ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA154

JavaFX AND CS-STUDIO: BENEFITS AND DISADVANTAGES IN DEVELOPING THE NEXT GENERATION OF CONTROL SYSTEM SOFTWARE

C. Rosati*, European Spallation Source ERIC, Lund, Sweden
K. Shroff†, BNL, Upton, NY, USA
K. Kasemir‡, ORNL, Oak Ridge, TN, USA

Abstract

The new developments inside the CS-Studio community [1–4] were made using the JavaFX platform [5–9] to overcome the limitations and difficulties of using Eclipse SWT. This article will explain the benefits and disadvantages of using the JavaFX technology inside Eclipse RCP, and try to foresee the path of the new generations of CS-Studio application.

INTRODUCTION

Control System Studio (CS-Studio, see [1, 2]), is a multi-platform, Eclipse-based [10–12] desktop application, containing tools and features to monitor and operate large scale control systems, such as the ones in the accelerator community.

The first version of CS-Studio is dated back to 2006 [1]. Its implementation was based on the Eclipse Rich Client Platform (Eclipse RCP, see [11, 12]), and its user interface was realized using the Eclipse Standard Widget Toolkit (Eclipse SWT, see [12, 13]).

Since August 2012, when Oracle released version 2.2 of JavaFX (finally available for Windows, Max OS X, and Linux, see [5]), thanks to the availability of the `FXCanvas` class[1] [7,8,14] allowing JavaFX be embedded into an Eclipse SWT user interface, the CS-Studio community started to develop new tools and features using JavaFX instead of Eclipse SWT.

Currently, the following features are based on JavaFX (see Fig. 1):

- Data Browser 3;
- Display Builder (see [15, 16]);
- Fault tools;
- Logging Configuration;
- Probe;
- PV Tree;
- Save & Restore, and its Periodic Table.

* claudio.rosati@esss.se
† shroffk@bnl.gov
‡ kasemirk@ornl.gov

[1] Already available on JavaFX 2.0, only for the Window platform [5].

ECLIPSE RCP

Eclipse Rich Client Platform [11, 12] has served the CS-Studio community well for about a decade. While Eclipse RCP is stable, this also means it has not offered significant new features for the CS-Studio use cases in the last couple of years.

SWT [13] and RAP (Remote Application Platform, see [17]) developments have halted. e(fx)clipse (JavaFX Tooling and Runtime for Eclipse and OSGi, see [18]) has stalled, not progressing towards replacing SWT. E4 [12, 19] is an interesting concept at the lower API level, but the 'compatibility' layer remains the only practical API. The tycho-based build system [20] causes repeated issues with CS-Studio build setups.

Benefits

The following are recognized by the CS-Studio community to be the advantages of developing CS-Studio with the Eclipse RCP framework:

- OSGi [21] packaging, control of exported packages, dependencies;
- Extension points mechanism
  - for PV types, logbook support, archive data source, widgets,
  - for online help,
  - for preference UI;
- Cross platform (Max OS X, Linux, Windows);
- Configuration of site-specific products via plugins and features;
- OSGi console with 'telnet' access;
- Jetty [22] for services that have web interface;
- Hierarchical preferences;
- Workspace persists window layout and preference changes;
- Object contribution mechanism for context menu: Open apps on files, "PV Name" context menu;
- Support for CVS [23], Subversion (SVN, [24]), Git [25];

