ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THSH303

CS-STUDIO DISPLAY BUILDER*

K.-U. Kasemir, Megan Grodowitz, Oak Ridge National Laboratory, Oak Ridge, TN, USA

Abstract

The Display Builder started as a comprehensive update to the Control System Studio "BOY" panel editor and runtime.

The design was changed to a modular approach, separating the model of widgets and their properties from the graphical representation and the runtime. The model is fully multithreaded. The representation has been demonstrated in both SWT and JavaFX, for now intending to concentrate on the latter. The runtime, based on the thread-safe model, avoids user thread delays and improves overall performance for complex widgets like images as well as scripts and rules.

We present the current state of the development and initial deployments at beam lines of the Oak Ridge National Laboratory Spallation Neutron Source (SNS).

MOTIVATION

Control System Studio (CS-Studio) is a collection of control system tools [1,2]. Its most visible component to end users is typically the operator interface panel builder known as "BOY" [3]. Its versatility has led to the adoption of CS-Studio at several sites, sometimes replacing existing display technologies [4]. At SNS beam lines, the BOY support for scripting languages allows good integration between interactive and automated experiment control [5].

After about a decade of successfully using BOY at the SNS, we started to recognize limitations in the underlying architecture. Both the BOY display editor and the runtime are based on the Eclipse Graphical Editor Framework (GEF) [6]. While this greatly accelerated the initial development of BOY, it ties the software to the SWT [7] graphics library. Furthermore, the BOY widget model can only be accessed on the user interface (UI) thread, in part because of its tight integration with GEF and SWT, which limits access to widget properties to the UI thread.

Opening a new display file that resides on a busy file server may be slow. While scripts are very convenient to implement and update, their execution speed is often limited. In the BOY software architecture, the loading of displays and the execution of scripts needs to be handled on the UI thread, which can cause the whole user interface to temporarily stop updating, which end users then experience as a "freeze-up".

DISPLAY BUILDER DESIGN

The Display Builder is meant to offer the same basic functionality as BOY, but in a modified software design.

Widget Model

The Display Builder Widget Model is a description of all widgets and their properties. At this time, we implemented 42 widget types, categorized as follows.

- Graphic widgets that display static content: Label, Rectangle, Polyline, Picture, ...
- Monitor widgets that display the value of a control system Process Variable (PV): Text Update, LED, Meter, …
- Control widgets that allow the user to modify the value of a PV: Text Entry, Knob, various buttons, …
- Plot widgets that show one or more PVs, including waveforms in various ways: X/Y Plot, Image, Data Browser
- Structure widgets that are used to arrange widgets within a display: Array, Group, Embedded Display, Navigation Tabs
- Miscellaneous widgets: Web Browser

The Widget Model is fully thread-safe, that is multiple threads can for example concurrently update the "text" and "foreground color" properties of a "Label" widget. Software that has subscribed to widget property changes receives immediate notification of updates. The widget model is not tied to any specific graphics library. This includes the description of colors or fonts, for which the Display Model implemented its own data objects to avoid outside dependencies.

Widget Representation

The Widget Representation module of the Display Builder renders the Widget Model with a specific graphics library. There are currently two implementations.

One is based on SWT, as in BOY. The SWT implementation is meant to demonstrate that our design is not tied to a specific graphics library, but it is limited to very few widget types and will at this time not be extended.

The second implementation is based on the newer JavaFX technology [8]. All widgets are fully supported by this representation, which is our emphasis for the foreseeable future of the Display Builder.

Widget Editor

The Widget Editor shown in Fig. 1 allows users to interactively create and modify displays by positioning widgets and adjusting their properties.

