ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA182

COMMON STANDARDS FOR JavaFX GUI DEVELOPMENT AND ITS APPLICATION TO THE RENOVATION OF THE CERN BEAM INSTRUMENTATION SOFTWARE PORTAL AND DELIVERY MECHANISM

Iason-Dimitrios Rodis and Athanasios Topaloudis
CERN, Geneva, Switzerland

Abstract

Until recently, Java GUI development in the CERN Beam Instrumentation Group has followed an ad-hoc approach despite several attempts to provide frameworks and coding standards. Triggered by the deprecation of Java's Swing toolkit, the JavaFX toolkit has been adopted for the creation of new GUIs, and is foreseen for future migration of Swing-based GUIs. To increase homogeneity and encourage modular coding of JavaFX GUIs, libraries have been developed to standardise accelerator context selection, provide inter-component GUI communication and optimise data streaming between the control system and modules that make up an expert GUI. This paper describes how this has allowed the use of model-view-controller techniques and naming conventions via Maven archetypes. It also details the modernisation of the software delivery process and subsequent renovation of the software portal. Finally, the paper outlines a vision to extend the principles applied to this Java GUI development for future Python-based developments.

INTRODUCTION

The CERN Beam Instrumentation Group (BI) is responsible for studying, designing, building and maintaining all the instruments that allow the observation of the particle beams and their related parameters in the CERN accelerator complex [1]. The BI Software Section provides software necessary to develop, test, diagnose and maintain such instruments including expert graphical interfaces (GUI) implemented in Java. The main GUI clients are hardware specialists responsible for the instruments, along with a few operators and accelerator physicists who require additional status and control beyond that provided by operational applications. Such clients benefit from signal visualisation, parameter setting, error diagnostics, calibration, data post processing, etc. [2]

Over the past few years, there has been an increase in demand to have platform independent applications that would be able to operate on both the Unix and Windows operating system. Java fulfils this requirement and hence was standardised at CERN for GUI development in the accelerator sector.

Initially, development of such applications was more ad-hoc and based upon the Java Swing toolkit. However, as the complexity and number of applications grew, so did the necessity of standardising the development process through the use of conventions and libraries [3].

ADDRESSING THE PROBLEM

There are many factors that determine the need for new expert GUIs, such as the type of instrument, the acquisition electronics, as well as the type of diagnostics to be performed. Specifications for a new system are therefore often particular to that system and cannot be fulfilled with static GUIs and a fixed list of options.

Functionality reviews on current expert GUIs show that a common, generic and modular JavaFX design is an effective means of making reusable and maintainable GUI components. This standardises and facilitates the development process of a Java project. As a result, despite the different software requirements, the similarity in their overall structure allows the applications to be built based on common JavaFX standards. Additionally, the use of common conventions, graphical components and libraries increase software quality, as well as speeding up the development process.

COMMON STANDARDS

The main goal of the adoption of common standards is to facilitate GUI development and maintenance as well as to increase its homogeneity. In addition, it encourages the use of a modular architecture, with specialised libraries that perform common tasks. Typical examples of such tasks are the: inter-component GUI communication, optimised data streaming between the control system and GUI modules, accelerator context selection (the categorisation of the beams of particles in the accelerators).

Maven Archetypes

In order to help the developers use the new common standards, JavaFX template applications can be created. These provide a clear structure, documentation and demo examples of the aforementioned libraries. Such skeleton applications are generated based on Maven archetypes and aim to establish a common modular architecture for all GUIs being developed.

In short, an Archetype is a Maven project templating toolkit. It is defined as an original pattern or model from which all other things of the same kind are made. Archetypes help authors create Maven project templates for users, and provide users with the means to generate parameterised versions of those project templates [4].

The use of such a powerful templating mechanism facilitates the enforcement of the standards while accelerating GUI development with working, entry-point projects.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA182

Architecture

Following the structure imposed by the Maven archetypes, developers are able to structure their project using a variation of the Model-View-Controller (MVC) Pattern, called Model-View-Presenter (MVP) [5].

The MVC pattern is the oldest and most popular when it comes in GUI modelling. Using the pattern facilitates the division of the domain code, which deals with domain-specific data and business rules, and the presentation code, which deals with the manipulation of user interface widgets. The MVC pattern consists of three components: model, view, and controller. Figure 1 shows a pictorial view of the MVC components and their interactions.