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA154

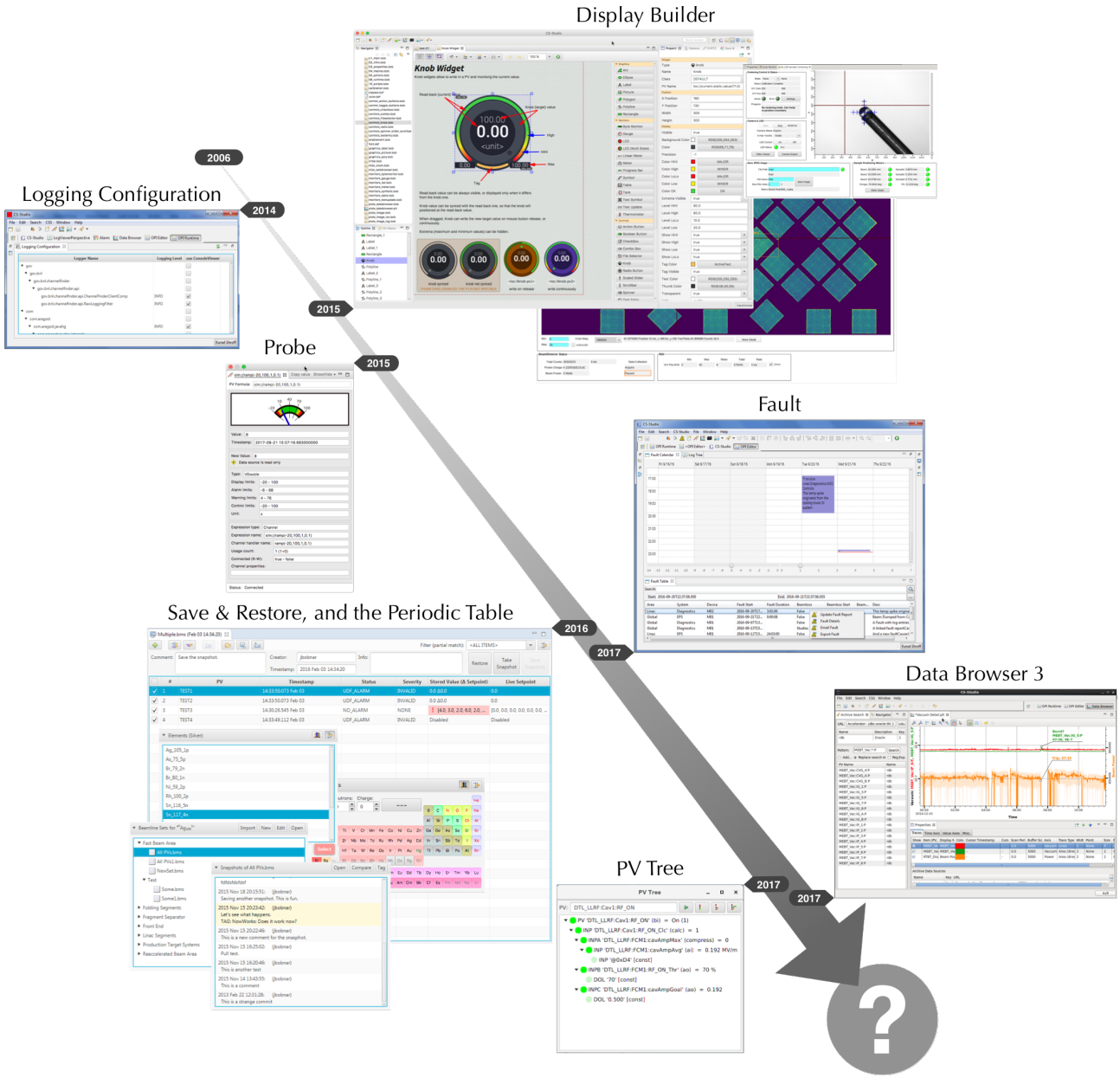

Figure 1: JavaFX features in CS-Studio.

- PyDev [26];
- XML Editor;
- WikiMedia Editor [27];
- Maven (in theory, [28]).

Disadvantages

The following are instead the disadvantages of the Eclipse RCP development:

- 'Editors' part stack doesn't participate in Perspective handling, opening up when any 'Launcher' is invoked [12];
- No control over initial location of newly opened parts, with restrictions on moving parts around (this being one of the bigger overall issue);
- API has grown in too many directions: PopupMenu, ActionSets, Action & Handler, E4 model;
- JavaFX in SWT FXCanvas [7, 8, 14] is sluggish;
- Workspaces and associated restrictions running on multiple machines;
- Somewhat heavyweight application with slow initial launch phase;
- Maven (in practice, when Tycho plug-in [20] is involved).

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA154

PHOEBUS APPLICATION FRAMEWORK

The upcoming Java 9 release [29] will come with an important improvement: the Java Platform Module System (JPMS, [30, 31]). The new way of packaging modules and declaring modules dependency opens a new perspective on developing the new generation of CS-Studio based only on Java/JavaFX.

Few application frameworks based entirely on JavaFX exist. Some are based on OSGi for module management [32], others are not open-source [33], or not exclusively targeted to JavaFX [32, 34, 35], and no one seems to be really full-fledged, well supported, or even mature enough to start developing the new CS-Studio [36–38] [2]. The same was proved true also for some specific components, like the docking framework [39–41], not available by default in the standard JavaFX platform.

