ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR03

# OUR JOURNEY FROM JAVA TO PYQT AND WEB FOR CERN ACCELERATOR CONTROL GUIS

I. Sinkarenko, S. Zanzottera, V. Baggiolini, BE-CO-APS, CERN, Geneva, Switzerland

*Abstract*

For more than 15 years, operational GUIs for accelerator controls and some lab applications for equipment experts have been developed in Java, first with Swing and more recently with JavaFX. In March 2018, Oracle announced that Java GUIs were not part of their strategy anymore [1]. They will not ship JavaFX after Java 8 and there are hints that they would like to get rid of Swing as well.

This was a wakeup call for us. We took the opportunity to reconsider all technical options for developing operational GUIs. Our options ranged from sticking with JavaFX, over using the Qt framework (either using PyQt or developing our own Java Bindings to Qt), to using Web technology both in a browser and in native desktop applications.

This article explains the reasons for moving away from Java as the main GUI technology and describes the analysis and hands-on evaluations that we went through before choosing the replacement.

## INTRODUCTION

The majority of operational GUI applications running in the CERN Control Centre and other accelerator control rooms have been written in Java Swing. This technology is used here since the early 2000s. The Controls Group only develops general-purpose GUIs for mission critical applications, such as the InCA/LSA settings management [2] or the Sequencer [3]. For all the other GUIs, we rely on the equipment experts and operators to develop them because they know best what these GUIs should look like, and want to flexibly adapt them. To facilitate their task, we provide a framework, which consists of an application frame with a toolbar and a logging console, a graph component (JDataViewer), and, of course, different controls-specific widgets. We also provide a comprehensive set of client APIs to interact with the control system.

In early 2016, we moved away from Swing because we saw it as a legacy technology and recommended JavaFX as the successor, after it had become an official part of the Oracle JDK. Of the 500 operational GUIs currently in use, 90% are still in Java Swing. Unfortunately, in 2018, Oracle, the company backing Java, stated in their Java Client Roadmap Update [1] that JavaFX is a "niche" technology with a "market place [that] has been eroded by the rise of mobile-first and web-first applications", and announced that JavaFX is no longer a part of the Oracle JDK, but shall live on as an independent product, to be maintained by the open-source community. This official announcement was a wake-up call for us. We decided to completely re-evaluate our strategy and technology choices for GUI, even at the cost of not using Java – our core technology – for GUIs anymore.

## CRITERIA FOR SELECTING A NEW GUI TECHNOLOGY

In our evaluation of GUI technologies, we considered the following criteria:

- Technical match: suitability for Desktop GUI development and good integration with the existing controls environment (Linux, Java, C/C++) and the APIs to the control system;
- Popularity among our current and future developers: little (additional) learning effort, attractiveness for new recruits;
- Longevity of the technology and reasonable maintenance cost medium-term to long-term.

We looked at a very broad spectrum of technology and then seriously evaluated the following options:

- Java Swing or JavaFX – continue with Java in spite of the Oracle announcement;
- Web technology – use the currently most popular GUI technology;
- Qt – use one of the most popular desktop GUI frameworks with either Java or join the exploding popularity of Python with PyQt, or adopt future-oriented QtQuick GUIs.

In our evaluation, we accepted a possible move away from Java to another high-level language, such as JavaScript/TypeScript or Python, but we have never considered moving our users or ourselves to C or C++.

Most of the content below is explained in far more detail in the Master Thesis of one of the authors [4].

## ANALYSIS

This section summarizes our findings and assessment during the analysis. Please note that this evaluation was done in 2018 and refers to the situation at that moment. Also, some of the statements are based on empirical analysis and might be tainted by personal interpretations.

### *Java Swing or JavaFX*

The technical match and integration with the controls system is (obviously) very good: as our long-standing technology, it fulfils our needs very well and has an excellent integration with the controls system. Acceptance is mixed. Those developers who know and use JavaFX typically like it very much. However, it clearly is less attractive for new recruits – very few young engineers want to learn JavaFX, let alone Swing.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR03

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Longevity is the main problem, especially for JavaFX without the support of Oracle. During several months, we tried to assess the liveliness of the JavaFX community, by looking at Twitter feeds [5], discussion groups [6], and activity on the GitHub repo [7]. We participated in a workshop "JavaFX beyond 2022" in Munich, to meet with many other companies invested in JavaFX and to talk in person to an official Oracle representative. We concluded from all this that the future of JavaFX is not promising enough; we rather expect it to slowly decline. On the other hand, the future of Java Swing seems quite stable to us, for several reasons. It is impossible for Oracle to abandon Java Swing unilaterally, because it is part of the official Java SE platform, which is governed by the Java Community Process (JCP) [8]. Any change to the Java SE platform has to be approved by an expert group and a community vote. Given the fact that many companies are heavily invested in Java Swing (much more than in JavaFX), it is unlikely that the JCP will vote for removing Java Swing. If - against all odds - Swing eventually was to be removed from Java SE, there will be companies who see a business opportunity in providing commercial support for Java Swing to many legacy users. Unfortunately, it is impossible for us to take over the support of Java Swing ourselves because it is a huge framework consisting of over a million lines of Java and half a million lines of (low-level) C and C++. For the same reason we have never considered taking over support of JavaFX.