A widget is added to a display by dragging it from the widget palette into the display, and then adjusting its properties. The "Properties" lists all properties of a widget, grouped into sections. Key properties tend to be listed at the start of the list, including for example the position and PV Name, which are often the only properties that

* ACKNOWLEDGMENT OF FUNDING
This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under contract number DE-AC05-00OR22725.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THSH303

need to be adjusted. Widget properties are listed in a consistent, meaningful order. For instance, the position is always presented as "X", "Y", "Width" and "Height". Widgets with "Foreground" and "Background" colors will always show them in this order.

The position and size of a widget can also be adjusted by moving the widget within the display with the computer mouse. The primary PV name of a widget, or for Label widgets the text, can alternatively be entered by double-clicking the widget, which opens in in-place editor for the text respectively PV name.

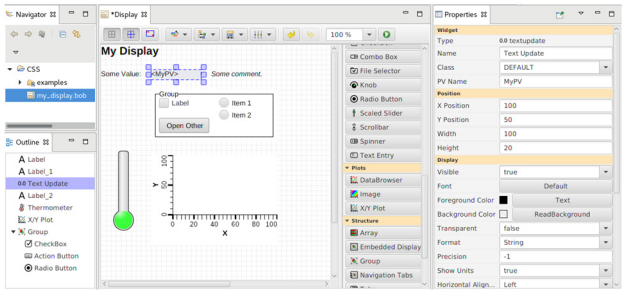

Figure 1: Widget Editor.

Widget Runtime

The Runtime connects widgets which depend on PVs to the control system, and dispatches value updates received from the control system to the associated widget properties. The Runtime instantiates script engines, currently supporting JavaScript as well as Jython and Python, for widgets that use scripts.

The Runtime is prepared for the latest EPICS V4 [9] data types, including the image data type, which is displayed in the associated Image widget shown in Fig. 2.

Multithreading

When opening a display file, the actual file access, parsing of the content and the creation of the Display Model are all handled in background threads. Compared to BOY, this is especially efficient for modular displays where a top-level display contains several embedded displays. With BOY, all CS-Studio display updates would be suppressed while the top display and then each of the embedded displays are sequentially loaded, until finally the complete display can be represented.

In the Display Builder architecture, the top display is opened in a background thread. Once its main-level widgets have been loaded, they are represented on the screen. Now the content of all embedded displays is loaded in parallel background threads. They are represented on the screen as their information is parsed.

When for example the update of a numeric value is received from the control system, this data may need to be converted into a string, or it may trigger the execution of a script, and finally the result is displayed on the computer screen. The handling of PV value updates as well as the execution of scripts is performed in background threads against the thread-safe Model. The UI thread is only used as necessary by the Representation when it performs the actual user interface update. The Representation further throttles screen updates to reduce the likelihood of user interface "freeze-up".

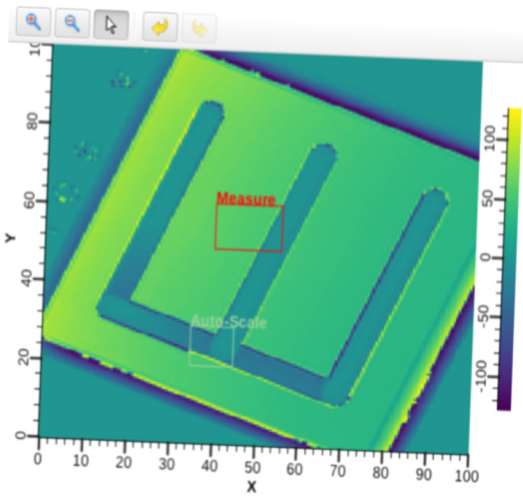

Figure 2: Widget displaying EPICS V4 Image.

Macros

The Display Builder supports macros. As in BOY, macros can be used for text, including the PV Name, Label or Tool Tip of a widget. In addition, macros are now supported for numeric or boolean widget properties. For example, the width of a widget can be based on a macro.

The values for macros can be set as global preferences, or they can be defined in a display, for example in the properties of a "Button" widget that opens a display. As a fallback, environment variables are read as macro definitions, allowing for example "$(USER)" as a macro to fetch the name of the current user on Linux systems.

