoleAppender>
    <Console name="StdOut" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%-10t] %-5p %-10c{1} ==> %m%n" />
      <ThresholdFilter level="debug" />
    </Console>
  </Appenders>
</Configuration>
```
The controls framework for Java/JavaFX applications

```xml
</Appenders>
<Loggers>
  <Root level="debug">
    <AppenderRef ref="LogStatusBar" />
    <AppenderRef ref="LogConsole" />
    <AppenderRef ref="StdOut" />
  </Root>
</Loggers>
</Configuration>

**Important**: note that in order to avoid the duplication of the stack trace, in the console using log4j2, it is necessary to set the alwaysWriteException flag to false when using one of the provided appenders.

**Usage Example**

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Example {
    private static final Logger LOGGER = LoggerFactory.getLogger(Example.class);

    ...

    private void doSomething(){
        ...
        LOGGER.info("Done");
    }
}
```

In case one is willing to use the controls (LogConsole or LogStatusBar) as standalone components, outside the ApplicationBase infrastructure, but at the same time, wants to exploit one of the appenders, one has to manually tell its application which component to consider as a target. Let’s suppose one wants to add only a LogConsole component to a view, and use a standard Logger to log messages there, here is what one should do:

```java
private static final Logger LOGGER = LoggerFactory.getLogger(Example.class);
...
LogConsole myNewConsole = new LogConsole();
ApplicationLogger.setPrimaryLogConsole(myNewConsole);
...
LOGGER.error("a message");
```
6.5.2 ApplicationLogger

This class provides static methods to obtain slf4j loggers, able to publish messages to the console, the status bar or both the components. Through this methods, it is possible to obtain a logger with the default targets (primary stage components) or targeting specific components in secondary stages. In order to specify a stage, different from the main one, one has to provide a Node contained in the Scene one is targeting.

```java
import org.slf4j.Logger;
...

public void usageExample(String msg, Throwable t, Node node) {
    Logger dualLogger = ApplicationLogger.getDualLogger();
    Logger consoleLogger = ApplicationLogger.getConsoleLogger(node);

    // msg to main stage status bar and console
    dualLogger.debug(msg);

    // msg to specific scene’s console
    consoleLogger.debug(msg, t);

    // msg to main frame’s status bar
    ApplicationLogger.getStatusBarLogger().debug(msg);

    // msg to specific scene’s status bar
    ApplicationLogger.getStatusBarLogger(node).debug(msg);

    // with static import
    getStatusBarLogger().debug(msg);
    getConsoleLogger(node).debug(msg);
}
```

6.6 Finding elements

6.6.1 Finding frame components

From the entry point of the application, i.e. the class extending ApplicationBase, one can have access to all the components of the frame, by invoking the getApplicationPane() method:

```java
ApplicationPane appPane = getApplicationPane();
LogStatusBar statusBar = appPane.getLogStatusBar();
LogConsole console = appPane.getLogConsole();
TabPane tabPane = appPane.getTabPane();
...
The controls framework for Java/JavaFX applications

In order to retrieve a node part of the ApplicationPane (status bar, console etc), when one does not have a direct reference to it, one should use the StageManager class, which provides access to the ApplicationPane:

```java
//Finds the ApplicationPane node in the scene of the primary stage
ApplicationPane appPane = StageManager.getApplicationPane();
...
//Finds the ApplicationPane node in the scene of the provided node
ApplicationPane appPane = StageManager.getApplicationPane(node);
...
```

If one needs to retrieve a generic node, when one does not have a direct reference to it, one should use the Node class, which provides some utilities to simplify this task:

```java
Node nodeOfPrimaryStage =
    ApplicationBase.getPrimaryStage().getScene().getRoot();

//Finds the MyNode node in the scene of the primary stage
MyNode node = Nodes.findNode(nodeOfPrimaryStage, MyNode.class);
...
```

6.6.2 Finding Stages

The framework also provides ways for the developer to retrieve the references to a Stage in an application, by using the following static methods:

```java
// Finds the ApplicationBase "main/Primary" Stage:
Stage stage= StageManager.getPrimaryStage();

// Finds the Stage containing the given "plsChooserButton" Node reference. // The returned Stage will be either the PrimaryStage or a child Stage.
Button plsChooserButton = new Button("Open PLS Selector dialog box");
plsChooserButton.setOnAction((actionEvent) -> {
    Stage parentStage= StageManager.getStage(plsChooserButton);
    String selectedPlsCondition = PlsSelector.showDialog(parentStage,
                CernAccelerator.PS.getTimingDomain().toString(), true, TgmUtil.USER,
                true, null);

    System.out.println("Selected PLS Condition [" + selectedPlsCondition + "]");
});
```
6.7 Preloader - Splash Screen

If an application requires to execute some time-consuming initialization tasks at startup (such as hardware or database accesses), the correct place to run them is the "init" method. This method is called immediately after the Application class is loaded and constructed. Once this method is defined in the class extending ApplicationBase, a preloader/splash screen will be displayed during its execution.

It is possible to notify the preloader in the JavaFX standard way, by calling the notifyPreloader (PreloaderNotification) method. The default ProgressNotification, provided by JavaFX allows only to send progress values (0 -> 1, meaning 0% -> 100%) [Orag], in addition to that, it has been extended with the MessageNotification class, allowing to send both String messages and numeric percentage progress.

The figure 6.8, shows how the default splash screen looks like, but again, it is possible to modify its appearance. To do so, one needs to override the default getPreloaderRootNode and return the desired Node to be displayed, instead of the default image.

![Figure 6.8: The default preloader in action.](image)

Example of how to use the included preloader

```java
import cern.accsoft.gui.application.ApplicationBase;
import cern.accsoft.gui.application.ApplicationPreloader;
import javafx.scene.Parent;
import javafx.scene.control.TextArea;

public class ExampleApp extends ApplicationBase{
```
6.8 Multi-stage applications

In case an application needs to display more than a single stage, and hence, one needs to create secondary stages provided with the ApplicationBase frame, (console, status bar etc) one may consider to use the StageManager and ApplicationPane classes. Through the StageManager it is possible to create stages with different attributes, (owner, modality, title) and then one is able to add a Scene containing an ApplicationPane to it.

//Create a secondary Stage using the ApplicationPane layout
The controls framework for Java/JavaFX applications

Stage secondaryStage = StageManager.createStage(ApplicationBase.getPrimaryStage(), Modality.WINDOW_MODAL, "Stage without Application Pane");
VBox userContents = new VBox(new TextArea());
ApplicationPane pane = new ApplicationPane(userContents);
secondaryStage.setScene(new Scene(pane));
secondaryStage.show();

The StageManager provides utilities to retrieve all the stages that are directly or indirectly descending from a specified stage in the owner-own stage hierarchy. This may be useful in case, for example, one wants to propagate specific actions to all windows related to a given stage (e.g. reduce all the windows where the owner is the primary stage). In case one needs to create a stage somewhere else, without using the methods provided by the StageManager, one needs to register it in the StageManager manually (by calling the register(Stage s) method).

