ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WESH1002

NEW JAVA FRAMEWORKS FOR BUILDING NEXT GENERATION EPICS APPLICATIONS

K. Shroff, National Synchrotron Light Source II, Upton, USA
C. Rosati, G. Weiss, European Spallation Source, Lund, Sweden
K. Kasemir, E. Smith, Oak Ridge National Laboratory, Oak Ridge, USA

Abstract

Phoebus is a Java/JavaFX framework for creating state-of-the-art, next-generation desktop applications for monitoring and controlling EPICS systems. The recent developments in Java and JavaFX have made it possible to reconsider the role of the Eclipse Rich Client Platform (RCP) in the development of client applications. Phoebus's aim is to provide a simple to use and yet "rich-enough" application framework to develop modular JavaFX desktop applications for the most recent Java platform. Phoebus is an extensible framework for multiple control system protocols. It provides features for developing robust and scalable multi-threaded client applications. Key features include event rate decoupling, caching and queuing, and a common set of immutable data types to represent controls data from various protocols. The paper describes the framework as used to implement applications and service for monitoring EPICS PVs. The benefits highlighted will provide the EPICS community a new development perspective.

MOTIVATION

Since 2006 Control System Studio[1, 2, 3] has been adopted as the graphical user interface for the control systems of many Accelerators at various universities and laboratories. The growth of an active collaboration has accompanied this increase in adoption. In 2018 a survey was conducted of both the end users of the tools and applications included in the CS-Studio products as well as the developers that built and maintained them. The results of the survey highlighted a consistent set of liked and disliked aspects of the CS-Studio project.

The users generally liked the available set of applications and the integrated environment of CS-Studio allowed for effective and intuitive workflows however the product was too much like an IDE with performance and reliability issues. The developers appreciated the extensible architecture which allowed them to easily add new applications, expand the functionality of the common product, and integrate with their existing infrastructure however the learning curve was very steep. In particular, understanding the build system and OSGi life cycle presented a significant barrier to new participants. Most of the identified issues were associated with CS-Studio's dependence of the Eclipse Rich Client Platform [4] and thus the Phoebus project [5, 6] was initiated to serve as a replacement of the Eclipse RPC framework.

INTRODUCTION TO PHOEBUS

Historical Considerations

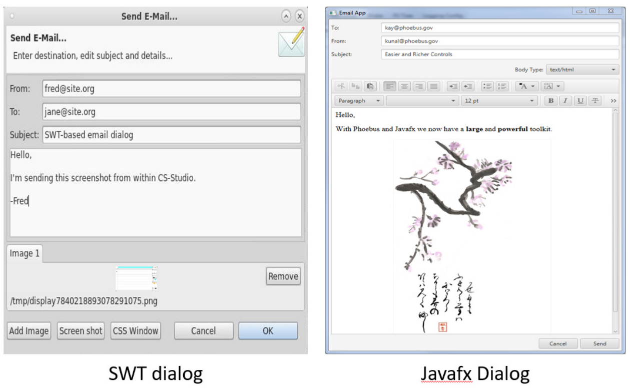
In 2004 the Eclipse rich client platform provided a great array of features for developing extensible client applications. It supported a modular architecture with support for extensibility via pluggable extension points and plugin life cycle management. It had its own build system and a workbench consisting of views, editors, and perspective. It also provided support for managing preferences, logging, native language support, and updates.

Current Situation

The benefits of using the Eclipse RCP framework are currently accompanied by some major drawbacks. The build system has increased in complexity which has made it difficult to understand and manage. For example, a complete compilation of CS-Studio from sources can take almost an hour

A lot of the features of the Eclipse RCP framework are now part of the java language itself and are better supported by a larger community. The modular architecture of Eclipse built on OSGi bundles/ Eclipse plugins could be replaced with the use of Java modules. The Java service provider interface (SPI) provided a viable alternative to the Eclipse extension points.

Additionally, the Eclipse framework ties us to the standard widget toolkit (SWT) [7], which was the best GUI library a decade ago, but is being overshadowed by JavaFX [8]. The JavaFX library, which was developed to replace swing as the standard GUI library for the java se, is a far more feature rich and better performing alternative to SWT. The library comes with a rich set of well designed easy to use widgets, supports properties bindings, CSS, etc.



Send E-Mail...
Enter destination, edit subject and details...
From: fred@site.org
To: jane@site.org
Subject: SWT-based email dialog

Hello,

I'm sending this screenshot from within CS-Studio.