That said, we are exploring the implementation of a new application framework: *Phoebus* [42]. The aim is to allow the update of the Control System Studio toolset removing the dependencies on Eclipse RCP and SWT.

The key goals of the Phoebus project are (see Fig. 2):

- Allow easy migration of the functionalities of key CS-Studio tools, specifically the Display Builder, Data Browser, PV Table, PV Tree, Alarm UI, and Scan UI, supporting their original configuration files with 100% compatibility;
- Provide full control of windows placement, free from RCP restrictions;
- Use JavaFX as the graphics library to overcome limitations of SWT;
- Prefer core Java functionality over external libraries whenever possible (JavaFX for UI, Java 9 Platform Module System for bundling, SPI – Service Provider Interface [43] – for locating and loading services and extensions, `java.util.logging` for logging and `java.util.prefs` for preferences, ...);
- Reduce build system complexity, fetching external dependencies in one initial step, then supporting a fully standalone, reproducible build process;
- Provide shared modules and services for
  - Actions,
  - Menu bar,
  - Toolbar,
  - Status bar,
  - Selection (contextual content),
  - Docking,
  - Windows management,
  - Splash screen,
  - Application start-up and shutdown.

[2] Drombler FX [32], Gluon Desktop [33], Griffon [34] being nevertheless the most interesting ones.

OSGi vs. JPMS

OSGi (Open Services Gateway initiative [44]) specification describes a modular system and a service platform for the Java programming language that implements a complete and dynamic component model, something that did not exist in standalone Java/VM environments before version 9. Applications or components, coming in the form of bundles for deployment, can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management is implemented via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly.

The OSGi specifications have evolved beyond the original focus of service gateways, and are now used in applications ranging from mobile phones to the open-source Eclipse IDE/RCP.

The Java Platform Module System [30] is intended to raise the abstraction level of coding in Java making the Java SE Platform, and the JDK, more easily scalable down to small computing devices, improving the security and maintainability of Java SE Platform implementations in general, and the JDK in particular, enabling improved application performance, and making it easier for developers to construct and maintain libraries and large applications, for both the Java SE and EE Platforms. The module system is be powerful enough to modularize the JDK and other large legacy code bases, yet still be approachable by all developers.

Comparing the two technologies in the context of the CS-Studio application we discover that:

- OSGi and JPMS offer similar control of dependencies and exposed packages;
- OSGi supports multiple versions of the same plugin. In practice this is more of a disadvantage. We never needed, for example, two different versions of MySQL being loaded contemporarily;
- OSGi allows dynamic shutdown and replacement of plugins (necessary mostly for embedded applications), but we never used this feature.

Extension Points vs. SPI

Eclipse extension points can provide information (label, icon path, ...) as well as classes which can then be instantiated. In principle, this allows users of extension points to, for example, present menu entries with the label and icon of an extension without actually invoking any code from the extension.

In comparison, Java Service Provider Interface only allows obtaining implementations of the service interfaces. There is no way to obtain a descriptor-like information such as label or icon for a service without instantiating and invoking the service intself. This likely requires many

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA154

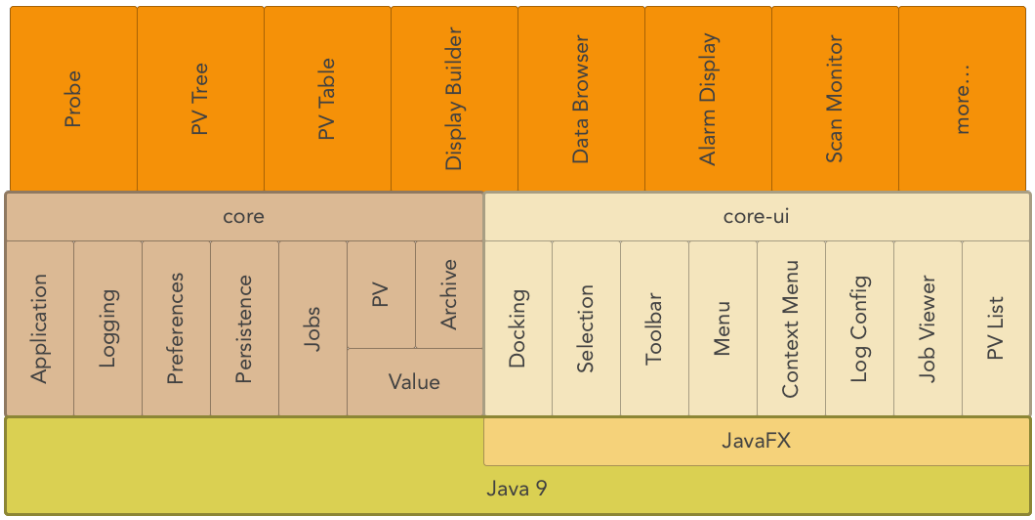

Figure 2: Phoebus architecture.

