ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA173

GRAPHICAL USER INTERFACE PROGRAMMING CHALLENGES MOVING BEYOND JAVA SWING AND JavaFX

S. Bart Pedersen, S. Jackson, CERN, Geneva, Switzerland

Abstract

Since Oracle has decided [1] to stop supporting Java Swing and JavaFX in 2018, replacing Java as our language for the development of graphical user interfaces has posed numerous challenges. Many programmers in the CERN Accelerator Sector will have to adapt and to be re-trained in whatever the next technology will be. Performance of the new GUIs will have to be at least as fast as the existing ones. In addition, programming environment, code versioning, repositories, dependencies and documentation need to be considered as well when choosing the new technology.

First, this paper provides an overview of the research done by the software section of the CERN Beam Instrumentation Group related to comparing GUI languages and explains the reasons for selecting PyQt as a possible future technology. Secondly, the basis for starting a project in PyQt is defined as a guideline for programmers and providing recommendations for the Python [2] environment to be supported.

INTRODUCTION

The software section of the CERN Beam Instrumentation Group (BI) has the mandate to implement real-time servers in C++ that control and monitor instruments developed for beam diagnostics located throughout the CERN accelerator complex. These servers are designed and implemented using an in-house software framework called FESA (Front-End Software Architecture) [3].

The section is also mandated to provide expert graphical user interfaces (GUI) which until now have been developed almost exclusively in Java. These GUIs allow hardware experts easy access to their equipment for parameter setting, signal visualization, error diagnostics, calibration, data post processing and so on. This relies on the underlying, low-level software architecture, middleware (ex: Communication MiddleWare CMW [4]) and the various Java component libraries.

Due to the decision from Oracle to stop supporting Java Swing and JavaFX in 2018, alternative software platforms are currently being tested to replace Java for writing new expert GUI applications. This gives us the opportunity to plan for a good technological solution from a long-term perspective, addressing new needs highlighted by the GUI users, and the limitations identified in the current Java-based solutions.

In defining what a good graphical interface should be, it is important to differentiate between the choice of a programming technology versus the functionalities and performance of the resulting GUI. The user should be presented with features similar to what is available with the current Java implementations, whilst providing improved performance if possible. A comparison of different languages and a list of mandatory graphical functionalities are essential to make a fair analysis and the right final choice.

GUI USAGE AND EXPECTED GRAPHICAL PERFORMANCES

Context and Usage

The main purpose of an expert GUI is to give access to the hardware of any beam instrumentation system through a FESA interface. The GUI is mainly targeted to hardware or software specialists although these GUIs are sometimes also used for machine operation.

Most expert applications are written in Java Swing and more recently JavaFX with a few applications developed with Qt (C++ and PyQt [5]). The applications only run on the non-public technical network (TN) at CERN and, thanks to Java, they execute under both the Windows and Linux operating systems with the only difference being the graphical rendering performance. Some experts have expressed the need to run expert applications on other platforms (mobile telephones, tablets...) and from the CERN general public network (GPN). As it is, most expert GUIs are only connected to FESA devices, but this could be extended to direct access to low-level hardware in the future, which means a direct connection to the hardware driver. A handful of applications are already communicating directly to the hardware through TCP/IP and UDP using in-house protocols, but these applications are rare. Interfaces to retrieve data both from the logging database and from the LHC post-mortem data [6] files are also available. Figure 1 shows all the different communication links between an expert GUI or a Python script and the hardware (HW).

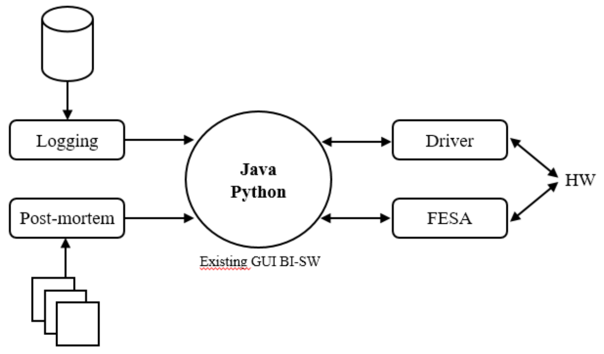

Figure 1: Snapshot of the actual links between the expert programs, the hardware and the databases.

