ADOPTING AND ADAPTING CONTROL SYSTEM STUDIO AT DIAMOND LIGHT SOURCE

M. Furseman, N. Battam, T. Cobb, I. Gillingham, M. T. Heron, G. Knap, W. Rogers,
Diamond Light Source Ltd, Oxfordshire, UK

Abstract

Since commissioning, Diamond Light Source has used the Extensible Display Manager (EDM) to provide a GUI to its EPICS-based control system. As Linux moves away from X-Windows the future of EDM is uncertain, leading to the evaluation of Control System Studio (CS-Studio) as a replacement. Diamond has a user base accustomed to the interface provided by EDM and an infrastructure designed to launch the multiple windows associated with it. CS-Studio has been adapted to provide an interface that is similar to EDM's while keeping the new features of CS-Studio available. This will allow as simple as possible a transition to be made to using CS-Studio as Diamond's user interface to EPICS. It further opens up the possibility of integrating the control system user interface with those in the Eclipse based GDA and DAWN tools which are used for data acquisition and data analysis at Diamond.

INTRODUCTION

Diamond Light Source is a third generation light source, comprising of an injection chain of 100 MeV linac, 3 GeV booster ring and a 561.6 m 3 GeV storage ring [1]. There are currently a total of 33 photon beamlines which are either completed, in construction, or planned. All control system parameters are exposed via the Experimental Physics and Industrial Control System (EPICS) [2], and GUIs for these are realised with the Extensible Display Manager (EDM) [3], which is widely used in the accelerator community for monitoring and controlling live process variables (PVs).

EXISTING INFRASTRUCTURE

At Diamond Light Source we use EDM extensively across the Accelerator complex and Photon Beamlines, which allows us to provide a common interface to many core components such as vacuum systems and motors. Typically a user will access synoptic overviews of a beamline or the machine areas from the Diamond Launcher, an application that resembles a 'start menu' with Diamond specific content. From the synoptic screen a user will access more detailed information by clicking through a series of panels.

All the panels are stored on a read only file system which contains multiple versions of releases for different modules. EDM is started from a script invoked by the launcher which sets environmental variables to point at the correct versions of dependencies for any particular synoptic screen. A new instance of EDM is run for each synoptic allowing them to depend on different modules. In total around 7000 EDM screens are installed at Diamond with many of these being auto-generated.

Not all displays in Diamond are thin clients that interact only with EPICS PVs; some require additional processing to provide the user with interactive information or need to calculate values based on a physics algorithm. It is not plausible to do these processes in EDM and is sometimes difficult to achieve them in an IOC. These are typically implemented in PyQT and interact with EPICS using the Cothread Python library [4]. Developing these GUIs is time consuming and can provide an inconsistent feel to the overall operator interface.

MOTIVATION FOR MOVING TO CONTROL SYSTEM STUDIO

While Diamond has been using EDM since it began operation, the long term prospects for the application are uncertain. As Linux distributions move away from X to Wayland, EDM will lose support for Motif, the widget toolkit it is built on. While it is plausible that EDM could be written to use a more modern toolkit, this transition provides the opportunity to look for a new GUI tool for creating operator interfaces that are based on more modern technologies and provide more features.

Control System Studio [5] (CS-Studio) is written in Java and based on the Eclipse Rich Client Platform (RCP). It provides a modular plugin architecture that allows extensions to be easily contributed in the form of plugins. This architecture has given rise to a number of features that can be bundled with the application to tie together a suite of control system tools into a single interface; an example of one of these plugins is the Data Browser, which can be used simultaneously as a replacement for the 'Striptool' and the 'Archive Viewer'. Eclipse RCP is capable of running on multiple platforms including Linux and Windows. While our control interfaces run on Linux workstations, this does remove a barrier in providing control system information to office users with Windows workstations.

CS-Studio is the choice for many new sites [6–8]. This increases confidence that the application will benefit from community maintenance in the future. Indeed, there has been an average of 7.82 commits a day to the master branch since 2007 [9].

Yet another benefit of moving to CS-Studio is that an automated conversion framework already exists for converting EDM layout files (EDL) to CS-Studio's XML based layout files (OPI). With so many EDL files automated conversion is a necessity if Diamond is to transition its entire GUI infrastructure in a reasonable time scale. The conversion framework is also easily extensible allowing developers to add unimplemented features.