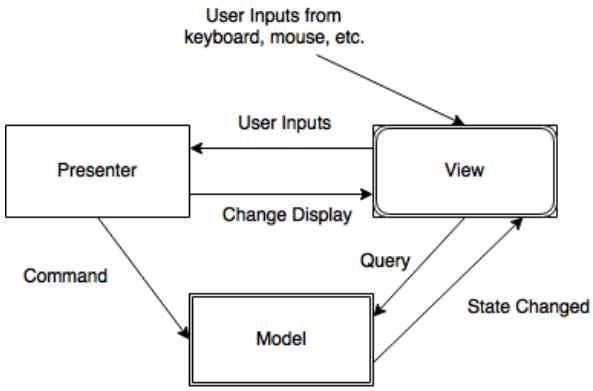

Figure 1: MVC Components and interactions.

In MVC, the model contains the domain objects that describe the real world problems. The view consists of the presentation objects that compose the visual GUI elements. Finally, the controller accepts user inputs and handles them appropriately connecting the view with the model.

In modern GUIs, widget-based architectures are becoming more and more popular. These widgets combine the functions of the view and controller into one. They also support data binding, which helps keep the view and model in sync with fewer lines of code. To benefit from such functionality, we adopted a widget-based architecture in our archetypes. Hence, we defined JavaFX TabViews with their own unique controller and view and, most of the time, their own model. Each widget is able to run as a standalone application, as well as be part of a more complex programme (in its own tab for example).

Dependency Management & Deployment

Currently, we use Maven as our deployment strategy. Maven is a "build management tool" that takes care of tasks required to build a project. Source code compilation, packaging, (pre/post) processing, java classpath managing to name a few.

Furthermore, Maven manages the dependencies of a project, automatically. This is achieved by declaring the needed libraries in its POM[1] file enabling Maven to locate them and their dependencies and import them in its repository system. Maven can provide an arbitrary number of repositories, where it can fetch the declared libraries. It offers developers the possibility to specify local repositories which can be used internally in the team and not be publicly available to others, thus allowing them to control and manage changes. We benefit from such functionality by having local internal repositories cached.

The structure of the POM file and Maven's inheritance of configuration and dependencies facilitates the development process. Since the included libraries also contain a POM file, their dependencies can also be identified and fetched. As a result, it is sufficient to group all common dependencies in the POM file of a "parent" project that will be propagated to all descendants by simply including the parent as their dependency.

We have therefore defined multiple parent projects based on the different use-cases intended for our GUIs. The resulting dependency set is compatible with CERN Control standards [7] as the general GUI components that were developed for JavaFX are used in our archetype programmes (Fig. 2). Another set of dependencies consists of libraries that aim to standardise accelerator context selection (TimingPane), provide inter-component GUI communication (EventBus) and optimise data streaming (CommunicationService) between the control system and GUI modules.

As a result, a new project only needs a POM file that inherits from the parent POM file in order to have all the necessary dependencies, tools and project configurations that compose the development lifecycle. This structure not only standardises the GUI development environment but accelerates and simplifies it as well.

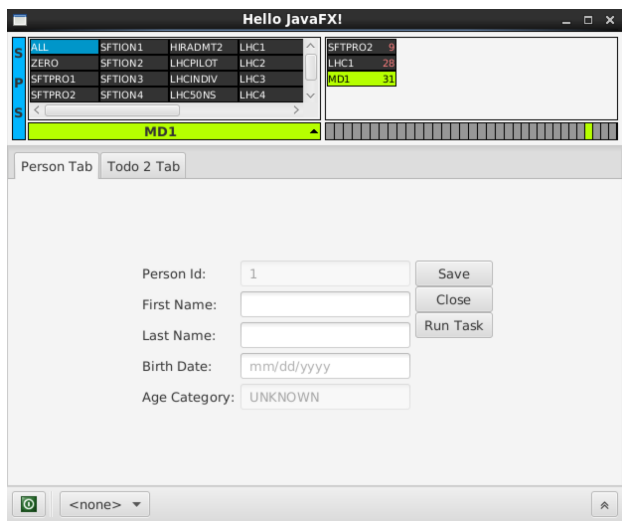

Figure 2: JavaFX Skeleton application based on standard CERN control system GUI components.

Our deployment strategy does not implement any versioning of a given JavaFX application maintaining only a "Release Candidate" and "Operational" version at all times.

