SYNOPTIC DISPLAYS AND RAPID VISUAL APPLICATION DEVELOPMENT*

B. Frak, K. Brown, T. D'Ottavio, M. Harvey S. Nemesure, Brookhaven National Laboratory, Upton, U.S.A.

Abstract

For a number of years there has been an increasing desire to adopt a synoptic display suite within BNL accelerator community. Initial interest in the precursors to the modern display suites like MEDM quickly fizzled out as our users found them aesthetically unappealing and cumbersome to use. Subsequent attempts to adopt Control System Studio (CSS) [1] also fell short when work on the abstraction bridge between CSS and our control system stalled and was eventually abandoned. Most recently, we tested the open source version of a Synoptic Display developed at Fermilab [2]. It, like its previously evaluated predecessors, also seemed rough around the edges, however a few implementation details made it more appealing than every single previously mentioned solution and after a brief evaluation we settled on Synoptic as our display suite of choice. This paper describes this adoption process and goes into details on several key changes and improvements made to the original implementation – a few of which made us rethink how we want to use this tool in the future.

SYNOPTIC IMPROVMENTS

Synoptic Display received a number of runtime and design-time enhancements, which allow us to use it in a conventional fashion – as a synoptic display - as well as a replacement for simpler, custom written applications. These improvements range from simple additions, which ease the display development process, to fundamental overhauls, which significantly change the way, both runtime and design-time mechanisms behave. The following sections outline the most significant of those changes.

Property IO

The most important addition to the Synoptic Display was the way the data is treated as it flows between components. In the original design, data context is bound to the input and output pins, which serve as the end points for component data IO. These pins have neither context nor any other additional discernable properties, which makes components one-dimensional. Furthermore, it makes their state management difficult without tacking on additional information to the existing value transfer objects. This approach is inadvisable, because more generic, simpler data constructs promote component reusability. Their complex counterparts usually tie in with a select group of components.

In order to allow for more elaborate displays, which mimic behavior of custom written applications, components have to be able to differentiate between different data properties and act accordingly based on their content. The implementation of this feature called for a change in the Pin and Property component configuration. The former construct received one additional member called function, which can be one of the following two values 1) Data – default, old behavior, 2) Property – new behavior, property data. Component properties, which drive the context for the IO pins also received additional parameters. These include:

- Scope, which can be set to design-time, runtime or both. Most properties will fall into design-time or both design and runtime categories. An example of a runtime only property is a "Trigger" property, which acts on a boolean input only at runtime and would provide no additional value during a display design process.
- IO, which can be set to input, output or both. This parameter defines the types of pins this parent property can be applied to. "Trigger" mentioned in the point above would be input only.

Figure 1 shows an example of a simple display, which showcases the core concepts of property IO. Two pieces of data are acquired from the control system. One of them is the current value of a device object and the other one are the legal values associated with the fetched value property. These are fed to the two property inputs in the List component – "Values" pin receives the list of all possible values and displays them in the list, while the "Selection" pin receives the current selection. Additional "Filter" input is used to filter out unwanted options from the List component. This pin is fed from a data output of an Input Field.

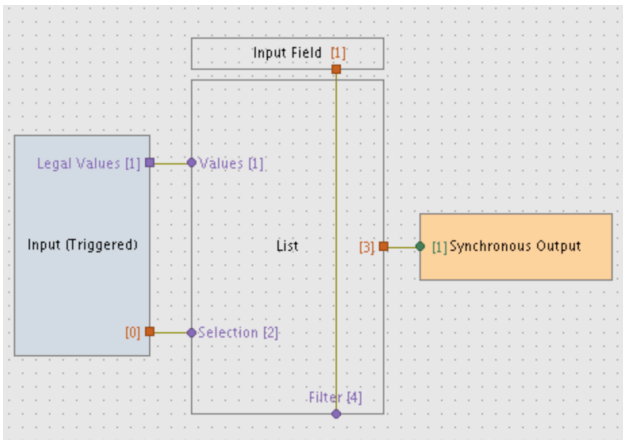

Figure 1: Property IO example.

*Work supported by Brookhaven Science Associates, LLC under contract no. DE-AC02-98CH10886 with the U.S.

ISBN 978-3-95450-139-7

Embedded Components & Namespaces

Although the original design supported embedded components, we found them lacking and unsuitable for abstraction of our control system. One key feature that was missing was targeting specific components in child displays from either its siblings or a parent container and vice versa.

The solution comes in a form of global transfer components and associated namespaces, which are now part of a property set of every embeddable component, and which can send and receive arbitrary pieces of data between every display level. Synoptic Display designer routes data from either a root or an embedded display to a "Global Property" component, which via a design/runtime property targets another component property with a matching namespace and name. Figure 2 shows property sheet editor of a "Global Property" instance, which routes incoming data to a component property aliased to "comboboxSelection" located in an embedded display with a "tab1" namespace.

Category: All

Property Name	Value	Global	Alias
X	217	☐	x
Y	10	☐	y
Width	80	☐	width
Height	20	☐	height
Property Name	tab1:comboboxSelection	☐	propertyName

Figure 2: Global properties.

Additional improvements to the embeddable Synoptic subsystem come in a form of additional components, which now include: 1) Tab Layout, 2) Split Pane and 3) Original Embedded Display.

Component Palette

Synoptic display component palette has been supplemented by dozens of brand new widgets and runtime components and component families.

- 2D and 3D Chart widgets based on the JClass [3] charting framework. These terminating components can be prepended with a specialized DataSet components, which attach additional design or runtime properties to the outgoing data, such as color, marker style, line width, visibility or axis association.
- Image display and processing widgets, which include both UI components for displaying static images in various formats as well as a number of filter components, which when chained together transform various properties of the transferred image data. Figure 3 shows an example of image filtering. The top image viewer widget shows an unmodified image read from a file, while the bottom one shows the same image, which had been post filtered (color inverted and vertically flipped).