ISBN 978-3-95450-148-9

On photon beamlines, Diamond uses tools developed in house that provide Generic Data Acquisition (GDA) and a Data Analysis Workbench (DAWN) [10]. These tools are developed for external users coming to site to be able to collect and analyse their data without having to understand how to interact with the underlying control system. Both applications provide a GUI which is also implemented with the Eclipse RCP; due to its modular nature it is easy to share components between applications. Using CS-Studio as an interface to the control system further affords the possibility of developing shared software.

CHALLENGES IN SWITCHING

While the decision to use CS-Studio on its merits seems valid, it was necessary to determine if there were any issues that would prevent successful adoption.

Performance

CS-Studio was expected to run more slowly than EDM, which was written to run on computers with significantly fewer resources and less power. To determine if there is a big enough difference in performance to cause issues for users, a series of tests were carried out to asses this and other issues of usability.

In general the time taken to open screens was perceived as acceptable with a few notable exceptions; screens which had been using the 'Symbol' widget in EDM were very slow to open. To work around this Diamond has implemented a custom CS-Studio Symbol widget that has much better performance, but is limited to rendering static rasterized images. Producing those images is part of the post-process conversion process which is described later. In order to achieve this speed CS-Studio does use multiple cores, but this is acceptable given the computers in use at Diamond.

CS-Studio uses considerably more memory than EDM. The Java Virtual Machine (JVM) imposes a fixed upper limit on the available memory when the application is launched. In practice CS-Studio quickly consumes all the memory available to it and then relies on Garbage Collection (GC) to free memory when required. This presents a problem only when the required memory exceeds the allowance at which point CS-Studio will consume a lot of CPU resource before eventually crashing the JVM. This is satisfactorily managed by setting an appropriate value for the memory usage; 2 GB is sufficient for most use cases at Diamond.

A machine operator's shift will often cover 8 hours of continuous GUI use. Due to the complexity and time critical nature of machine control it is not acceptable for the application to crash during this period. Long duration tests have been completed that give us confidence that an instance of CS-Studio can be left running over multiple days without crashing or slowing down. CS-Studio does crash occasionally, however these crashes do not seem to occur more frequently than were experienced with EDM.

Technology and Interface

CS-Studio is based on the Eclipse RCP, which is itself built on a number of other technologies which support the modular plugin architecture such as OSGi. This framework is powerful, but it is also dense and complex making it difficult to understand. Attempting to use the framework to do things outside the Eclipse paradigm is tricky. The Maven [11] based build system also provides those who want to do command line builds of CS-Studio with a steep learning curve.

As SWT is responsible for the widget toolkit in Eclipse RCP, the overall look and feel of the application is defined by the native widgets that it is built upon. Many of the widgets available for users to create their own displays in CS-Studio are implemented in Draw2D and therefore do not share the same appearance as the framework that surrounds them. This has the advantage that displays look very similar on different platforms, but the disadvantage that panels may not be consistent with native applications.

So far, none of the issues described have been sufficient to prevent Diamond's adoption of CS-Studio. However we have many users of EDM whose workflow uses stand-alone windows that are handled by the Linux window manager. CS-Studio presents users with a workbench, a single large window with tiled panes that can contain other windows as shown in Fig. 1. As we are automatically converting our panels they will still retain the EDM layout, which has been designed to be used as a proliferation of many small panels. While it is possible to detach one of these panels in CS-Studio, the new window behaves differently to the EDM windows we have now. There are distinct differences in behaviour with regard to Linux virtual desktops; these are heavily used in the standard workflow of the Diamond operators. The workbench based layout of CS-Studio is also restrictive to those who wish to move panels across multiple physical displays. These limitations, combined with the poor display of many of our existing screens due to their small size, present an issue that needs to be resolved before Diamond could adopt CS-Studio for our operator interfaces.

MODIFICATIONS TO CS-STUDIO

The main purpose behind modifying CS-Studio was to present an interface that would be familiar enough to our users that they could transition from EDM painlessly. To do this the core modification we have made is the ability to display CS-Studio panels in an SWT Shell, which is not integrated into the Eclipse window and so can be manipulated in the way we require. These changes have now been merged into the master branch of CS-Studio and will be available following the 4.2 release [12]. The changes provide users of CS-Studio with the ability to choose a 'Standalone Window' either when right clicking on an OPI file or creating screens which open related displays.