The ApplicationPane is an extension of a BorderPane, containing, as described before, a status bar, a console and a task tab inside a panel that can be collapsed. In case one wants to place this component in a custom position on the screen (e.g. the status bar on the top or the console in some other place), one can use this components directly as standard JavaFX controls (LogStatusBar, LogConsole classes) and place them where it is needed.

### 6.9 Application task

![Figure 6.9: The different application tasks running in the task tab of the ApplicationBase.](image)

The ApplicationTask class is an extension of JavaFX Task, that displays its progress in the specified LogStatusBar if a Node is provided through its constructor. If no node is provided the progress will be shown in the primary status bar. The progress of all active tasks can be monitored in the Tasks tab, close to the Console Tab (see figure 6.9). If the target is the primary stage one can create, or, create & start, invoking the two static methods (create and start), specifying a name, a Callable and a Consumer to consume the result of the Callable. The Consumer is executed on JavaFX Application Thread and thus safe for GUI manipulation.

If one uses the standard way to create a task, by extending the ApplicationTask class in this case, one will have access to the Task’s methods, useful to update the status of a process:

- **updateMessage(String msg)**: to update the message property of the thread, that will be displayed below the progress in the Tasks tab.
The controls framework for Java/JavaFX applications

- **updateProgress(double workDone, double max)**: to modify the progress of the bar.
- **isCancelled()**: to check if a task is cancelled.
- ...

**Example of an ApplicationTask #1**

```java
public class TaskExample {
    private static final Logger LOGGER =
            LoggerFactory.getLogger(TaskExample.class);

    public static void main(String[] args) {
        ApplicationTask<String> task = new DoSomethingTask("TaskName");
        task.start();
    }

    private static class DoSomethingTask extends ApplicationTask<String> {
        private static final int COUNT_MAX = 1000;

        private DoSomethingTask(String taskTitle) {
            super(taskTitle);
        }

        @Override
        protected String call() throws Exception {
            updateMessage("Starting the task");
            for (int i = 1; i < COUNT_MAX; i++) {
                updateMessage("Counting " + i + "...");
                updateProgress(i, COUNT_MAX);
                // Modify something in your GUI from the JavaFX thread
                Platform.runLater(DoSomethingTask::updateGui);
                Thread.sleep(30);
            }
            return "Reached " + COUNT_MAX;
        }

        @Override
        protected void cancelled() {
            LOGGER.info("Task has been cancelled");
        }

        @Override
        protected void finished(String result) {
            // GUI nodes manipulation is allowed as we run on JavaFX Thread
            ApplicationPane appPane =
                    Nodes.findNode(ApplicationBase.getPrimaryStage().getScene()).
            
```
The controls framework for Java/JavaFX applications

getRoot(), ApplicationPane.class);
appPane.setBottomPaneVisible(true);
appPane.getLogConsole().info("Task completed");
}

private static void updateGui() {
    //Update GUI components
    ...
}
}

Example of an ApplicationTask #2

public static void main(String[] args) {
    ApplicationTask<String> task = ApplicationTask.create("TaskName", () -> {
        Logger logger = ApplicationLogger.getStatusBarLogger();
        logger.info("Starting the task");
        for (int i = 1; i < COUNT_MAX; i++) {
            logger.info("Counting " + i + "...");
            Thread.sleep(30);
        }
        return "Reached " + COUNT_MAX;
    }, LOGGER::info);

task.start();
}

Example of an ApplicationTask #3

public static void main(String[] args) {
    ApplicationTask<String> task = ApplicationTask.start("TaskName", () -> {
        Logger logger = ApplicationLogger.getStatusBarLogger();
        logger.info("Starting the task");
        for (int i = 1; i < COUNT_MAX; i++) {
            logger.info("Counting " + i + "...");
            Thread.sleep(30);
        }
        return "Reached " + COUNT_MAX;
    }, LOGGER::info);
}
The controls framework for Java/JavaFX applications

6.9.0.1 FxmlView

In order to be able to simplify the most common feature to any JavaFX program, FxmlView was created. In javaFX the first operation someone will think about is: "how can I load my FXML file and associate it to a controller and an CSS file?", previously one would have to do something like this:

```java
@Override
public void start(Stage stage) throws Exception {
    Parent root = FXMLLoader.load(getClass().getResource("fxml_example.fxml"));

    Scene scene = new Scene(root, 300, 275);

    stage.setTitle("FXML Welcome");
    stage.setScene(scene);
    stage.show();
}
```

This example is of course the simplest one, admitting the user does not have very special needs. More so, this example does not provide a way to add a CSS file to our example window. It is so simple, that in fact, it is not at all helpful. What FxmlView aims at, is providing a simple way for a user to interlink the controller, the FXML file, and the CSS file, injecting all the required data, wherever it is needed, and also, provide a way for the user to specify how the controller is defined. To explain, maybe a user has a controller that comes from a spring environment, and does not want to just create a new instance of it, but wants to do something else. He can set his own controller factory and make that happen. By default the controller factory only creates a new instance of a controller. As of now the user simply has to define his views in the following way:

```java
FxmlView mainView = new FxmlView(MainController.class);
```

The user then has access to the controller and the view directly from this object and can use that in his other component. Imagining an example where he wants to add a view to a parent panel he can do the following:

```java
FxmlView mainView = new FxmlView(MainController.class);
AnchorPane a = new AnchorPane();
a.getChildren().add(mainView.getRootNode());

//the user also has access to the view controller by doing the following
MainController controller = mainView.getController();

//the user can then use the controller however he sees fit
controller.getDataFromDb();
```
6.10 Charting

6.10.1 Overview

JavaFX comes with the javafx.scene.chart package, that contains charting components supporting the following types of charts: AreaChart, StackedAreaChart, BarChart, StackedBarChart, LineChart, ScatterChart, BubbleChart and PieChart. Although the package can be used to create sophisticated charts, it lacks a number of built-in features that are crucial for controls GUIs. Examples are possibility to zoom and part the chart content, superposition of different plot types, data annotations, decorations or a logarithmic axis. The standard charts also show performance limitations when exposed to large data sets. These missing features have been implemented in the cern.accsoft.gui.chart package (residing within accsoft-gui-fx product).

6.10.2 Getting started

First, one needs to get familiar with JavaFX charting package e.g. by looking at the Oracle’s JavaFX Charts tutorial and an introduction to JavaFX charts by examples. Those two resources contain a good overview of the charting API, major concepts, styling with CSS and several useful examples. Except PieChart, all other charts included in JavaFX package extend XYChart class i.e. they have X and Y axes and Data points with X and Y coordinates. Both X and Y values can be of any type but typically they would be Numbers or Strings (in case of category axis).
6.10.3 **XYChartPane**

The central class of the cern.accsoft.gui.chart package is XYChartPane. It is a container that can hold one or more instances of XYChart (i.e. LineChart, ScatterChart, BarChart, etc.). The main purpose of XYChartPane is the support to overlay different chart types on top of each other (as on the figure X) and the possibility to add chart plugins (instances of cern.accsoft.gui.chart.XYChartPlugin) which can be either interacting components, e.g. Zoomer or Panner, or passive graphical elements drawn on the chart, e.g. labels with static text, data indicators, etc.