-Fred

Image 1
/tmp/display7840218893078291075.png

To: kay@phoebus.gov
From: kunal@phoebus.gov
Subject: Easier and Richer Controls
Body Type: text/html
Paragraph 12 pt

Hello,

With Phoebus and Javafx we now have a **large** and **powerful** toolkit.

Figure 1: Comparison of dialogs created using SWT and JavaFX GUI libraries.

Figure 1 illustrates the significant difference in the look and feel of the same dialog developed using SWT and JavaFX.

The Path Ahead

Given the drawbacks of the Eclipse RPC framework and the improvements to java and JavaFX GUI library, the Phoebus[1] project was developed to serve as a replacement for the Eclipse RCP[3] for CS-Studio[2] application and tool development.

ARCHITECTURE OF PHOEBUS

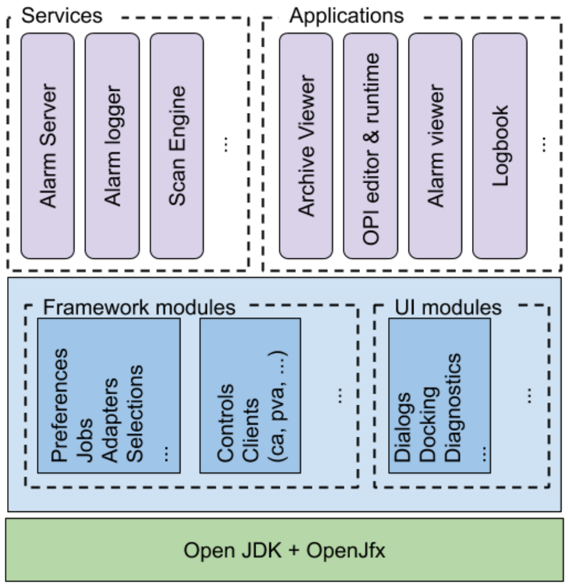


Figure 2: Architecture of Phoebus.

The Phoebus framework, as shown in Fig. 2., is organized in the following parts.

Core A set of modules containing shared services, descriptions of service provider interfaces, protocol libraries, etc. These core modules simplify application development by allowing the developer to focus only on the functionality of the application and not have to worry about implementing clients to multiple controls protocols, a preference manager, etc... The use of these core modules also optimized the use of common resources, like connection to various web services and protocol clients.

Applications(apps) Application examples include a simple "Probe" tool, an operator display panel editor and runtime, an alarm display client, logbook viewer, etc. While many of the applications are generic and interface with any EPICS-based control system, each site can also add site-specific applications to meet local needs.

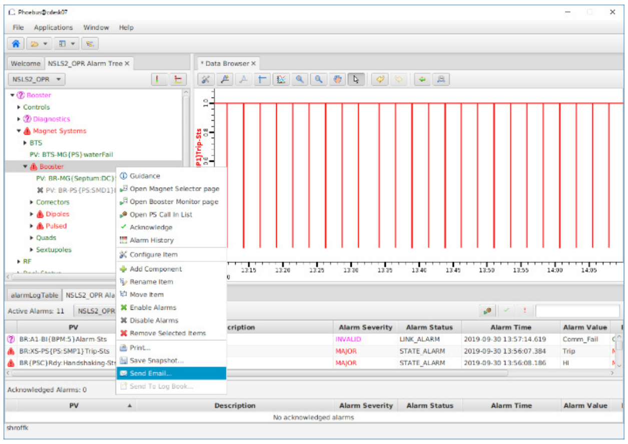
By now all essential CS-Studio functions that were previously implemented for Eclipse RCP have been translated into Phoebus applications. In this process, the Java code for application logic was often simply copied. Adjustments were required for UI code that needed to transition from SWT to JavaFX. Code that interfaced to RCP for reading preferences, scheduling background tasks and integrating into the window system needed to use the corresponding API provided by Java itself or the Phoebus core modules.

Products It describes the final packaged product that is delivered to the users. It consists of a launcher packaged with a set of applications and the core & third parties libraries needed. The common product can be easily customized to include only the desired applications along with the site specific implementation of various SPI's.

The Fig. 3 shows a screenshot of the NSLS2 CS-Studio product built using Phoebus. The product includes the alarm viewers and the databrowser which is retrieving historical data from the appliance archiver. These applications and the specific datasources they use were packaged as per the requirements of the NSLS2 controls environment and demonstration pluggable nature of the Phoebus framework. The figure also shows the context menu population with actions integrating the alarm viewer with other CS-Studio applications available like probe, logbook, databrowser, etc. The integrated environment of CS-Studio applications is preserved on the Phoebus framework.