services to be in the form of factory classes that offer `getLabel()`, `getIconPath()` methods, and finally allow creating objects for that service. This is acknowledged in the `ServiceLoader` javadoc, where:

> "The provider class is typically not the entire provider itself but rather a proxy which contains enough information to decide whether the provider is able to satisfy a particular request together with code that can create the actual provider on demand."

The code for the service, i.e. the factory proxy class, is instantiated even if just the label is required and the actual service will otherwise never be used. Nevertheless, this service implementation pattern will produce service code with an acceptable, very low memory footprint.

Cross Platform Support

Eclipse supports Windows, Linux, Mac OS X. Java 9 approach is likely at least as 'cross platform', may even allow support of Raspberry Pi and Android/iOS [45, 46]

CONCLUSION

Current development of CS-Studio is based on Eclipse RCP, but in recent years a lot of development was done using embedded JavaFX because of its many advantages and easiness of use compared with Eclipse SWT. Most of the more important tools in CS-Studio have a JavaFX implementation: Display Builder, DataBrowser, PV Tree, ...

The upcoming release of Java 9 will bring the new Java Platform Module System, providing the Java Platform with the most critical feature available in Eclipse RCP through the OSGi technology.

This is most favourable moment in time to start the development of a "pure JavaFX" version of CS-Studio, while maintain the current Eclipse RCP-based version by improving the JavaFX tools already available.

Phoebus project was started few months ago to explore the feasibility of using the new Java 9 JPMS and the standard SPI as means for porting the existing codebase into a new JavaFX-only application, and the initial test are more than promising. The development of such a framework is running fast, and more developers from the various accelerator facilities around the world are expected to join soon the initial taskforce setup by BNL, SNS and ESS.

Soon the current Phoebus project [42] will be restructured to provide

- better separation between the core frameworks and the extension ones,
- better documentation,
- various examples of different complexity, exploiting the various aspects of the framework.

Not only CS-Studio will benefit from the Phoebus project development. Our hope is that other applications in the accelerator community [47] (and not only), wishing to use a pure Java/JavaFX modern UI framework, will adopt it, and possibly more developers will join the Phoebus community in the common effort of improve and keep alive this exciting project.

REFERENCES

[1] J. Hatje, M. Clausen, C. Gerke, M. Moeller, and H. Rickens, "CONTROL SYSTEM STUDIO (CSS)," in *ICALEPCS 2007*. Oak Ridge National Laboratory, October 2007. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/MOPB03.PDF

[2] Control System Studio. [Online]. Available: http://controlsystemstudio.org/

[3] GitHub – Control System Studio. [Online]. Available: https://github.com/ControlSystemStudio

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA154

[4] K. Kasemir and G. Carcassi. Control System Studio Guide. [Online]. Available: http://cs-studio.sourceforge.net/docbook/index.html

[5] JavaFX. [Online]. Available: https://en.wikipedia.org/wiki/JavaFX

[6] JavaFX: Getting Started with JavaFX. [Online]. Available: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.html

[7] H. Schildt, *Introducing JavaFX^{TM} 8 Programming*, 1st ed. McGraw-Hill Education, 2015.

[8] J. Vos, W. Gao, S. Chin, D. Iverson, and J. Weaver, *Pro JavaFX 8: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients*, 1st ed. Apress Media, 2014.

[9] H. Ebbers, *Mastering JavaFX® 8 Controls*, 1st ed. McGraw-Hill Education, 2014.

[10] Eclipse. [Online]. Available: https://www.eclipse.org/home/index.php

[11] Eclipse Rich Client Platform. [Online]. Available: https://wiki.eclipse.org/Rich_Client_Platform

[12] L. Vogel, *Eclipse Rich Client Platform: The complete guide to Eclipse application development*, 3rd ed. Lars Vogel, May 2015.