Type of GUIs?

It is important to differentiate four types of expert GUIs:

1. Single driver, device or FESA class interface (with get/set and subscribe modes) giving real-time data snapshots.
2. Fixed display to show read-only (acquisition) data.
3. Advanced application either to combine the data, status and setting from different FESA devices, or to do data manipulation before displaying.
4. Other types of "non-FESA" applications i.e. using direct TCP/IP or UDP connections to hardware.

Performances and Data Rendering?

An application should allow the user to be authenticated and then to select the desired device (instrument), a particular timing event and to provide the communication interface to this/these server(s) displaying any log information along the way. It should be made-up of different panels containing functional components and charts. At this level, a standard look and feel should be available for the user, with programmers adhering to this for all their new graphical interfaces.

Software performance should, in principle, be high enough to keep up with the hardware. However, it is accepted that this is not always possible or desirable, and hence can be limited to a certain amount of data and to a maximum update rate. Typically, an application should be able to display:

- 1D array chart or table at 10Hz with 10000 points;
- 5 x 1D array charts or tables at 10Hz with 2000 pts;
- 2D array chart/table at 10Hz with 100x100 pts;
- 10 x 1D array charts or tables at 1Hz with 10000 pts;
- 4 x 1D array charts or tables at 1Hz with 100000 pts;
- 2D array chart/image/table at 1Hz with 512x512 pts.

COMPARISON BETWEEN AVAILABLE TECHNOLOGIES

Java Swing

In order to prepare for the end of Java Swing and JavaFX support, all JavaFX applications will initially be migrated to Swing, with unused or outdated GUIs removed. Since they are then sharing the same dependencies (using a single Maven archetype), they will be adapted to the latest Java project template structure in order to ease maintenance and facilitate future migration.

QtQML

QML allows interfaces to be easily created from documents with a JSON type format. Chart display performance testing using random values were performed for three different QML technologies: QML&JavaScript, QML&C++ and QML&Python. Charts were tested using QtChart.

The performance obtained with QML were found to be very poor in all cases. In fact, it took several minutes to create the charts, with constant application blockages and a high consumption of CPU time and RAM.

Qt C++

Surprisingly, building a whole interface in this technology is relatively easy, even without the use of QtCreator, with numerous low-level mechanisms (memory management, etc.) taken care of. The C++ API of the CERN common middleware is also functional, and can be easily integrated into a Qt application.

In terms of rendering, several plugins and widgets have been tested. *QPainter* is the low-level drawing tool of Qt. It served as a reference to benchmark other plugins.

Qwt is a popular library for plotting data with Qt. It is a well-documented library. However, its setup and its usage are both tedious and its performance is similar to QPainter.

QCustomPlot is another popular library. It comes with less support than Qwt (no 3D plotting for example) but outperforms it in all other aspects. Its integration and API are simpler, and its performance is impressive: large graphs with a million points can be displayed and used effortlessly.

Other drawing plugins are also available, such as *QtCharts*. They may be good alternatives, but they were not tested because the three plugins mentioned above already provided a good overview of the capacity of Qt/C++.

A pure C++/Qt GUI seems to be a very good option. It has good performance (depending on the widget/plugin used), and it avoids intermediary layers which can increase memory usage and/or decrease responsiveness (as seen with PyQt for example).

However, different problems make it difficult to adopt. Firstly, the initial setup is difficult. One can struggle to find the proper use of qmake with another programmer then finding it difficult to link with the CMW library. Secondly, it is not well documented. Finally, it is not easy to find new (young) PyQt and C++ developers on the market.

Web Based Application

Many client-side and GUI developments have moved to the web and to cloud solutions (SaaS). Thus, web solutions have already evolved, and they could be a mature solution matching CERN's requirement.