In summary, we concluded that we cannot bet on Java as the future GUI technology and need to get rid of JavaFX, but we see Java Swing as "the new X/Motif", a technology that is outdated but likely to exist for many years to come.

### *Web Technology*

Web applications generally run in web browsers and connect to server-side business logic using HTTP(S). Due to security constraints, Web GUIs have very limited access to the computer that they run on, which is inconvenient for desktop applications. This can be overcome using frameworks such as Electron that rely on web technology to build desktop applications.

Integration with our controls system is currently difficult. Web-based clients communicate with REST, while almost all our operational services (InCA/LSA [2], Sequencer [3], PostMortem [9], LASER, etc.) have Java clients and use Java RMI (Remote Method Invocation) and JMS (Java Message Service) to communicate with the servers. While it is possible to embed Java libraries into a Python program (c.f. below), there is currently no modern and mainstream way to embed our existing Java client libraries directly into the browser.

Popularity depends. Web technology is very popular amongst young software engineers, and several software teams in the Accelerator Sector successfully use it to develop their applications. For example, our Data Services team uses it for advanced applications, such as AFT [10], CCDE [11], ASM [12], and has developed an infrastructure called "accsoft-commons-web" (ACW) [13] based on Angular. Other teams use alternative frameworks such as Vue.js. Conversely, Web is much less popular among our physicists, operators, or hardware specialists, and in general, in the scientific world, where Python reigns.

Longevity is the main downside of Web. Technology changes frequently, new frameworks come and go [14] every few years. Currently there are three major frameworks (Angular, React, and Vue.js), but none of them is the clear winner for the future. There is Web components standard [15], aimed at sharing components between frameworks, but it is not widely adopted. Consequently, a team that invests into Web must be willing to adapt to a changing technology, both in terms of learning new frameworks and rewriting existing applications. This is not an option for our users, who do not have an intrinsic interest in software, but just want to use it as a tool to get their main job done.

In our specific context, we will explore Web for our read-only GUIs, such as Fixed Displays and user-configurable dashboards, which need to be accessible outside of control rooms, for instance, in offices or outside CERN.

### *Python and PyQt*

Python is an easy to learn general-purpose programming language. It is used for a broad spectrum of purposes, from embedded systems programming all the way to big data analysis. Qt, a popular GUI framework implemented in C++, is the default choice for writing desktop GUIs in Python, with PyQt or Qt for Python providing Python bindings. Qt is used, amongst others, in desktop applications, such as Linux KDE and WinCC OA [16]. It is also widely adopted in car dashboards, medical appliances and entertainment equipment.

Over the last few years, Python has been exploding in popularity. Worldwide, it is amongst the most popular programming languages [17] and the number one "want-to-learn" language [18]. Qt lies, in terms of popularity, between Web and Java GUI technology.

For already some years, Python has been very popular in CERN. Physicists use it for data analysis and Machine Development; equipment specialists conduct prototyping and testing; Controls and IT people write tools for code generation, DevOps tasks and system administration.

Integration with the current controls system is average: it is possible to call Java directly from Python using JPype [19], but overly complex from a technical viewpoint. We use this solution currently to enable Python GUIs to communicate with our Java servers. However, we envisage better alternatives, for example, direct integration of Python with C/C++ libraries, such as our controls middleware [20]; or communication over REST APIs, once they are provided by our core controls services.

For longevity, Python and PyQt look good. Both have existed for almost 30 years; Python is on the rise, and PyQt shows no weakness either. Maintaining Python code is more difficult than maintaining Java, but with the appropriate methods and tools, it is possible to develop well tested and easy to maintain operational GUI applications.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR03

## HANDS-ON EVALUATION

While doing the above analysis we evaluated the Qt GUI framework in general and PyQt in particular more thoroughly. In our evaluation, we excluded C++ from the list of possible programming languages because we considered it too complex for an average user in our community.