![Figure 6.10: Example of chart overlays.](image)

The code corresponding to the figure 6.10 representing an overlaid chart example is the following:

**Overlaid Charts Sample**

```java
public class ChartPaneSample extends Application {

    @Override
    public void start(Stage stage) {
        stage.setTitle("Overlaid Charts Sample");
        AreaChart<Number, Number> energyChart = new AreaChart<>(createXAxis(), createYAxis());
        energyChart.setTitle("Test Data");
        energyChart.setAnimated(false);
        energyChart.getYAxis().setLabel("Energy");
        energyChart.getYAxis().setSide(Side.LEFT);
    }
}
```

62
The controls framework for Java/JavaFX applications

LineChart<Number, Number> currentChart = new LineChart<>(createXAxis(), createYAxis());
currentChart.setTitle("Current");
currentChart.setAnimated(false);
currentChart.setCreateSymbols(false);
currentChart.getYAxis().setTickLabelFill(Color.GREEN);
currentChart.getYAxis().lookup(".axis-label").setStyle("-fx-text-fill: green;");
currentChart.getYAxis().setLabel("Current");
currentChart.getYAxis().setSide(Side.RIGHT);

ScatterChart<Number, Number> voltageChart = new ScatterChart<>(createXAxis(), createYAxis());
voltageChart.setAnimated(false);
voltageChart.getYAxis().setTickLabelFill(Color.BLUE);
voltageChart.getYAxis().lookup(".axis-label").setStyle("-fx-text-fill: blue;");
voltageChart.getYAxis().setLabel("Voltage");
voltageChart.getYAxis().setSide(Side.RIGHT);

XYChartPane<Number, Number> chartPane = new XYChartPane<>(energyChart);
chartPane.setCommonYAxis(false);
chartPane.getOverlayCharts().addAll(voltageChart, currentChart);
chartPane.getPlugins().add(new CrosshairIndicator<>());
chartPane.getPlugins().add(new DataPointTooltip<>());
chartPane.getPlugins().addAll(new Zoomer(), new Panner());

energyChart.getData().add(new Series<>("Energy", createTestData(x -> x * x)));
currentChart.getData().add(new Series<>("Current", createTestData(x -> 5 + x * x / 2)));
voltageChart.getData().add(new Series<>("Low Voltage", createTestData(x -> 10 + x * x / 4)));
voltageChart.getData().add(new Series<>("High Voltage", createTestData(x -> 15 + x * x / 3)));

Scene scene = new Scene(chartPane, 800, 600);
stage.setScene(scene);
stage.show();

private NumericAxis createYAxis() {
    NumericAxis yAxis = createXAxis();
yAxis.setAutoRangePadding(0.1);
The controls framework for Java/JavaFX applications

```java
yAxis.setAutoRangeRounding(false);
return yAxis;
}

private NumericAxis createXAxis() {
    NumericAxis xAxis = new NumericAxis();
xAxis.setAnimated(false);
xAxis.setForceZeroInRange(false);
return xAxis;
}

private ObservableList<Data<Number, Number>>
createTestData(Function<Double, Double> func) {
    List<Data<Number, Number>> data = new ArrayList<>();
    for (int i = 0; i < 10; i++) {
        data.add(new Data<>(i, func.apply((double) i)));
    }
    return FXCollections.observableArrayList(data);
}

public static void main(String[] args) {
    launch(args);
}
```

XYChartPane allows having single (shared) Y axis or distinct axes per overlaid chart.

Note that in order to draw charts properly on top of each other - some properties of the overlaid charts are modified at runtime.

### 6.10.4 Chart Plugins

Chart plugins represent add-ons to the standard charts, that can be added to the XYChartPane to either interact with chart content or to add decorations/annotations. Below one can find a small example of a XYChartPane with Zoomer and Panner plugins added.

#### Adding plugins to the XYChartPane

```java
XYChartPane<Number, Number> chartPane = ...;
chartPane.getPlugins().addAll(new Zoomer(), new Panner());
```

Currently the package provides the following plugins:

- **ChartOverlay**: allows to add any arbitrary node that will overlay the chart area. A typical usage would be to overlay an instance of Pane (e.g. AnchorPane) containing child nodes at the arbitrary location.
The controls framework for Java/JavaFX applications

• **CrosshairIndicator**: a cross (horizontal and vertical line) following mouse cursor and displaying current coordinates.

• **DataPointTooltip**: a tooltip label, displaying coordinates of the data point hovered by the mouse cursor.

• **Zoomer**: zooms the plot area to the dragged rectangle.

• **Panner**: allows dragging the visible data window with mouse cursor.

• **XValueIndicator** and **YValueIndicator**: a vertical or horizontal line (accordingly) indicating specified X or Y value, with optional text label that can be used to describe the indicated value.

• **XRangeIndicator** and **YRangeIndicator**: a rectangle indicating vertical or horizontal range (accordingly) indicating specified range of X or Y values, with optional text label that can be used to describe the indicated range.

The following example presents all plugins on a single chart pane.

```java
All plugins on a single chart

public class PluginsSample extends ApplicationBase {

    public PluginsSample() {
        super("Plugins Sample", 800, 600);
    }

    @Override
    protected Parent createRootNode() throws Exception {
        NumericAxis xAxis = new NumericAxis();
        xAxis.setLabel("X Values");

        NumericAxis yAxis = new NumericAxis();
        yAxis.setAutoRangePadding(0.1);
        yAxis.setLabel("Y Values");

        LineChart<Number, Number> chart = new LineChart<>(xAxis, yAxis);
        chart.getData().add(new Series<>("Test Data", createTestData()));

        XYChartPane<Number, Number> chartPane = new XYChartPane<>(chart);
        XValueIndicator<Number> internalStop = new XValueIndicator<>(75, "Internal Stop");
        internalStop.setLabelPosition(0.95);
        chartPane.getPlugins().add(internalStop);

        YValueIndicator<Number> yMin = new YValueIndicator<>(-7.5, "MIN");
```
The controls framework for Java/JavaFX applications

```java
yMin.setLabelPosition(0.1);
YValueIndicator<Number> yMax = new YValueIndicator<>(7.5, "MAX");
yMax.setLabelPosition(0.1);
chartPane.getPlugins().addAll(yMin, yMax);
XRangeIndicator<Number> xRange = new XRangeIndicator<>(40, 60, "X Range");
xRange.setLabelVerticalPosition(0.95);
chartPane.getPlugins().add(xRange);

YRangeIndicator<Number> thresholds = new YRangeIndicator<>(-5, 5, "Thresholds");
thresholds.setLabelHorizontalAnchor(HPos.RIGHT);
thresholds.setLabelHorizontalPosition(0.95);
thresholds.setLabelVerticalAnchor(VPos.TOP);
thresholds.setLabelVerticalPosition(0.95);
chartPane.getPlugins().add(thresholds);

Label label = new Label("Label added to the chart pane\n using ChartOverlay");
label.setStyle("-fx-background-color: rgba(255, 127, 80, 0.5)"uellement);
AnchorPane.setLeftAnchor(label, 5.0);
AnchorPane.setTopAnchor(label, 5.0);
chartPane.getPlugins().add(new ChartOverlay<>(OverlayArea.PLOT_AREA,
new AnchorPane(label)));
chartPane.getPlugins().addAll(new Zoomer(), new Panner(), new
CrosshairIndicator<>(), new DataPointTooltip<>());
chartPane.getStylesheets().add(getClass().getResource("plugins-sample.css").toExternalForm());
return chartPane;
}

private ObservableList<Data<Number, Number>> createTestData() {
    Random rnd = new Random(System.currentTimeMillis());
    List<Data<Number, Number>> data = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        data.add(new Data<>(i, (rnd.nextBoolean() ? 1 : -1) * 10 *
rnd.nextDouble()));
    }
    return FXCollections.observableArrayList(data);
}

public static void main(String[] args) {
    launch(args);
}
```
The code above will produce the result one can find in figure 6.11