Although not directly a part of Eclipse RCP, it is still possible to tie elements such as the right click menu to the standalone window, allowing it to interact with the workbench window. This is useful if, for instance, a user wishes

ISBN 978-3-95450-148-9

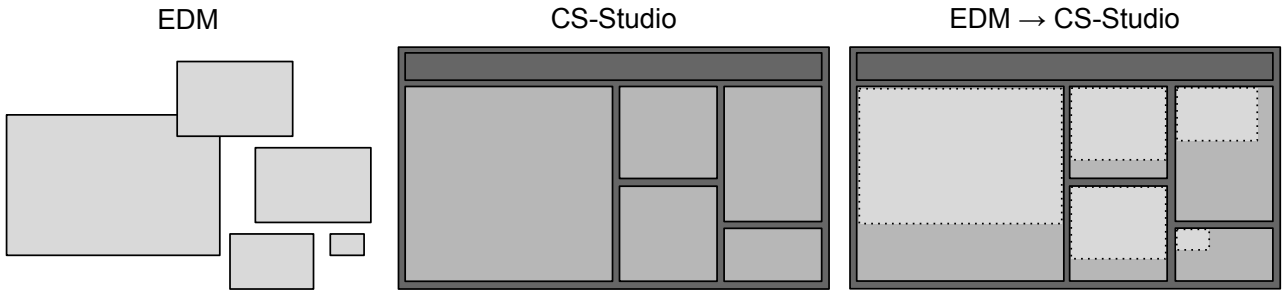

Figure 1: Converted EDM panels do not fit optimally into the default CS-Studio workbench.

to see more detailed information about a process variable that would normally open in the workbench.

There remain a few issues with the standalone window framework as it exists, and we hope to find mutually acceptable solutions to these for the variety of use cases CS-Studio enjoys at different sites. It is possible to open workbench based operator panels and also standalone panels in the same instance of CS-Studio. This could be confusing for a user who may not understand why two windows which appear similar behave in different ways. One possible solution to this is to restrict all open panels to either the view or shell-based method based on a runtime preference. Another solution is to contribute the shell as an optional 'feature' which allows maintainers of CS-Studio at each site to decide if they would like this functionality when building their Eclipse product.

CONVERSION OF EDM SCREENS

Our entire transition process is based around the ability to run existing EDM screens in CS-Studio without significant human intervention. To do this we require a good quality automated conversion from EDL files to the new OPI file format. CS-Studio includes a framework for a widget-by-widget conversion of EDM files, which we have extended to support more widgets and improve accuracy. Some logic contained in EDM screens cannot be reproduced by this method, so we have written post-processing scripts to alter the produced XML files. We have also created two specific EDM compatibility widgets to extend the CS-Studio toolset.

Symbol Widget

The Symbol widget shows a section of embedded panel depending on the value of a PV. Diamond has a site wide temperature monitoring GUI that has 1605 symbol widgets. With EDM the performance of these panels is acceptable, however with CS-Studio it is exceptionally poor. By creating a custom widget able to show sections of a rasterized image we have been able to demonstrate significantly better performance. There are a few edge cases that do not work with this approach, e.g. animated symbols, or embedded symbols that are very different resolutions to their containers, but for the vast majority of symbols this widget works well.

Menu Mux Widget

The Menu Mux widget in EDM changes one or more dynamic macro values in response to a user's control value selection. A new widget, based on the SWT combobox, was created for CS-Studio which offers similar functionality using local PVs. CS-Studio does not support dynamic macro values so these must be substituted with local PVs. The converter is unable to determine which macros referenced in a Menu Mux widget need to be changed to local PVs so they can be used by other controls on the OPI, therefore this update must be performed as part of a whole-file post-process step.

Post-Processing

Due to the difference in behaviour between CS-Studio and EDM not all widgets can be converted directly; their context in the panel determines how they should be converted. Any part of the conversion that has knowledge about the interaction between widgets is not supported by the CS-Studio converter so we have implemented this in a post processing Python script. As well as the Symbol and Menu Mux widgets there are a few other differences in behaviour we address with post processing: CS-Studio requires that a widget with a click action be above all other widgets in order to receive the click event; it also clips widgets that lie outside of a grouping container. We solve these by adding an invisible click widget on the top of the stack, and resizing the grouping container to fit the widgets respectively.