A new approach using this technique is currently being tested to provide a fixed display. This allows the real-time display of acquisition data from any FESA server from both the TN and GPN networks through a Tomcat web server. The initial performance was seen to fulfill the specifications.

Web interfaces are becoming more and more mature and their portability are very interesting. However, their graphical performance is still lagging behind C++/Java/Python, and other issues like security still need to be addressed.

Another alternative could be the use of a combination of Python and a web framework like Django [7]. This solution would allow us to automate the creation of the web interface as well as all graphical components including charts, and at the same time to include Python scripts that can be executed automatically on the server side.

PyQt

Three different technologies combining Qt and Python to do chart rendering have been looked at.

Mathplotlib is an old library with good performance. It is easy to draw one single plot but difficult to include it in an advanced GUI with multi-plot panels.

QtChart provides a set of easy to use chart components that can be integrated into modern user interfaces. It can be used for example as QWidgets. Users can easily create impressive graphs by selecting one of the available chart themes. It has very good performance, but some basic functionalities such as zooming are not implemented. 3D rendering is available but there is no 3D plot component.

PyQtGraph [8] is a pure Python graphical library built on PyQt which is intended for use in mathematics/scientific/engineering applications. The library is very fast due to its heavy use of NumPy for number crunching and Qt graphics for fast display. Zooming, coloring, axes and units are present by default in its components and 3D plotting is available.

PYTHON AND PYQT REQUIREMENTS

Python Distribution and Integrated Development Environment (IDE)

A common distribution containing several Python 3.6 packages as well as some dedicated accelerator control specific packages has been released at CERN. There are still discussions ongoing about the IDE. QtCreator and QtDesigner fit well for the design and implementation of a GUI interface with Qt, but other frameworks such as PyCharm are also being assessed.

Project Design and Template

It is foreseen that each programmer that needs to write a new expert GUI should use the same project code template. This template should be structured in a way that it can separately handle:

the data interface (FESA, file, Logging database, Post-Mortem…). An intermediate layer to manage both data subscription for the different clients (mainly graphical components) and the dispatching of the data is currently being tested.

The graphical components. A nice way to standardize GUIs is to share common graphical component libraries.

The models that do the control

Basic examples could also be included into this project to simplify the implementation. Furthermore, scripts to either release code, to obfuscate packages or to deploy operational programs should make life much easier and would strengthen standardization.

Repository and Versioning

Python code and "binaries" should be committed to a standard repository. The same approach as for Java could be followed for the versioning, i.e. source code to be committed to Git and applications deployed under two possible versions: PRO for production and DEV for development.

Graphical Components

In order to structure the graphical components, three levels and repository locations are proposed. The low-level components composed of the basic graphical components used by everyone (button, label, text field…). The medium-level components that gather panels to be inserted into the GUI's main window. Some of them will already be implemented and available for all GUIs, such as a data viewer panel and timing panel. Finally, the high-level components that mainly include frames and windows.

Qt Widget Architecture

All Qt widgets should be distributed into three different layers. Every widget should contain the same Qt structure, especially if it is a common interface to exchange data.

Naming conventions can also be used for every class, variable and methods.

The design file (.ui file created with QtDesigner) and interface can be used to create a graphical skeleton. It should be possible to open the QtDesigner tool and to have automatic access to these Qt widgets.

CONCLUSION

The software section of the CERN beam instrumentation group already has an idea of the direction it will head towards for future GUI development efforts. Web, Python, and to a lesser extent C++ are all likely to be part of such projects, with the use of Java already in steady decline.

The investigations into C++ did not bring about any big surprises. It was already suspected that a C++ based GUI would outperform its web, Python and even Java counterparts. This was confirmed by the performance figures which were very good especially using QCustomPlot. Equally unsurprising was the significant learning curve required for C++ developments, such as understanding tools such as qmake, and the development environment with the complex dependency requirements for Qt and in-house libraries like CMW. Another problem with C++ is the difficulty to hire adequately trained developers and to find complete web documentation as well as code examples.