There are two ways of developing Qt GUIs: Qt Widgets and QtQuick. Qt Widgets are the original implementation of Qt, originating from the 1990s, suitable for traditional, natively looking desktop applications. QtQuick and QML (Qt Modelling Language) is the new way of developing Qt GUIs. QtQuick is a library and QML is "a user interface specification and programming language. [...]. QML offers a highly readable, declarative, JSON-like syntax with support for imperative JavaScript expressions combined with dynamic property bindings [21]." The Qt Company heavily promotes QtQuick as the future technology, but remains firmly committed to supporting Qt Widgets in the future as well [22].

Based on existing experience, we took for granted that we would be able to fulfil all our needs using PyQt based on Qt Widgets. However, as we wanted to be up-to-date with the newest Qt developments, we decided to invest time into evaluating QML.

The following sections describe our experience with QtQuick, QML and JavaScript, and explains why we eventually abandoned it in favour of PyQt with Qt Widgets. In our evaluation, we mainly studied two areas: (1) representing data in many different types of charts, with special attention to good performance, and (2) use of typical desktop widgets such as trees, tables, tree-tables, with the focus on how well they can be customized.

### *QtQuick/QML with JavaScript*

Given the fact that we excluded C++ as a main programming language, the idea was to use JavaScript (or even better, TypeScript) to develop Qt GUIs. This would have enabled us to align with web development and join forces with our colleagues doing Web. Based on preliminary research, we even hoped that it would be possible to integrate QML GUIs with 3rd-party JavaScript libraries and share code with our Web colleagues.

QtQuick appears to be aimed at building highly aesthetic, carefully styled UIs such as car dashboards with beautiful gauges and stunning dynamic content. On the other hand, it puts less emphasis on typical desktop application widgets, such as tables and trees.

QML gives JavaScript access only to a sub-set of the Qt APIs. In general, the Layout API is exposed, while most of the underlying data model APIs are hidden from QML and reserved for C++, in an attempt to enforce a very strict MVP (Model-View-Presenter) implementation. In other words, many APIs accessible from C++ or Python are not available to JavaScript. This fact can be limiting in advanced use cases, especially where the interface between C++ and QML has been not designed carefully enough, like in the case of QtCharts (c.f. below).

QtQuick limitations become more apparent when using third party libraries because many of them do not have QML bindings (yet). For instance, an excellent library for plotting, QCustomPlot, does not support QML [4]. When we contacted its main developer regarding the QML support, he replied that he was considering it, but wanted QML to become stable before investing any time into it.

The only plotting library with QML bindings was QtCharts [4] [23], which is provided by The Qt Company. When used with JavaScript only, it exhibits terrible performance: while the library can render almost a million points per second using the C++ or Python API, the API exposed to QML reduce this value by two/three orders of magnitude [4]. The reason is that the underlying data model exposes only two functions to JavaScript: `append(x,y)` and `clear()`, and the graph is repainted every time a single point is appended.

For people with experience in JavaFX, the declarative programming style of QML felt awkward, because unlike QML, which the developer is expected to read and even edit when integrating JavaScript logic, FXML (the declarative part of JavaFX) remains hidden. Conceptually FXML is in fact closer to the XML *.ui files used for Qt Widgets. Also, several of our Python users, who had already explored PyQt with imperative Qt APIs, were not willing to embrace QML. QML might feel familiar to web developers, but none of our JavaFX developers had experience with Web.

QtQuick GUIs are typically a mix of QML, JavaScript and the "host" language (C++ or Python). It is a good practice to add some JavaScript to the QML, to specify GUI behaviour. However, there are no obvious and easy to explain guidelines on how to split code between QML, JavaScript methods, and the "host" language. While we were confident that software engineers would find a good balance between the three, we were sceptical that our typical physicist, operator or equipment specialist would have achieved the same. In other words, we were expecting them to develop hard to maintain GUIs with large amounts of JavaScript mixed into QML.

Integration with the native libraries, such as control systems client libraries, would normally have to be done in the "host" language. However, having a vision of a JavaScript-oriented development approach, we attempted to expose aforementioned libraries via the Qt plugin mechanism, removing the need to write any glue code, as long as UI elements are attached to Qt Signals/Slots exposed by the library plugins. In addition to removing overhead from high-level JavaScript developers, it limits the maintenance of the plugins to be managed only by experienced developers.

Debugging JavaScript and QML is possible only with the Qt Creator, the IDE that comes with Qt. However, we found this IDE far less advanced and user-friendly than the competing IDEs we currently use.

Using TypeScript rather than JavaScript turned out to be elusive. We had discussed this idea with a senior technical engineer at the Qt developer conference, and it appeared that the company was very interested in strong typing.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR03

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Encouraged by them, we filed a Feature Request [24], but the type system support is still pending, even a year later [25].