Figure 6.11: An example of several plugins on a single chart.
6.10.5 HeatMapChart

HeatMapChart is a specialized chart that uses colors to represent data values. An example of the chart is on the figure 6.12 showing a beam image.

![Beam Image](image)

Figure 6.12: A beam image represented by the HeatMapChart.

A snippet of a code used to create this heatMap chart can be found below:

**Usage of HeatMapChart**

```java
NumericAxis xAxis = new NumericAxis();
xAxis.setAnimated(false);
xAxis.setAutoRangeRounding(false);
xAxis.setLabel("X Position");

NumericAxis yAxis = new NumericAxis();
yAxis.setAnimated(false);
yAxis.setAutoRangeRounding(false);
yAxis.setLabel("Y Position");

HeatMapChart<Number, Number> chart = new HeatMapChart<>(xAxis, yAxis);
chart.setTitle("Beam Image");

// readImage() creates a DefaultData class containing X, Y and Z values
chart.setData(readImage());
chart.setLegendVisible(true);
chart.setLegendSide(Side.RIGHT);
```

By default the HeatMapChart uses a "rainbow" colors gradient but this can be changed using colorGradient property. The chart can be also used in combination with CategoryAxis. The example in figure 6.13 illustrates this.
The controls framework for Java/JavaFX applications

Figure 6.13: HeatMapChart with CategoryAxis.

```java
@override
public void start(Stage primaryStage) {
    primaryStage.setTitle("HeatMapChart Category Sample");
    CategoryAxis xAxis = new CategoryAxis();
    xAxis.setLabel("Week Days");
    CategoryAxis yAxis = new CategoryAxis();
    yAxis.setLabel("Teams");
    HeatMapChart<String, String> chart = new HeatMapChart<>(xAxis, yAxis);
    chart.setTitle("Avg #coffees per Person");
    chart.setColorGradient(ColorGradient.BLUE_RED);
    chart.setData(createData());
    chart.setLegendVisible(true);
    chart.setLegendSide(Side.RIGHT);
    Scene scene = new Scene(chart, 800, 600);
    primaryStage.setScene(scene);
    primaryStage.show();
}

private static Data<String, String> createData() {
```
The controls framework for Java/JavaFX applications

String[] team = {"A", "B", "C", "D", "E"};
String[] days = {"Mon", "Tue", "Wed", "Thu", "Fri"};

Random rnd = new Random(System.currentTimeMillis());
double[][] coffees = new double[days.length][team.length];
for (int i = 0; i < days.length; i++) {
    for (int j = 0; j < team.length; j++) {
        coffees[i][j] = 3 * rnd.nextDouble();
    }
}
return new DefaultData<>(days, team, coffees);

6.10.6 Displaying large data sets

When used with large data sets - the JavaFX charts show performance limitations. Below one can find a table with rough measurements of time needed to re-draw a LineChart with and without symbols (the measurements were done on a decent Windows virtual PC):

<table>
<thead>
<tr>
<th>Number of data points</th>
<th>without symbols time (ms)</th>
<th>with symbols time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10</td>
<td>20-30</td>
</tr>
<tr>
<td>2000</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>5000</td>
<td>160</td>
<td>200-250</td>
</tr>
<tr>
<td>7500</td>
<td>350-400</td>
<td>500</td>
</tr>
<tr>
<td>10000</td>
<td>500-600</td>
<td>900-1000</td>
</tr>
</tbody>
</table>

From the table above one can see that displaying up to 2000-3000 data points is handled within a reasonable time but displaying more than that, with an update rate of 1Hz, could efficiently block the FX GUI thread, making the application not responsive. On the other hand, displaying more than a few thousand points on a pane that is, e.g. 1000 pixels wide, means that several data points might be drawn on the same pixel.

To overcome the performance issues, the accsoft-gui-fx package provides DataReducingObservableList that performs data reduction to the specified number of points within the given X data range, i.e. it reduces only the part of initial data set that is currently visible on the chart. This means that while performing a zoom-in, one can see "more details" in the interesting region.

By default DataReducingObservableList uses DefaultDataReducer that is an implementation of Ramer-Douglas-Peucker algorithm [Qvi] - sufficiently fast and giving desired results in majority of cases. If zooming-in is not needed, the DefaultDataReducer can be used directly to filter the data, before it is passed to the chart.

Below one can find an example of how the DataReducingObservableList can be used in practice:

Usage of DataReducingObservableList
NumericAxis xAxis = new NumericAxis();
6.11 GUI Testing

TestFX has been chosen as the foundation for the testing tools created for this framework. In order to be able to use testFX to test a GUI, one will need to add dependencies on the product. To do so one has to use the product.xml file and specify the following:

```xml
<dep product="testfx-core" version="4.0.5-alpha" local="true" />
<dep product="testfx-junit" version="4.0.5-alpha" local="true" />
<dep product="openjfx-monocle" version="1.8.0_20" local="true" />
```

From this point on one will be able to test the project in two different ways:

- Testing simple elements with unit tests
- Testing your whole application with automated robots and unit tests

As one will see further ahead the user can choose to run tests where the computer itself clicks on buttons, this is how testFX runs, but there is a way to run those tests headlessly to allow the user to have tests running on bamboo. (or any equivalent CI software). To do so one just needs to specify the property headless to true, in eclipse one would do so by going to:

- Run Configuration > Arguments > VM arguments and then type: -Dheadless=true

6.11.1 Testing simple elements with unit tests

To be able to tests javaFX components the user will need to create a test class that extends AbstractFxTest, created to take away the complicated part of testFX.

From this point on, the user can create the views that need to be tested, and following this, the user is able to create tests, using the annotation @Test.