Services A set of standalone services, like the alarm server, archiver, scan engine, performing a well defined function and with a well defined interface. While technically similar to a product, services do not contain UI elements, they are not directly accessed by end users. For example, an archive engine service is deployed by a system administrator to continuously collect and store data. End users can then access this data from a history plot like the one shown in Fig. 3.

Services can be independently deployed, and permit the use of any protocol, database, etc. These services allow for the adoption of a service oriented architecture, moving business functions into these services allows for thinner clients while also improving scalability, maintainability, and reliability of the entire system.



PV	Alarm Severity	Alarm Status	Alarm Time	Alarm Value
BR:A1-BI{BPM:5}Alarm-Sts	INVALID	LINK_ALARM	2019-09-30 13:57:14.619	Comm_Fail
BR:XS-PS{PS:SMP1}Trip-Sts	MAJOR	STATE_ALARM	2019-09-30 13:56:07.384	Trip
BR{PSC}Rdy:Handshaking-Sts	MAJOR	STATE_ALARM	2019-09-30 13:56:08.186	HI

PV	Description	Alarm Severity	Alarm Status	Alarm Time	Alarm Value
No acknowledged alarms					

Figure 3: CS-Studio built using Phoebus.

BUILD SYSTEM

The Phoebus project depends on features which are now part of the java language. This use of standard java features has resulted in a significant simplification of the project's build system in comparison to Eclipse's tycho build. The simplification is not just represented in the greatly reduced and easy to manage project metadata files and dependencies but also an order of magnitude reduction in the build time, which is now on the order of minutes. By moving away from the complexities of the Eclipse tycho build system we were also able to improve the predictability and reproducibility of the build. Currently the dependencies of Phoebus are managed via maven and the project can be built using either maven or ant. In the future we intend to move to the Java modules system and continue to transition towards using standard Java features.

TESTING

In addition to unit testing using JUnit, the simplified Phoebus framework has facilitated the creation of integration testing. The adoption of JavaFX has enabled the use of modern easy to use UI testing solutions like TestFX.

CONTINOUS INTEGRATION

Continuous integration is an essential part of any modern collaborative software development workflow. The Phoebus project uses Travis as the default CI engine triggered on any commit, pull request, and merge. In addition to Travis multiple CI solutions have been adopted to best suit the needs of different collaborators, these include Appveyor, Gitlab, and Jenkins. The easy adoption of multiple CI solutions has been possible due to the simple and standardized build system of Phoebus.

MOVING FROM ECLIPSE TO PHOEBUS

As CS-Studio applications and tools are moved from the Eclipse RCP to the Phoebus framework there is a concerted effort to ensure that this transition is made as seamless as possible. In order to allow individual sites to control the transition it is possible for a create composite product which contains applications built on both the Eclipse RCP and the Phoebus framework. Fig. 4 represents an example composite CS-Studio product.

The "org.csstudio.phoebus" plugin allows Phoebus application can be invoked from menus, toolbars, and OPI screens built on the Eclipse framework making the transition transparent to the end user.

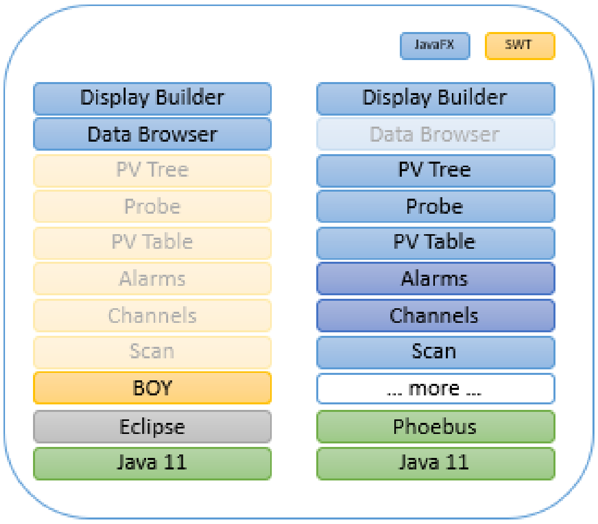


Figure 4: A composite CS-Studio product containing both Phoebus and Eclipse based applications.