Last but not least, our dream of reusing JavaScript code with Web has vanished as well. The JavaScript in the QML engine is intended for relatively small snippets of code related to GUI logic. While it is up-to-date with modern JavaScript standard, the JavaScript Host Environment [26] is not directly compatible with 3rd-party libraries commonly found in NPM (Node Package Manager) repositories. There are conversion tools, and some developers have tried to integrate full Node.js-like environment with QML, but this remains an exotic approach, and the corresponding projects are stale [27] [28] [29].

### *Qt with Java Bindings*

This was an alternative exploration. Our (obvious) hope here was to reuse the massive investment we already have in Java client libraries and tooling, and only switch the GUI toolkit from JavaFX to Qt. Our first attempt was to use Qt Jambi [30], the official Java binding for Qt. We soon discovered that it was completely obsolete, based on a Qt version from over 10 years ago, and not supported by The Qt Company anymore. We nevertheless tried to take the hopelessly outdated sources in an attempt to make it run with the latest Qt version. We failed miserably, and had to admit that updating the code to Qt5 ourselves would be a major endeavour.

We did not give up and instead explored the possibility to make a binding between QtQuick/QML and Java. Unlike Qt Jambi, which maps the whole Qt API with hundreds of classes and thousands of method calls, a binding for QtQuick/QML would have been limited to a dozen of classes and their methods. Generally, only QObject, QMetaObject and related APIs would need to be exposed along with types for data packaging, such as QVariant. With this limited set of APIs, Qt's Signal/Slot mechanism could be used to marshal data between UI elements created from QML and logic written in Java. We have made a successful proof of concept using a nice little open-source library DOtherSide [31]. We finally abandoned this approach after we had decided to abandon QML.

### *PyQt*

After discarding QML for the reasons above, the obvious choice was to go with the flow and adopt Python and PyQt.

We can benefit from prior knowledge and experience that exists among our users, which have been developing PyQt GUIs in the labs. PyQt connects us to the Python community that stands strong in the scientific environment, hence allowing us to attract interested people in CERN and associated entities.

While switching GUI development from Java (which served us loyally for decades) to another language inevitably presents challenges in terms of migration, we were confident with our decision and have not looked back since. Our resulting plans are described below.

## CONCRETE PLANS

Based on the above analysis, we have chosen Python and PyQt as the new recommended technology for operational GUIs. We are developing a similar framework as we provide it for Java Swing. In parallel, we are working on a PyQt-based Rapid Application Development (RAD) toolkit, which shall make it possible to develop dashboards and simple applications with no or little Python coding. One important design goal is that a RAD application could be transformed into a standalone fully-fledged operational PyQt application without any need to rewrite it from scratch. This transition is not possible in current RAD frameworks, such as CO Fixed Displays and Inspector. In terms of technology, we use PyDM [32] developed at SLAC. Regarding Java, we will maintain our existing frameworks and components, without however adding any new functionality. Our support for JavaFX will end in 5 years. Java Swing remains our recommendation to develop new operational GUIs, until the new PyQt-based environment is ready. We will do our best to keep Swing operational for the next 15 years.

As for the web technology, inside our team, we currently limit ourselves to evaluating its suitability to build Fixed Displays and fully-fledged controls applications. This will allow us to gain experience with Web and to give our colleagues feedback on the ACW framework.

## SUMMARY

Java GUI technology is declining after almost 20 years of loyal services. In 2018, Oracle has officially announced that they will not bundle JavaFX anymore with the Java distribution, and relies on the open source community to maintain it as a separate project. This has made us completely review our strategy and technology choices for GUIs. We made a broad analysis based on various criteria such as suitability for desktop GUI development and controls, popularity among current and future developers, learning effort, longevity and expected maintenance cost. The outcome of this analysis leads us to choose Python and PyQt as the new platform for operational GUI applications and rapid application development (RAD), and Web mainly for Fixed Displays and dashboards. We aim for the first developer preview to be ready at the end of 2019. We will support JavaFX for 5 years, and Java Swing during next 15 years. Java Swing is the recommended technology until the PyQt-based frameworks are ready.

## REFERENCES

[1] Oracle Corporation, "Java Client Roadmap Update," March 2018. [Online]. Available: https://www.oracle.com/technetwork/java/javase/javaclientroadmapupdate2018mar-4414431.pdf

[2] D. Jacquet, R. Gorbonosov, and G. Kruk, "LSA - the High Level Application Software of the LHC - and Its Performance During the First Three Years of Operation", in *Proc. ICALEPCS'13*, San

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPR03

Francisco, CA, USA, Oct. 2013, paper THPPC058, pp. 1201-1204.