[13] SWT: The Standard Widget Toolkit. [Online]. Available: https://eclipse.org/swt/

[14] Class FXCanvas. [Online]. Available: http://docs.oracle.com/javase/8/javafx/api/javafx/embed/swt/FXCanvas.html

[15] K. Kasemir, M. Grodowitz, A. Carpenter, and C. Rosati, "CS-Studio Display Builder Update," in *Spring 2017 EPICS Collaboration Meeting*. Research Reactor Institute, Kyoto University (KURRI), May 2017. [Online]. Available: http://www.rri.kyoto-u.ac.jp/EPICS/materials/DisplayBuilderUpdate_2017_05.pptx

[16] M. Grodowitz and K. Kasemir, "CS-Studio Display Builder – Tutorial," in *Spring 2017 EPICS Collaboration Meeting*. Research Reactor Institute, Kyoto University (KURRI), May 2017. [Online]. Available: http://www.rri.kyoto-u.ac.jp/EPICS/materials/DisplayBuilder_training_2017.pdf

[17] Eclipse Remote Application Platform. [Online]. Available: http://www.eclipse.org/rap/

[18] e(fx)clipse – JavaFX Tooling and Runtime for Eclipse and OSGi. [Online]. Available: https://www.eclipse.org/efxclipse/index.html

[19] Eclipse e4 Project. [Online]. Available: https://eclipse.org/e4/

[20] Eclipse Tycho – Building Eclipse plug-ins with maven. [Online]. Available: https://eclipse.org/tycho/

[21] OSGi™Alliance. [Online]. Available: https://www.osgi.org

[22] Eclipse Jetty. [Online]. Available: http://www.eclipse.org/jetty/

[23] Eclipse CVS. [Online]. Available: https://www.eclipse.org/eclipse/platform-cvs/

[24] Eclipse Subversive – Subversion (SVN) Team Provider. [Online]. Available: http://www.eclipse.org/subversive/

[25] Eclipse EGit. [Online]. Available: http://www.eclipse.org/egit/

[26] Eclipse PyDev. [Online]. Available: http://pydev.org

[27] Eclipse Mylyn WikiText. [Online]. Available: https://wiki.eclipse.org/Mylyn/WikiText

[28] Eclipse M2Eclipse. [Online]. Available: http://www.eclipse.org/m2e/

[29] JDK 9 Early-Access Builds. [Online]. Available: http://jdk.java.net/9/

[30] S. Colebourne. (2017) Java 9 modules - JPMS basics. [Online]. Available: http://blog.joda.org/2017/04/java-9-modules-jpms-basics.html

[31] Ł. Gajowy. (2017) Exploring Java 9 – Java Platform Module System. [Online]. Available: https://www.polidea.com/blog/Exploring-Java-9-Java-Platform-Module-System/

[32] Drombler FX – The modular application framework for JavaFX. [Online]. Available: http://www.drombler.org/drombler-fx/

[33] Gluon Desktop. [Online]. Available: http://gluonhq.com/products/desktop/

[34] Griffon. [Online]. Available: http://griffon-framework.org

[35] JacpFX. [Online]. Available: http://jacpfx.org/index.html

[36] Basilisk. [Online]. Available: https://github.com/basilisk-fw/basilisk

[37] JRebirth. [Online]. Available: http://www.jrebirth.org/index.html

[38] mvvmFX. [Online]. Available: https://github.com/sialcasa/mvvmFX

[39] AnchorFX. [Online]. Available: https://github.com/alexbodogit/AnchorFX

[40] DockFX. [Online]. Available: https://github.com/RobertBColton/DockFX

[41] Mosaic. [Online]. Available: https://github.com/fxpresso/Mosaic

[42] phoebus. [Online]. Available: https://github.com/shroffk/phoebus

[43] Introduction to the Service Provider Interfaces. [Online]. Available: https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html

[44] OSGi. [Online]. Available: https://en.wikipedia.org/wiki/OSGi

[45] Gluon VM. [Online]. Available: http://gluonhq.com/products/mobile/vm/

[46] Mobile Project. [Online]. Available: http://openjdk.java.net/projects/mobile/

[47] C. Rosati and E. Laface, "NEW APPROACH IN DEVELOPING OPEN XAL APPLICATIONS)," in *IPAC 2017*. European Spallation Source ERIC, May 2017. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ipac2017/papers/thpab137.pdf