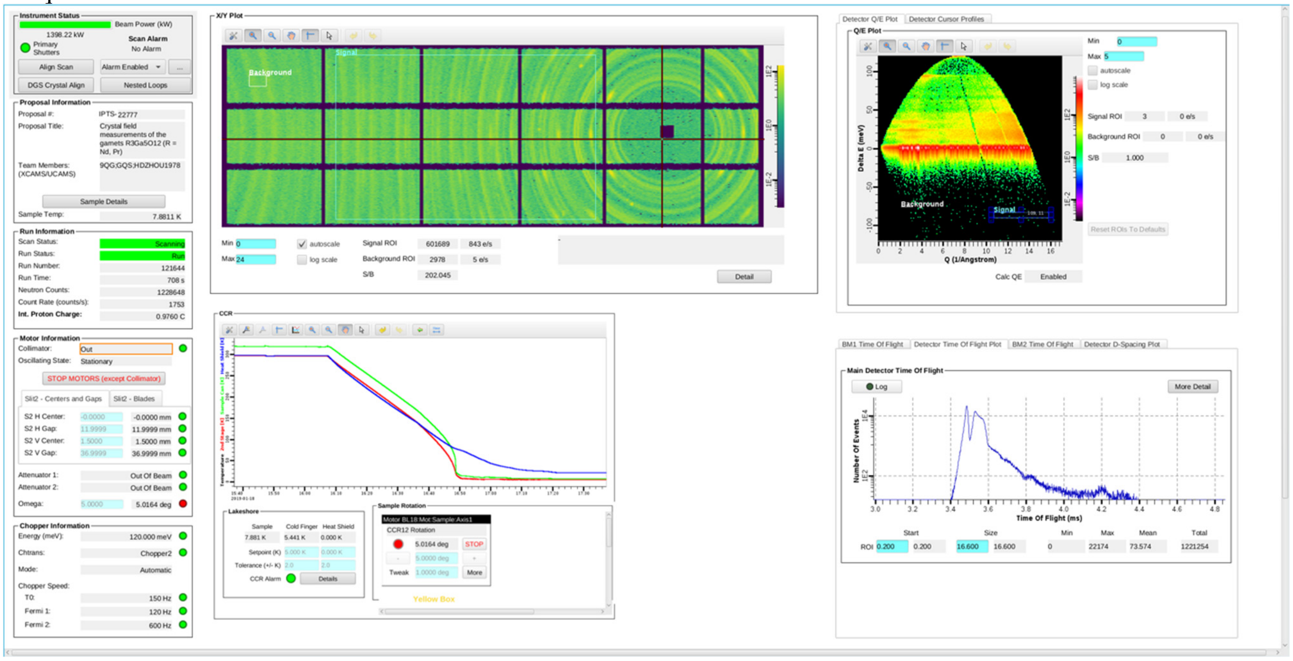


Figure 5: SNS beam line dashboard after upgrading to Phoebus.

SUMMARY

Using the latest feature of java and the JavaFX GUI library the Phoebus framework provides an effective alternative for the Eclipse RCP framework for developing CS-Studio applications and services. The framework uses standard build systems, generic testing solutions and set of core services and interfaces reduces the entry barrier for developers.

By now some deployments were able to completely transition from the Eclipse-based CS-Studio to the Phoebus platform, proving that the Phoebus project is fundamentally successful in meeting its requirements. Fig. 5 shows the SNS version of CS-Studio built on Phoebus.

For end users the best features of the CS-Studio product, like the integrated environments, are preserved while the performance and reliability are enhanced with the upgrade to JavaFX.

ACKNOWLEDGEMENTS

The efforts of the CS-Studio collaboration are responsible for the creation of the Phoebus framework. In addition to the collaborators the authors would like to thank the many users who have over the years spent a lot of time and effort to understand this tool and through their observations and feedback help guide CS-Studio and Phoebus projects.

REFERENCES

[1] CS-Studio, `http:// http://controlsystemstudio.org/`

[2] M. R. Clausen, C. H. Gerke, M. Moeller, H. R. Rickens, and J. Hatje, “Control System Studio (CSS)”, in *Proc. 11th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'07)*, Oak Ridge, TN, USA, Oct. 2007, paper MOPB03, pp. 37-39.

[3] K. Kasemir *et al*., “Control System Studio Applications”, in *Proc. 11th Int. Conf. on Accelerator and Large Experimental Physics Control Systems. (ICALEPCS ‘07)*, Knoxville, USA.

[4] Eclipse, `https://www.eclipse.org`

[5] K.-U. Kasemir, “Control System Studio Applications”, in *Proc. 11th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'07)*, Oak Ridge, TN, USA, Oct. 2007, paper ROPB02, pp. 692-694.

[6] Phoebus, `https://github.com/shroffk/phoebus`

[7] Standard Widget Toolkit, SWT, `https://www.eclipse.org/swt`

[8] JavaFX, `https://www.openjfx.io`