[1] Contains information about the project and configuration details used by Maven [6]

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA182

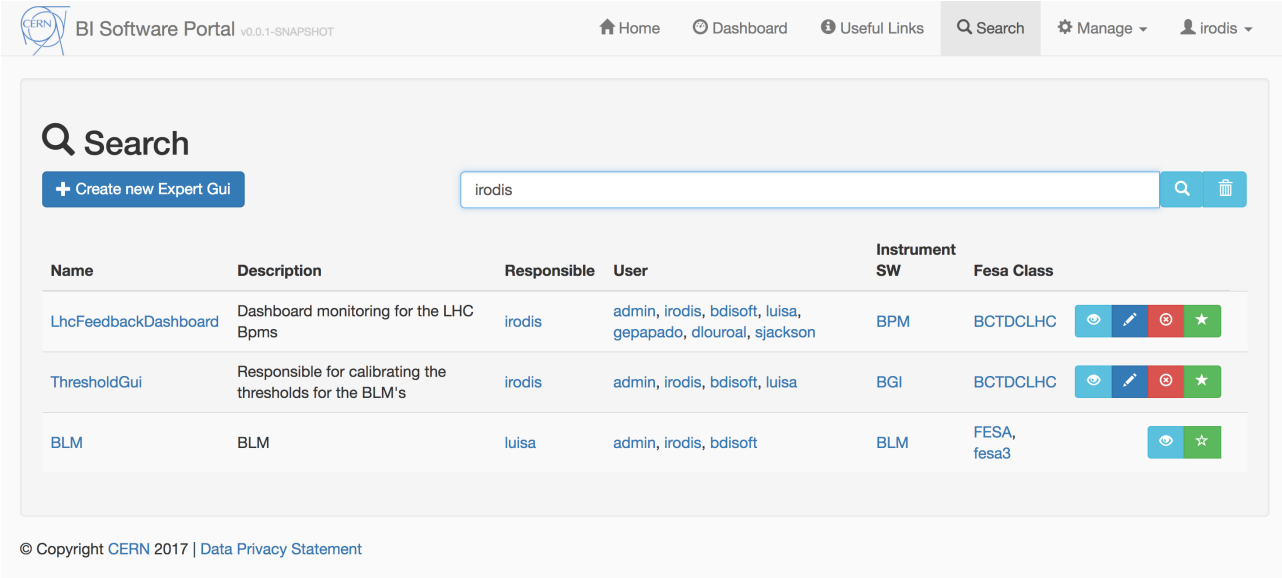

Figure 3: BI Software Portal Search functionality.

Typically, they are deployed by an ANT[2] [8] script and their executable java files can be found under two different folders, each containing all the necessary dependencies specified in the application. Both versions can be launched either through a Web service, or from a dedicated "Application Launcher" [3].

Naming Conventions

In our effort to standardise the dependency management and deployment of our expert GUIs, we found it useful to use the same coding style and naming conventions. By adopting naming conventions, programs become more comprehensible and easier to maintain. They can also give information about the function of the identifier, whether it is a constant, package, or a class, which can be helpful in understanding the code. Lastly, naming conventions enable us to develop and use libraries that can inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time using Java Reflection [9].

RENOVATION OF THE BEAM INSTRUMENTATION SOFTWARE PORTAL

The Beams Instrumentation (BI) Software Portal is a Web application that aims to enhance homogeneity, standardisation and user experience for all beam instrumentation software products. Not only does it comply with new software standards but has been developed in such a way as to benefit from them, in order to automate functionalities such as documentation, GUI launching etc. Its main purpose is to replace old and outdated software tools, combining them into one product, having a single point of reference for different services. Three existing software services that are being upgraded are:

1. ***ApplicationLauncher*** - a software tool that classifies, describes and starts the expert GUIs for beam instrumentation. It uses Java Web-start technology and extends Java's JNLP format. Furthermore, it standardises the configuration of the applications, adds a level of security and makes execution platform independent.
2. ***LHC Instrumentation Documentation Software (LIDS)*** - a Web service that aims to document realtime software that controls instruments developed for beam instrumentation in the CERN accelerator complex.
3. ***Webpages*** - such as:
   - Useful links: a Web service that provides key functionalities such as:
     - The launch of the "Navigator" for deployed real-time FESA (Front-End Software Architecture) classes running the instruments of the accelerator complex [10].
     - List of widely used links.
   - Sharepoint page: whose major functionalities are:
     - Housing a document server for papers, meeting notes, images, etc.
     - Housing the Wiki for developers.
     - Maintaining a list of widely used links.

Architecture

The software portal Web application uses modern Web development standards. With the extensive experience and expertise available for development in Java, a Java-based approach was considered as the most suitable.

The technologies used for the application are based on the JHipster [11] stack. JHipster is a development platform that generates, develops and deploys Spring Boot and Angular Web applications. The goal is the rapid generation of a complete and modern Web application unifying:

- A high-performance and robust Java stack on the server side with Spring Boot.

[2] Software tool for automating software build processes

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA182

- A sleek, modern, mobile-friendly front-end with AngularJS and Bootstrap.
- A powerful workflow to build an application with Yeoman, Webpack/Gulp and Maven.