[3] V. Baggiolini, R. Alemany-Fernandez, R. Gorbonosov, D. Khasbulatov, and M. Lamont, "A Sequencer for the LHC Era", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THC003, pp. 670-672..

[4] S. Zanzottera, "Evaluation of Qt as GUI Framework for Accelerator Controls," *Politecnico di Milano, Italy,* 2018.

[5] "Feed for JavaFX," Twitter, [Online]. Available: https://twitter.com/hashtag/javafx?lang=en. [Accessed 2018]

[6] "The openjfx-dev Archives," [Online]. Available: https://mail.openjdk.java.net/pipermail/openjfx-dev/. [Accessed 2018].

[7] "OpenJFX Mirror," GitHub, [Online]. Available: https://github.com/javafxports/openjdk-jfx/. [Accessed 2018].

[8] "Java Community Process," Oracle Corporation, [Online]. Available: https://www.jcp.org/en/home/index

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

[10] C. Roderick, L. Burdzanowski, D. Martin Anido, S. Pade, and P. Wilk, "Accelerator Fault Tracking at CERN", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 397-400. doi:10.18429/JACoW-ICALEPCS2017-TUPHA013

[11] L. Burdzanowski *et al.*, "CERN Controls Configuration Service - a Challenge in Usability", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 159-165. doi:10.18429/JACoW-ICALEPCS2017-TUBPL01.

[12] B. Urbaniec and C. Roderick, "Accelerator Schedule Management at CERN", presented at the ICALEPCS'19, New York, NY, USA, Oct. 2019, paper MOPHA149, this conference.

[13] BE-CO-DS, "Accsoft Commons Web," CERN, [Online]. Available: https://gitlab.cern.ch/accsoft-commons-web

[14] M. Raible, "History of Web Frameworks 2015," Flickr, [Online]. Available: https://www.flickr.com/photos/mraible/20606289343/

[15] "Web Components Specifications," [Online]. Available: https://www.webcomponents.org/specs

[16] "SIMATIC WinCC Open Architecture Portal," ETM professional control GmbH, [Online]. Available: https://www.winccoa.com/

[17] "TIOBE Index for September 2019," September 2019. [Online]. Available: https://www.tiobe.com/tiobe-index/

[18] "Most Loved, Dreaded, and Wanted," Stack Overflow, [Online]. Available: https://insights.stackoverflow.com/survey/2018/#most-loved-dreaded-and-wanted

[19] "JPype," GitHub, [Online]. Available: https://github.com/jpype-project/jpype

[20] V. Baggiolini, S. Jensen, K. Kostro, F. DiMaio, A. Risso, and N. Trofimov, "Remote Device Access in the New CERN Accelerator Controls Middleware", in *Proc. ICALEPCS'01*, San Jose, CA, USA, Nov. 2001, paper THAP003, pp. 496-498.

[21] The Qt Company, "QML Applications," [Online]. Available: https://doc.qt.io/qt-5/qmlapplications.html

[22] The Qt Company, "Technical vision for Qt 6 - The next big release," 7 August 2019. [Online]. Available: https://www.qt.io/blog/2019/08/07/technical-vision-qt-6

[23] The Qt Company, "Qt Charts," [Online]. Available: https://doc.qt.io/qt-5/qtcharts-index.html. [Accessed 2018].

[24] "Qt Bug Tracker: Use TypeScript to write GUI logic in Qt Quick (instead of JavaScript or C++)," [Online]. Available: https://bugreports.qt.io/browse/QTBUG-68810

[25] "Qt Bug Tracker: QML Type system," [Online]. Available: https://bugreports.qt.io/browse/QTBUG-68791. [Accessed September 2019].

[26] The Qt Company, "JavaScript Host Environment," [Online]. Available: https://doc.qt.io/qt-5/qtqml-javascript-hostenvironment.html

[27] "Quickly," GitHub, [Online]. Available: https://github.com/quickly/quickly

[28] "Brig," GitHub, [Online]. Available: https://github.com/BrigJS/brig. [Accessed 2018].

[29] "Node.qml," GitHub, [Online]. Available: https://github.com/trollixx/node.qml

[30] The Qt Company, "Qt Jambi Reference Documentation," [Online]. Available: https://doc.qt.io/archives/qtjambi-4.5.2_01/com/trolltech/qt/qtjambi-index.html

[31] "DOtherSide," GitHub, [Online]. Available: https://github.com/filcuc/DOtherSide

[32] SLAC, "PyDM - Python Display Manager," [Online]. Available: http://slaclab.github.io/pydm/

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2019). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.