Widget Classes

A widget class defines a set of widget properties. For example, a "COMMENT" class for the Label widget might be defined as having its font set to a smaller size, while a "TITLE" class for the Label widget can be defined to have a larger font, a certain color and position that is fixed to the upper-left corner of the display.

When editing a display and adding a Label widget, the user can decide to either directly configure the font property of the Label, or to select one of the predefined Label classes, which would set the font, position etc. based on the class definition.

BOY Compatibility

The Display Builder reads and executes most BOY displays without any changes. The widget set and their properties are very similar, but have been refined. Instead of the one LED widget in BOY, the Display Builder provides a basic on/off type LED and a separate Multi-State LED widget, since their configuration details are quite different with respect to addressing bits in PV values. At the same time, the BOY Rectangle and Rounded Rectan-

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THSH303

gle widgets have been joined into one Rectangle widget, with corner radius simply becoming a rectangle property. Existing *.opi displays with LED or Rectangle widgets are automatically translated into the updated format.

Compatibility is limited for plotting widgets like the XY Plot and Image widgets. The new XY Plot widget supports waveforms, including error information. Legacy displays that used the XY Plot for scalar data over time need to be updated to the Data Browser widget.

Finally, while BOY displays that used rules are often translated without problems, legacy scripts need to be updated because the underlying programming model has changed.

In case of incompatibilities, the original BOY *.opi file is loaded in the Display Builder editor, manually adjusted, and saved under the new *.bob file extension. When the Display Builder runtime opens a file, it will always check for both the legacy *.opi and the new *.bob file extension, preferring the latter. Existing display file hierarchies can thus for the most part remain unchanged, only adding new *.bob files as necessary when the existing *.opi files are not sufficient. This is especially useful during transition periods where BOY continues to be used, while the Display Builder is phased in, because the existing *.opi remain unchanged.

CS-Studio Integration

While the Display Builder editor and runtime can be executed as standalone tools, they are also fully integrated in CS-Studio. For example, when the display builder runtime executes a display file within CS-Studio, the context menu allows users to inspect displayed PVs in other CS-Studio tools.

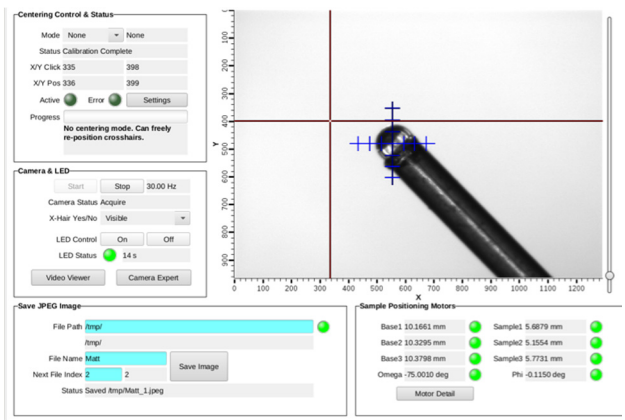

Figure 3: Sample Centering.

A drawback of the CS-Studio integration is currently the need to host the JavaFX graphics within an SWT canvas, which slightly reduces the achievable update rates compared to the standalone versions.

USE AT SNS

The Display Builder is operational on 5 SNS beam lines, and is the designated tool for SNS beam lines from now on. The displays for 8 other SNS beam lines can be viewed in the Display Builder, but their operation still depends on BOY until scripts have been converted.

Several beam line applications benefit from the enhanced plotting and image support of the Display Builder. Fig. 3 shows the Display Builder interface for a sample centering application. An image widget displays a camera snapshot of the sample which is mounted on the end of a stick. Users select a desired point on the sample by clicking into the image. The coordinates are written to PVs which trigger motor moves to position the selected sample spot in the center of the beam.