Functionality

The main goal of the BI Software Portal is to unify all the software tools and documentation produced into one, user-friendly application. Integrating CERN's authentication services enables effortless user identification and classification. Therefore, clients varying from hardware experts and operators to developers, are able to login with their CERN account and access the multiple expert GUIs, documentation, wiki and useful links. Additionally, they have the flexibility to run search queries in order to retrieve information they are seeking. For that purpose, the Portal is using ElasticSearch [12], a search and analytics engine, that allows searching and filtering on multiple fields such as the name of the expert GUI, its developer, the instrument, or the FESA class (Fig. 3).

Moreover, the Portal provides personalisation functionality through a dashboard page, which allows users to customise it according to their needs. They are able to select expert GUIs as favourites, and have a list of their most recently visited applications (Fig. 4).

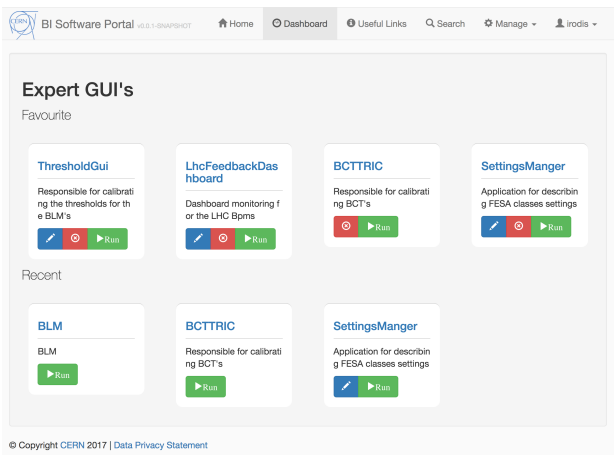

Figure 4: Dashboard personalisation.

Finally, the BI Portal allows the developers to deploy and configure their Java applications. Following the aforementioned standards and naming conventions, once an expert GUI is created, the Web application can automatically retrieve details and add them to the application, such as the links of the remote repository location, the issue management location, and the README file.

Availability

It is essential that the expert GUIs are constantly available for the hardware specialists to support operations. Thus, an "offline mode" will be provided, in case of server failure. The Web application server will frequently cache the .jnlp files, responsible for running the applications, to the CERN network file system (NFS), a guaranteed service. The offline mode will then be able to continue launching the applications until the issue with the online service is identified and resolved.

CONCLUSION AND FUTURE WORK

Although analysing and improving the quality of software is essential, this is often time consuming. In order to efficiently deal with any issues, the software design needs to be kept backwards-compatible and up to date. The programs also need to be structured according to CERN common standards, to ensure they are always maintainable, readable and modular.

In this paper, we have provided a brief overview of our common development standards and the logic for adopting them. We have also presented how their application was essential for the renovation of the CERN Beam Instrumentation Software Portal and Delivery Mechanism.

Our future work will focus on the optimisation of the architecture's performance and the extension of the current principles applied to the Java GUI development for future Python-based developments. In addition, we aim to provide GUI editing functionality that will give the opportunity to create new GUIs based on the components of existing ones, exploiting the modular architecture imposed by our archetypes.

REFERENCES

[1] https://be-dep-bi.web.cern.ch/

[2] S. Bart Pedersen, S. Bozyigit, and S. Jackson, "Java Expert GUI Framework for CERN's Beam Instrumentation Systems", in *Proc. ICALEPCS'11*, Grenoble, France, paper WEPKS027.

[3] P. Karlsson and S. Jackson, "The introduction of hierarchical structure and application security to Java Web Start Development", in *Proc. ICALEPCS'05*, Geneva Switzerland, paper 04_008.

[4] Maven: https://maven.apache.org/guides/introduction/introduction-to-archetypes.html

[5] Kishori Sharan, "Learn JavaFX 8: Building User Experience and Interfaces with Java 8", pp.420-422.

[6] POM, https://maven.apache.org/pom.html

[7] G. Kruk, O. Alves, L. Molinari, and E. Roux, "Best Practices for Efficient Development of JavaFX Applications", presented at ICALEPCS'17, Barcelona, Spain, paper THAPL02.

[8] ANT, http://ant.apache.org

[9] Java Reflection, https://docs.oracle.com/javase/tutorial/reflect/index.html

[10] M. Arruat *et al.*, "Front-End Software Architecture", in *Proc. ICALEPCS'07*, Knoxville, TN, USA, paper WOPA04.

[11] JHipster, http://www.jhipster.tech

[12] Elasticsearch, https://www.elastic.co/products/elasticsearch