The following example represents a class that was called MainControllerTest, supposed to test the MainController of an application.
The controls framework for Java/JavaFX applications

Example test controller

```java
package sampleFX;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static sampleFX.MainController.Status.ERROR;
import static sampleFX.MainController.Status.WARNING;

import org.junit.Test;
import cern.accsoft.gui.fxml.FxmlView;
import cern.accsoft.gui.test.AbstractFxTest;
import javafx.css.PseudoClass;
import javafx.event.ActionEvent;

public class MainControllerTest extends AbstractFxTest {

    MainController mainController;

    public MainControllerTest() {
        FxmlView mainView = new FxmlView(MainController.class);
        mainController = mainView.getController();
    }

    @Test
    public void testCopyMessage() {
        mainController.inputTextField.setText("test");
        mainController.copyButton.fireEvent(new ActionEvent());
        assertEquals(mainController.outputLabel.getText(), "test");
    }

    @Test
    public void testLabelStyleChange() {
        mainController.statusComboBox.setValue(WARNING);
        PseudoClass pseudoClassWarning = PseudoClass.getPseudoClass(WARNING.name());
        PseudoClass pseudoClassError = PseudoClass.getPseudoClass(ERROR.name());
        assertTrue(mainController.outputLabel.getPseudoClassStates().contains(pseudoClassWarning));
        assertFalse(mainController.outputLabel.getPseudoClassStates().contains(pseudoClassError));
    }
}
```
6.11.2 Testing an application with automated robots and unit tests

TestFX created the notion of FxRobots, those robots take control of the mouse and keyboard and are able, to click on elements from an interface.[Gud]

Let’s imagine that the user would like to test if a button is adding an element in a list, he could specify in his code click on button X and check if the list Y contains one more element than before.

All of this is now possible and available for the user via the framework provided.

To be able to use this feature the user needs first to create a new java class that extends AbstractApplicationTest, and that will be used for tests, in an example it has been called MainAppTest as it will be used to test the MainApp class.

It is important to refer in the controller of the test class, the application that will be tested as this last one needs to extend ApplicationBase.

An example is:

```java
public MainAppTest() {
    super(MainApp.class) //here the class needs to extend ApplicationBase
}
```

Once this is done the user can create tests. In each of his tests, different new functions are available such as clickOn, press, write, etc... For more information on how to use robots please refer to: https://github.com/TestFX/TestFX/wiki/Robots-and-Queries

### MainApplicationTest example

```java
/**
 * Copyright (c) 2017 European Organisation for Nuclear Research (CERN),
 * All Rights Reserved.
 */

package sampleFX;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import static sampleFX.MainController.Status.ERROR;
import static sampleFX.MainController.Status.WARNING;
import org.junit.Test;
import cern.accsoft.gui.test.AbstractApplicationTest;

```
import javafx.css.PseudoClass;
import javafx.scene.control.Label;
import javafx.scene.input.KeyCode;

public class MainAppTest extends AbstractApplicationTest {

    public MainAppTest() {
        super(MainApp.class);
    }

    @Test
    public void testCopyMessage() {
        clickOn("#inputTextField");
        write("This is an example");
        clickOn("#copyButton");
        clickOn("#inputTextField");
        write("of testFX");
        clickOn("#copyButton");
        clickOn("#inputTextField");
        write("testing your interface");
        clickOn("#copyButton");
        assertEquals("testing your interface", ((Label) lookup("#outputLabel").query()).getText());
    }

    @Test
    public void testCssChange() {
        clickOn("#inputTextField");
        write("Now i’m just going to change the colors...");
        clickOn("#copyButton");
        clickOn("#statusComboBox");
        type(KeyCode.DOWN, 3);
        clickOn("#inputTextField");
        write("checking colors...");
        PseudoClass pseudoClassWarning = PseudoClass.getPseudoClass(WARNING.name());
        PseudoClass pseudoClassError = PseudoClass.getPseudoClass(ERROR.name());
        assertTrue(((Label) lookup("#outputLabel").query()).getPseudoClassStates().contains(pseudoClassWarning));
        assertFalse(((Label) lookup("#outputLabel").query()).getPseudoClassStates().contains(pseudoClassError));
    }
}
6.11.3 Simplified testing

There is a bonus way to test applications running Java with or without JavaFX, it is what is called FxJUnit4Runner. It allows a user to define specific tests as a JavaFX test, which means, that a user can easily test several aspect of his application without the need to launch the whole javaFX environment, if he does not want to. Typically to save time. Also, it is now much simpler to use, here is an example:

```java
@RunWith(FxJUnit4Runner.class)
public class MainControllerTestWithRunner {

    public MainControllerTestWithRunner() {
    }

    @Test
    public void myTest1() {
        //test number one running with JavaFX environment NOT initialized
    }

    @Test
    @RunInFxThread
    public void myTest2() {
        //test number two running with JavaFX environment initialized
    }
}
```

The annotation @RunInFxThread means that the user requires the initialization of the JavaFX environment before starting his test. If a specific test requires it, it has to be put right before the method. In the case that the whole test class requires the initialization of the JavaFX environment, then the user can write his annotation directly above the class definition. Such as the following:

```java
@RunWith(FxJUnit4Runner.class)
@RunInFxThread
public class MainControllerTestWithRunner {

    public MainControllerTestWithRunner() {
    }

    @Test
    public void myTest1() {
        //test number one running with JavaFX environment initialized
    }
```
The controls framework for Java/JavaFX applications

```java
@Test
public void myTest2() {
    // test number two running with JavaFX environment initialized
}
```

### 6.12 Summary

The whole idea of this framework is to take away the general and tedious code, providing out of the box features required by 80% of controls apps. The second task of this framework is to provide tools to develop quicker an application, giving to the user a skeleton of an application that he can use, having advantages, such as being able to see the current running tasks. Finally, giving the user missing JavaFX functionality has been one of the main focus of this framework.
Chapter 7

The controls tools for Java/JavaFX applications

7.1 Introduction

As specified in the previous chapters the controls community is composed by roughly a hundred developers. Out of those, only 25% are actual software engineer/developer, the other 75% are physicist, machine operators, mathematicians, etc...

Those people (the other 75%) spend roughly 10 to 30% of their time developing graphical user interfaces, for them it is secondary. It is also important to note that 50% of those developers are on an long term contract. The rest (the last 25%) are temporary workers, usually staying up to two years, and primarily composed of students or recent graduates.

The reason to mention this information is that, those 75% of developers represent the biggest part of the target that will be using JavaFX as their main GUI toolkit. Another important part of the population that this plugin is directed to is the new comers. In this category we include people that are new to the language, but also, people that are new to the controls community. The plugin was developed to speed up development, making it easy to be up and running with a project, that respects the different conventions and guidelines of the controls group.

7.2 Plugin Description

The acssoft project manager plugin, is an eclipse plugin to generate quickly "seed" applications, or parts of an application. It is also able to "synchronize" effortlessly an FXML file with its corresponding controller.