FAST ACQUISITION DATA-BROWSER PLUGIN

Diamond has a fast acquisition (FA) data stream that records data from 255 BPMs at 10 kHz into a 30 TB ring buffer. The data is accessed via a socket with clients request decimated data in bins of either raw, 64 or 16384 decimations [13]. To get this data into CS-Studio a FA data source was added which allows us to query the status of live BPM data by simulating a PV as well as historic archived data. The decimated data is returned with mean, minimum and maximum values, the display of which is supported by Data Browser. Using this setup it is now trivial to plot slow acquisition EPICS data alongside fast acquisition data, something

ISBN 978-3-95450-148-9

that is very useful when determining how fast beam motion is affected by other parameters.

BEAMLINE USERS

On photon beamlines workstations Diamond hosts a variety of internal and external scientists with a range of experience using controls GUIs. Often EDM's usage paradigm can create a confusing proliferation of windows which are difficult to organise once they cover the available monitor real estate, especially as a beamline typically wants to control the state of multiple items of hardware simultaneously. For this reason we are currently developing a beamline GUI framework that takes advantage of CS-Studio's ability to control the layout of panels in line with the more conventional way of using an Eclipse RCP based application. Shared components between photon beamlines and accelerators such as vacuum and motors will be required to work appropriately in either usage paradigm.

PLANNED CS-STUDIO INFRASTRUCTURE

Any production release used for controls at Diamond must be built from a script without an internet connection, ensuring repeatable builds in the future. It is not practical or even permissible to download all the p2 repositories that the build depends on due to their large size. Instead, this has been achieved by archiving a clean Maven repository used for the build in order to gather the relevant dependencies. This results in a minimal set of dependencies required for an offline build.

All production modules in Diamond are placed in a read only file system with versioned releases. Currently module maintainers manage environmental variables which allow EDM to pick up different versions of EDL files they have dependencies on; each set of dependencies requires a new instance of EDM. Due to the large footprint when running CS-Studio we can not start a new instance for every set of dependencies, fortunately Eclipse provides a method of redirecting paths within the RCP environment to filesystem paths which are called 'share_links'. Part of our modifications to CS-Studio have been allowing additional command line parameters to modify the links in a running instance of CS-Studio, thereby allowing a module owner to keep a list of dependencies which is automatically updated each time a user opens a newly released panel.

INTEGRATION WITH DAWN AND GDA

Running CS-Studio on photon beamline workstations opens a significant avenue of collaboration opportunities with the other software groups at Diamond who are also developing Eclipse RCP applications for use on data acquisition and analysis [14]. At present the Diamond build of CS-Studio contains a version of the graphing widget used in DAWN which has been integrated for use in CS-Studio displays.

ACKNOWLEDGMENTS

The authors would like to thank Friederike Jöhlinger for building the fast archiver plugin as a summer placement project. We would also like to thank the CS-Studio collaboration for their receptiveness to the changes we have made to their application.

REFERENCES

[1] R. P. Walker, "Commissioning and Status of the Diamond Storage Ring", APAC (2007).

[2] M. T. Heron et al., "The Diamond Light Source Control System", EPAC (2006).

[3] J. Sinclair, `http://ics-web.sns.ornl.gov/edm/`.

[4] M. G. Abbott et al., "Diverse Uses of Python at Diamond", PCAPAC (2008).

[5] J. Hatje et al., "Control System Studio (CSS)", ICALEPCS (2007).

[6] F. Arnaud et al., "ITER Contribution to Control System Studio (CSS) Development Effort", ICALEPCS (2013).

[7] M. Giacchini et al., "The Control System of Spes Target: Current Status and Perspectives", ICALEPCS (2009).

[8] T. Satogata et al., "ESS Controls Strategy and the Control Box Concept", PCaPAC (2010).

[9] `https://github.com/ControlSystemStudio/cs-studio/graphs/contributors`

[10] M. Basham et al., J. Synchrotron Rad. **22**, 853-8, (2015).

[11] `https://maven.apache.org/`.

[12] `https://github.com/ControlSystemStudio/cs-studio/pull/1066`

[13] M. G. Abbott, "A New Fast Data Logger and Viewer at Diamond: the FA Archiver", ICALEPCS (2011).

[14] M. W. Gerring, "Open Source Contributions and Using Osgi Bundles at Diamond Light Source", WEB3O01, *these proceedings*, ICALEPCS'2015, Melbourne, Australia (2015).

ISBN 978-3-95450-148-9