Web technologies have already been demonstrated to have a clear role in generic, fixed-display type applications, but their extended use for fully-fledged interactive applications is still not ideal. Nevertheless, we see a bright future for the web interfaces combined with Python.

The clear all-rounder from these investigations was Python. The super-quick learning curve, enormous third-party library base, and availability of GUI design tools means that even a junior developer can make a small GUI in a day or two. In terms of the needs for GUI building, the performance of charting components was identified as paramount. In the case of Python, there is a choice of several technologies, with PyQtGraph standing out as the best option. Some GUIs have already been developed using this technology and it appears to tick all the right boxes. On a more negative note, there are some doubts about the long-term maintainability of Python code, with more guidance for code structure and conventions required at the institute level, similar to what is currently available for Java. Likewise, the development lifecycle of Python applications in general currently tends to be very ad-hoc from one developer to another. Standardizing on IDE use, versioning, dependency management, code structure, code conventions, etc., are all things that need to happen, before we launch headlong into making dozens of new Python applications.

There is already a move away from JavaFX, with the porting of old GUIs to Java Swing. We will attempt to standardize on new Maven archetypes for Swing applications, trying to standardize on high-level components and structures as much as possible. When the time comes to eventually move to the next GUI development platform, we hope that this effort will pay dividends, in that the porting will be more formulaic given that all the applications should roughly follow the same conventions. In parallel like-for-like Python applications will be developed for some well identified Java GUIs to gain experience with realistic Python GUI development.

OUTLOOK

Web technology has evolved rapidly over the last decade and having simple but nice interfaces available in any web browser to display and manipulate data is definitely attractive. On-demand generation of HTML and JavaScript web pages containing data from any location is already common in the IT industry.

For several of our users such technology could respond to their needs. This would include a client-server mechanism made of a web interface provided by a server in charge of finding, setting and processing data. This server should be able to mix different languages to enable Python scripts to be run to process data (Figure 2). The look and feel and performance on the client side should only depend on the architecture selected to generate the code. In the end, everything regarding the interface can be standardized and simplified with only the main data source servers needing to be modified. We will soon start to investigate the potential of this technology using the Django framework.

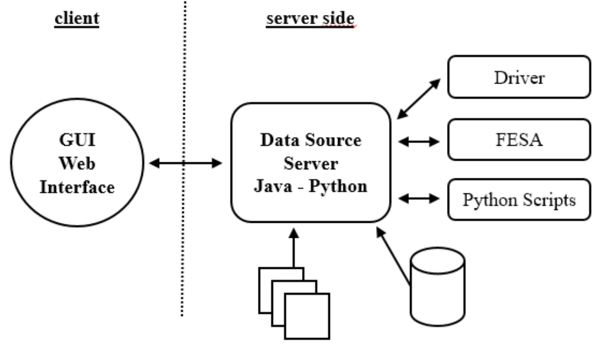

Figure 2: Web interface combined with Java or Python data source server.

REFERENCES

[1] Oracle Java SE Support Roadmap, https://www.oracle.com/technetwork/java/java-se-support-roadmap.html

[2] Python 3.6, https://docs.python.org/3.6/

[3] L. Fernandez *et al.*, "Front-End Software Architecture", in *Proc. ICALEPCS'07*, Oak Ridge, TN, USA, Oct. 2007, paper WOPA04, pp. 310-312.

[4] A. Dworak, F. Ehm, P. Charrue and W. Sliwinski, "The new CERN Controls Middleware", *Journal of Physics Conference Series*, vol. 396, Dec. 2012.

[5] PyQt5, https://pypi.org/project/PyQt5/

[6] M. Zerlauth *et al.*, "The LHC Post Mortem Analysis Framework", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper TUP021, pp. 131-133.

[7] Django, https://www.djangoproject.com/

[8] PyQtGraph, http://www.pyqtgraph.org/