7.2.1 Installation

As this plugin is now installed in the eclipse plugin repository of the accelerator community, it is possible for a member of this community to install it like any other plugin on Eclipse:
The controls tools for Java/JavaFX applications

1. On the menu bar, a user clicks on Help > Install from catalog.
2. In the search bar, a user types Accsoft, then presses enter.
3. A plugin called Accsoft FX Plugin will appear.
4. A user selects the latest version and installs it, by clicking on install.
5. On the next window a user confirms the feature(s) to be installed (in this case the Accsoft FX Plugin).
6. A user accept the terms of license agreement.
7. Eclipse will restart and once it is done a user will be able to use the plugin.

7.2.2 How to access it

In a standard version of eclipse with the installed plugin one is able to create a new project or view in 4 different ways:

1. From the new buttons available on the top toolbar.
2. From the menu File > New and then selecting new accsoft project or new accsoft view.
3. From a right click on the project explorer, or, on a project and selecting afterward, the menu File > New, and then, selecting new accsoft project or new accsoft view.
4. From the menu File > New and then selecting others... After that choosing from the list new accsoft project or new accsoft view.

7.2.3 Creating a new project

The first step is to select "New Accsoft Project" from one of the available options detailed in the previous sub-section. Once selected a page will appear.

Different options are now available to the user:

1. The first step is to select the project name, to do so one needs to select the right element, according to one’s need in the first dropdown box, and then, give a name to the project. In this example the project will be named: accsoft-example-project. It is important to notice that the rest of the fields will be changed according to the name of the project. This is why, all the other fields are already written. However the names are merely a suggestion that the user can change.
2. The "Main Package Name" field, represents the name that the primary package will have by default.
3. The "Launcher Name" field, represents the name of the main javaFX class. The entry point of the application.
4. The "SVN Location", represents where the program should be saved in the SVN directory.

5. A set of templates are available to the user, so he can have a premade project, and does not have to start from scratch. In this example, the future project will contain a basic application, with an integration of RBAC and also a couple of tests using the test tools provided by the framework discussed in the previous chapter. RBAC is a plugin developed by the controls community to authenticate members of CERN using specific applications.

6. The final step is to select where the project should be saved.

7. Once everything is filled, the user just has to click on "Finish".

8. A new project will then be created, and appear in the "Package Explorer", under the Eclipse Java Perspective.

After the user clicks on finish, all necessary files will be generated, as well as the product.xml and project.properties file. After the creation of the project, the user simply has to drag and drop his project into the CBNG view, double-click on eclipse to build his project, and, from this point on, the project is fully ready to be used.

7.2.4 How to create a new view

Once the user selects a project, a package, or, a file inside a package, he can create a new view. Views are snippets of codes, that allow the user to have different premade elements, ready for him to use. One can imagine a view that creates a new TableView, with all the elements ready to be linked, or a view that creates a new configuration window. The possibilities of views are endless, as they depend on templates, and templates can be any java code. The first step to create a view, is to select "New Accsoft View" from one of the available option detailed in the "how to access it" sub-section. Once selected, a page will appear. Different options are now available to the user:

1. The "source folder" appears by default, according to where the user clicked. It can be changed.

2. The "parent package name" is the location where the view will be created, in this example as the user right clicked on the "cern.accsoft.example.project" package, and then selected new accsoft view, this field is already filled.

3. The "view name", is the name for the new view that will be created.

4. The "Create a view package?" option, allows the user to create a sub package for his view.

5. The view templates have the same functionality as the project templates.

After the creation of a view, new files will be added to the project, in the specified location. By default a view, like in this example, consist of three different files, a controller, an fxml file and a css file. Since templates can be anything, it will not be a general case, according to what templates are made available in the future.
7.2.5 How to generate missing FXML fields and Handlers

A user creating a view, will most likely be editing an FXML file, to do so, he will add controls, event handlers and other fields. In order to have a reference to those objects, and use them, one needs to modify the controller accordingly. If a user wants to add a new label in an FXML file, and keep a reference to it, the user should do the following:

```xml
<!-- view.fxml -->
<label fx:id="operationLabel">...</...

The following step, is to add a reference to this label in the associated controller:

```java
@FXML
Label operationLabel;
```

Doing so, results in the fact that, once the controller has been initialized, the label object will be injected and will be usable. Users have found this operation as cumbersome, as they have to do this manually all the time. This is the reason behind the feature presented here. Every time a user now edit an FXML file, adding new IDs, or event handlers, he can afterward simply generate automatically, the missing controller code. To do so, a user needs to right click on his controller editor and select "source > Generate Missing FXML Fields and Handlers"

7.3 Third party

In order to develop in the most efficient way, without the need to reinvent the wheel, it has been decided that the developers should have, by default, access to two important third party tools/libraries.

7.3.1 ControlsFX

ControlsFX is considered, by the community, as being the greatest javaFX library currently available. It provides many component, and it is backed up financially by a company, Gluon. It has been decided to incorporate it in the current version of the framework, due to its functionality. The different developers will therefore have a direct access to this library once they start using the framework. Allowing them to use many powerful components without re-creating them.

7.3.2 Gluon Scene Builder

The Gluon Scene Builder is totally optional, but nevertheless, a very important time saver when it comes to creating graphical user interfaces, using javaFX. It is backed up once again by Gluon as seen in previous chapters, and it has been decided to provide it to the different developers of the controls community. Allowing them to develop faster and more intuitively their own applications.
7.4 Summary

As it has been seen previously, the controls community is unique, not everyone works on graphical interfaces a hundred percent of their time. Some developers are operators, mathematicians and etc... The idea behind this tool is to simplify tasks, allowing the developer to create his applications faster. The introduction of templates, will greatly increase the efficiency of the control’s community developers, allowing them to reuse other developer’s template as they see fit. Another great efficiency improvement tool, is the automatic synchronization of the controller and its associated FXML file. Improving coding speed and diminishing errors, related to this specific operation.
The controls tools for Java/JavaFX applications
Chapter 8

Result Analysis

8.1 Introduction

This chapter will be used to present the results of the work achieved during this internship. It will be divided into four sections. The first one, will be a short reminder of the original problem, the second one will contain the goals of the project, the third will contain the obtained results, and finally, the fourth will describe the future of the project.

8.2 Problem description

The problem at hand, as one may have already understood upon reading this dissertation, can be decomposed in three big pillars:

1. A change is required

The first sub problem faced by the controls community, is its aging software. They have been developing using Swing, they have a lot of applications running, but unfortunately, it has been deprecated by Oracle. One may of course continue to use swing, the SDK will not be completely stripped out of it any time soon, but, to be competitive and to have access to a more future proof technology one needs to change. The consensus is that the controls community must look for another newer technology, in this specific case JavaFX or Web.

As we have seen web has advantages such as being more attractive and easier to hire, but changes fast and it is more complex.

2. Specific requirements

The controls community is a very specific environment, as it has already been seen. Many applications are created by mathematicians, physicists, operators and more who are not software engineers. This variety made the task more complicated, as one needs to target many different types of developer. They have different backgrounds, needs, and use different tools on their day to day operations. This developer community was thoroughly studied and analyzed, in order to understand, what could be the tilting point for them to accept the
change and to be happy about it. The proposed solution should therefore be simple to learn and use.

3. **What should be the solution?**

The most fundamental problem to solve after acknowledging the change, and studied the developer community, was, what should be final product. What has to be created to make the transition as smooth, and simple as possible. How can one pave the way for the different future user interfaces, that will be created once this project is launched into production. It has been decided to pursue JavaFx, providing tools, guidelines, and support. This way will be followed at least for the next few years, but at the same time, the controls community will keep an eye on web technologies.

### 8.3 Goals description

- **New projects and components creation**
  
  Find a way, for a user, to be up and running in JavaFX, as fast as possible, with the less amount of complications. The goal was to provide tools, that would let the user have a simple application quickly, so that he could afterward build upon it and create his own solution. It has to be simple and yet very adaptable, as the needs from one application to the other can be very different. It is primordial that the user does not feel completely lost once he has to switch to another GUI framework, especially, when one talks about the different kinds of users, that may not have a software development background.

- **Simpler development**

  Developing using a totally new GUI framework, can be difficult. One of the goals, was to provide a framework that would allow a user to develop faster using less lines of codes. Avoiding to code, in each project, similar functionality. For instance, a code to create a window. Many repetitive tasks can be taken from the hands of the final user, allowing to focus more on the application, rather than having to think about how one should create trivial and repetitive functionality.

- **Specific components**

  JavaFX is great, but there is very specific features missing, a very good example is for instance the heatmap chart. Many user require specific components to be created, and so, one of the goal was to provide those components.

- **Guidelines**

  An important goal to achieve is the creation of rules and guidelines a developer must follow, upon creating a controls software. Enforcing a project structure, naming conventions and rules, is primordial to create a cohesion among all the various project part of the controls community. A homogenous ecosystem allows simpler and faster development.
Result Analysis

• **Documentation**
  It is important not to let the future developers without any information on the newly developed platform. One of the goal is the creation of a web page containing a step-by-step guide on all the developed product and functionalities. From the installation process to the use of a specific method. An introduction to JavaFX is also important, as the user needs to understand the concepts of the new GUI framework. Finally, a reference guide and Javadocs need to be put in place, to have information while developing an application for the controls community.

8.4 Evaluation of the provided solution

To reach the selected goals two different projects were created.

• **Accsoft GUI FX**
  This is a Java/JavaFX framework. For more informations, please refer to chapter 5: The controls framework for Java/JavaFX applications

• **Accsoft Project Manager Plugin**
  This is a plugin that allows a user, to create new projects, new views, according to provided templates, and also, to synchronize automatically a controller and its associated FXML file. For more information, please refer to chapter 6: The controls tools for Java/JavaFX applications

8.4.1 Comparison

Every comparison displayed here will follow the same pattern, before-after.
All comparisons will be realized on the eclipse used by the controls community.

8.4.1.1 Project creation

Creating a new project, for tests or production software, is an initial step for every developer. This task is very repetitive and will be the same for all Java/JavaFX projects.
Right below, a list describes the steps one needs to take to create a project from scratch, with and without, using the newly created eclipse plugin.
It is important to notice that in order for the project to follow the guidelines imposed by the controls group, one needs to have specific files, such as, product.xml and project properties, that do not come out of the box with eclipse.

**Before: without using the accsoft eclipse plugin**
Creating a new project without the provided plugin, requires a lot of different manual steps. Creating new files, adding specific CBNG elements, creating several packages, and more. The final user can achieve a well structured project,
creating every file manually. However, if he desires to take advantages of the internal Accsoft GUI FX Framework, he will additionally have to declare the required dependency in the product.xml, re run the CBNG plugin and create a new main class that takes advantages of the framework.

**After: using the accsoft eclipse plugin** Using the plugin, the previous operation requires much less operation, and can be concluded in a couple of clicks. The result is a well structured project skeleton, that follows the rules and guidelines of the controls community and takes advantages of the accsoft GUI FX framework.

1. Open Eclipse.
2. Click on the "New Accsoft Project" icon in the top toolbar.
3. Type the project name. (Main package name, launcher name and svn location will be written automatically according to the project name, but can still be edited).
4. Select a starting application template, according to the application one wants to develop.
5. Select a project location (or use the default one already provided).
6. Click on finish.
7. Drag and drop the project folder into the CBNG plugin view, double click on it and select the eclipse option.

At the end of these steps, the user has access to a newly create application that already uses the internal framework and follows the guidelines established by the controls group. (Creation of external required files like project.properties and product.xml). Another big advantage in using the new plugin, is the automatic creation of all files the selected template consists of. Imagining a plugin that already implements a master detail pane, after creating his new project, the user would have access to such master detail pane and could already start implementing his application.

The project creation plugin not only makes it easier and faster to create a new project, but also, makes sure a user is respecting the different guidelines created by the controls group.

### 8.4.1.2 View creation

A view, in this context, is a specific set of files that define a component. It can represent any kind of code structure. One can imagine a view as being a panel that contains a table view, some columns, a text field that is used to filter elements of a table and etc... There is no specific way to create a head to head comparison on effort it takes, and time it saves. It is very clear however that once the template for a specific view has been created, the next user will be much faster to implement such view. It is also important to notice that, the more user are contributing to the templates, the more time and energy will be saved, as they will have more and more component they can reuse and take advantage of.
8.4.2 Other improvements

Besides the help provided to create new elements, there is also many new features that were developed, to provide the users an answer to their specific needs.

8.4.2.1 Improved charting