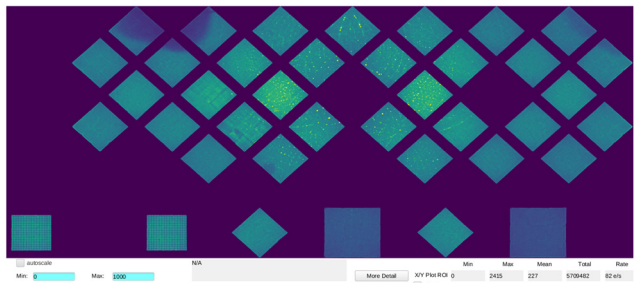

Figure 4: Detector Histogram.

Fig. 4 shows a detector histogram. The histogram image, sized about 3000 x 1600 points, exceeds the physical screen resolution. When receiving updated data, the Display Builder prepares the actual image, using the desired color map, in a background thread, and finally presents it on the screen. Users can interactively zoom into individual pixels of the histogram. Clicking on a pixel presents detail about the location of that detector pixel with respect to the sample, the scattering angle and more.

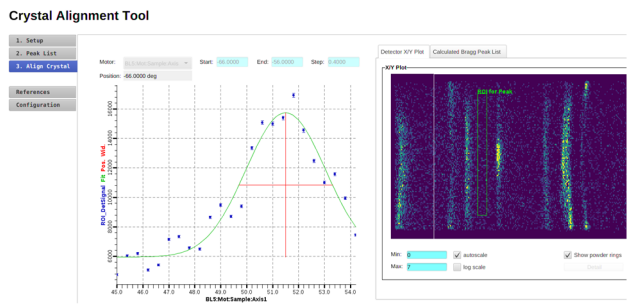

Figure 5: Example of Experiment Setup Tool.

The use of the Display Builder on SNS beam lines grew from simply interacting with the beam line control system, i.e. from commanding motors and displaying detector counts, to providing tools for experiment planning. Fig. 5 shows how users are guided through several steps of preparing their experiment, in this example by predicting expected Bragg peaks, then aligning the sample based on detected peaks. Such step-by-step planning tools benefit from the Navigation Tabs widget, a container widget that allows users to navigate between sub-displays within a page.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THSH303

SUMMARY

The Display Builder is a comprehensive update of the CS-Studio BOY. Initial use on SNS beamlines has proven to be successful. Many existing displays continue to be functional because of excellent compatibility of the Display Builder to BOY, and new types of displays, especially for detectors, have now become possible.

ACKNOWLEDGMENT

We thank the original developer of BOY, Xihui Chen, for the excellent set of features and ideas upon which we could base the new Display Builder architecture.

We thank part-time SNS team member Amanda Carpenter for her contributions to the Display Builder implementation, and Claudio Rosati from the European Spallation Source for the ongoing collaboration. We thank the members of the SNS Instrument Data Acquisition and Controls group as well as SNS beam line personnel for their acceptance and feedback.

REFERENCES

[1] J. Hatje et al, "Control System Studio", ICALEPCS, Knoxville, USA, 2007

[2] K. Kasemir, "Control System Studio Applications", ICALEPCS, Knoxville, USA, 2007

[3] X. Chen, K. Kasemir, "BOY, a modern graphical operator interface editor and runtime", ICALEPCS, New York, USA, 2011

[4] M. Furseman et al, "Adopting and Adapting Control System Studio at Diamond Light Source", ICALEPCS, Melbourne, Australia, 2015

[5] K. Kasemir, M. Pearson, "CS-Studio Scan System Parallelization", ICALEPCS, Melbourne, Australia, 2015

[6] Eclipse Graphical Editor Framework, GEF, http://www.eclipse.org/gef

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

[8] M. Pawlan, "What is JavaFX", Oracle Java Documentation, http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[9] L. Dalesio et al, "EPICS V4 Expands Support to Physics Application, Data Acquisition, and Data Analysis", ICALEPCS, Grenoble, France, 2011.