One of the most important feature required to display data in the controls community, is the ability for a specific tool to provide good quality charts. Most of the retrieved information is best displayed using some sort of chart, one can think of a pie chart or a line chart. The truth is, that in this regard, JavaFX fell a little flat. As described previously, it lacks features like zooming, also, it is not best tailored for very large data sets that one can expect while working on a particle accelerator. Due to all of this it was integrated in the framework a new set of classes related to charts.

8.4.2.2 Application Base

As seen in previous chapters the introduction of a new class called ApplicationBase also facilitates the life of the users, by giving them a much simpler way to develop their program. Its goal is to provide out of the box components used by many old swing applications, providing such elements not only saves time, but also, lines of code. It helps a user to start is project giving him a container where his own code will be running. More than that, it gives him tools, that can enhance the development process, such as a way to see logging messages directly in a running application. There is no need to go into much further details on the implementation, as it has been already discussed previously, but rest assured that this component saves a lot of time and effort.

8.4.2.3 Synchronizing controller and FXML file automatically

One of the latest function that has been added in the eclipse plugin, is the ability to synchronize an FXML file with its controller. For who does not understand what it means, it can be describe like this: "Once a user defined his interface in the Gluon Scene Builder for instance, an FXML file will be created. If one needs access to some controls, like a label, one needs to define an fx:id associated with such control. Once it is done, the user needs to create in his controller, a field, that corresponds to this fx:id. Doing so, will allow JavaFX to inject the object into the controller and now one can access it". All of this process is important, and can be seen as tedious. The plugin contains a feature that allows to do this operation directly. If an FXML file defines an fx:id or any kind of event, those will be added automatically into the associated controller, once the user decides it. It is again a new feature, used to save time and take away repetitive manual coding.
8.4.2.4 FxmlView

A class that simply allows to have access, and interlink, all the different components of a view, and a controller. As simple as this class may look like, it is very appreciated by current users, for its usefulness. Allowing the reduction of boilerplate code, and therefore, reducing lines of code.

8.4.2.5 Enhanced testing

The framework also integrates three different ways to test an application as previously seen. This again saves time and effort on the user part. The user no longer has to be preoccupied with setting up a test environment, he just needs to create his tests.

8.5 A sneak peek into the future

8.5.1 In-house development

This framework, the plugin and its templates are indeed going to evolve. More templates can be created, new functions may be needed for the framework. The integration of various CERN specific tools is already planned. In fact this last part will be created in another project that will complement the Accsoft GUI FX framework.

8.5.2 Integration in ControlsFX

The framework Accsoft GUI FX, turned out to be very appreciated by the internal controls community of developers, after discussion it has been decided to submit a request to integrate it, or at least some of its features into the ControlsFX library. As of June 2017 the ControlsFX administrators appeared interested. If for any reason the different feature would not make it to ControlsFX the plan is to release it open source, as part of an independent project.

8.5.3 Integration in E(fx)clipse

The Accsoft Project Manager Plugin is an agglomeration of three different functions as seen previously. One of its features has been submitted to the e(fx)clipse project. It is as of June 2017 an ongoing situation and should be added as soon as it has been validated by the different collaborators of e(fx)clipse. The feature that is being made open source is the automatic synchronization of a controller and its FXML file. Previously to filing an issue on the e(fx)clipse github repository, a post has been made on the e(fx)clipse community forum. This feature has been seen as a great feature by Thomas Schindl the current project leader of e(fx)clipse.

8.5.4 Use in external projects

Many different projects could benefit from this particular set of functions. The easy way to create components and views, could for instance, be applied to the creation of other monitoring
solutions, targeted towards monitoring and controls important infrastructure, like transportation management[ZRC14].

Maintaining such a complex infrastructure is highly difficult, a very strict and rigorous attention to detail is necessary to operate and monitor the accelerator complex. Due to the nature of this task, the multiplication of projects is a real issue, sometimes projects may implement similar features. In this specific case one could analyze the use of different templates and different projects using the framework and provide, to the infrastructure administrator, a report on how efficient is the created code infrastructure. To do this one would need to use the capabilities offered by artificial intelligence, a fleet of agents that would simulate the use of different projects. A similar approach has already been conceptualized, although in a different environment the core features would be similar[ARB15].

The concept of artificial intelligence could also go further, an intelligence that would analyze how users are creating applications or how they use them, could help to optimize not only development but also use. It would have to be aware of the actual development environment and then suggest upon collected data, how different processes could be optimized[PRO10].

8.6 Summary

The point of the provided solution, was to provide a full set of concept and tools, for a developer to create and organize his projects, in a more efficient way. When every pieces come together, one can see how a developer improved by using those tools. When one follows the provided guidelines, uses the new framework and takes advantage of the newly created plugin, the development process is much more fluid, efficient and fast. It was important to be able to provide external developers with such features too. This is why, since the beginning of this project, a strong emphasis has been made on having a fully open-source version by the end of the development process. Two important open-source libraries/framework have accepted to include different parts of the created code, which enforces the idea that those features are indeed seen as valuable. This is an on-going process.
Result Analysis
Chapter 9

Conclusion

9.1 Main findings and contribution

CERN controls group had to make a decision regarding the future of its graphical user interfaces: "What should be the next step in graphical user interfaces for our infrastructure?". The purpose of this dissertation, and the developed solution, was to answer this question, by analyzing the current situation, finding a direction and finally pursue it. It is clear, that this project was not simple, many different aspects are to be considered, this is why it was interesting.

The idea behind the current structure of this dissertation was to take a step back, consider the different routes one can follow, taking into account the established environment. By reflecting first, on which path to go, web or desktop, clearly explaining why a specific decision was made, allowed for a much more efficient development process. The following steps, meaning, the analysis of current tools, definition of guidelines and conventions, allows future developers, to understand ahead of development how they should proceed. They have a structure they can rely on, they will be more flexible, due to a better understanding of different projects they might take care of, again, due to a certain standardization of projects. It was very important to create a smooth way out of past technologies, used to create graphical user interfaces up until now. The Accsoft GUI FX and the Accsoft Project Manager Plugin have this idea at heart. It has been understood that the majority of the users are not software engineers, they may be operators, physicist or others, and they requires simple tools that allows them to develop faster and more efficiently. Specially, it was indeed very important not to make people afraid to switch to a new graphical framework, this was also part of this new approach.

The fact that, very different features created in the context of this project are encouraged by open source projects leaders, means that, what has been developed here is important, not only for the controls community specific developers, but also, for any other JavaFX developer.

All in all, the different solutions created, allows a user to transit from an aging technology to a newer one, much more effortlessly than having to create everything by himself. It also allows, for a user to be much faster, much more efficient when developing code. Finally it allows a user
to create many components that were not available previously in JavaFX, and that now are part of
the provided solution.

The work on the plugin or the framework is not finished yet. Internally at CERN the develop-
ment and improvements on the currently existing solutions will continue. Those two projects are
now at the center of the graphical user interfaces eco system in place. They are made to be used
at least as much as the current Swing solution was. On the other side, the near future open source
access, will allow the project to grow, and hopefully, become a central piece of any JavaFX project
developed out there.

9.2 Future work

This project and all of its components, the guidelines, the framework and the plugin have still
many areas to be pursued.

9.2.1 The guidelines

According to the controls community’s needs the specific guidelines implemented during this
project are not set in stone. They might be modified, new rules may be implemented and some old
rules may be deleted. The changes will depend on the different constraints the controls community
may have.

9.2.2 The framework

The framework is also under development, all the features presented have been implemented, but
others are still been worked on. A sub framework is being developed, it contains a very specific
set of tools, used internally by the CERN controls community. A last sub framework is also being
developed, to allow the developer to take advantage of several external libraries and frameworks,
such as Spring for instance.

9.2.3 The plugin

The plugin can still be improved, first of all more templates need to be created by the community.
Another feature to be implemented is the ability to organize the templates into categories and sub
categories. Another feature that is going to be implemented is the ability to export Java files to a
template, without having to manually export files.

9.2.4 Open Source

A last future development is to release everything open source, the process started but still needs
some time.
References


REFERENCES


[FXa] Test FX. TestFX. https://github.com/TestFX/TestFX.


REFERENCES


REFERENCES

[Pol00] Raul Ramos Pollan. The Java Series. GUI Building with Swing. http://hep.fi.infn.it/JAVA5.pdf, March 2000. The original presentation was a video but unfortunately it is not available, however a PDF transcript was created.


[Pro] Protractor. Protractor, end-to-end testing for AngularJS. http://www.protractortest.org/#!.


[Sma] SmartCSV. SmartCSV. https://github.com/frosch95/SmartCSV.fx.