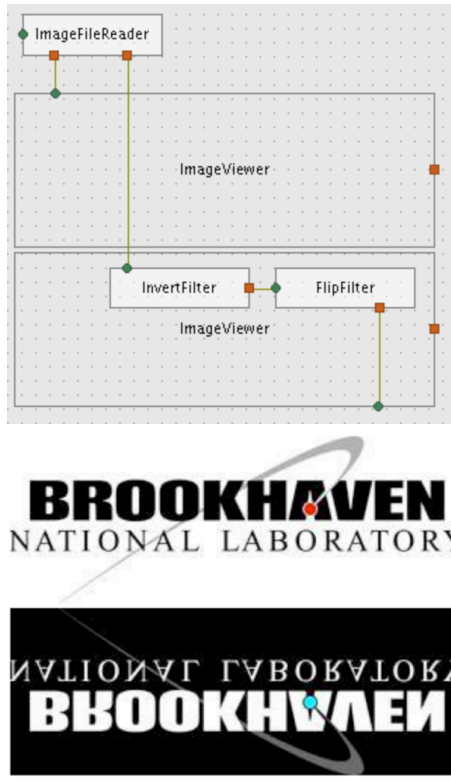

Figure 3: Image filtering.

- Data Acquisition components, which were designed from scratch with the Collider Accelerator control system in mind. These components are responsible for data I/O to and from device objects and logging system.
- Scripting components based on Java Scripting Engine [4]. These widgets come in Javascript and Python flavors and feature multiplexed pin I/O, which allows designers to feed and extract multiple named results from the script components.
- UI widgets, which have been supplemented by number of additional components such as tab panes, progress bar, split panes and many more.
- Data widgets, which have been split into several subcategories - each dealing with a specific data type, such as arrays, strings, booleans and generic types. Figure 4 shows three example array specific components: 1) Array Length, 2) Array Index and 3) Array Min and Max.

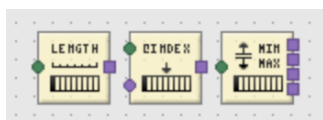

Figure 4: Example array data components.

- Synoptic display widgets, such as gauges. Figure 5 shows an example of a new circular gauge.

ISBN 978-3-95450-139-7

Figure 5: Circular gauge.

Builder Enhancements

Synoptic builder has also received a number of updates, which fall into two categories: general user experience and control system integration. The former are designed to ease display design process - particularly for complex displays, while the latter incorporate existing tools and features into Synoptic display making it easier for seasoned CAD personnel to transition into this new environment. Key amendments include:

- Layer support, which allows users to place components inside lockable and hide-able layers. This feature supplements the original "Hide Invisible" command, which allowed users to show/hide components, which have no user interface at runtime. Figure 6 shows a layer window for the Elens power supply control synoptic display.
- Device selector tool, which enables users to quickly associate control system device names with the data acquisition components.
- Alignment commands, which promote per pixel accurate component location and size mapping.
- Collider Accelerator specific file repository.

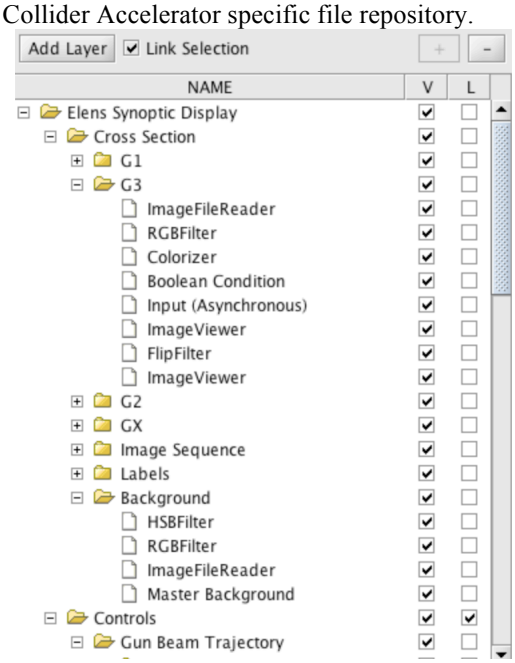

Figure 6: Layer window.

EXAMPLES

Synoptic Display's open architecture makes it a great tool for almost any UI project, which integrates with the control system. It works well for both simple feedback type displays as well as complex projects, which require stateful representation of remote objects. Even though Collider Accelerator development began only nine months ago, it has already been successfully used in several projects.

- EBIS synoptic display suite, which features tabbed environments, transparent buttons (to allow a portion of an underlying image to act as the button graphic, or a part of it), the passing of global variables over tabbed spaces, the ability to invoke operational sequences, and functions to allow display features to change with parameter range limits. The EBIS synoptic display application displays a top view of all the EBIS systems, with links to parametric display pages (pet pages), and summaries of various system parameters. Each tab takes the user to specific subsystems, such as vacuum, instrumentation, or ion source specific interfaces.
- Elens Power Supply Control (Fig. 7) - a fairly complicated display, which utilizes image filtering features to overlay and map power supply current measurements as color value on the instrument cross-section. It also features an animated beam, which was implemented using transparent image sequences. The bottom part of the display is made up of around 50 control points, which receive and send data from/to various UI widgets. The sheer number of pins and widgets, which overlay on top of each other in this display, had forced us to add layer support to Synoptic Display to categorize and compartmentalize components into logical groups.
- NSRL XDAS Control – an application style display, which allows users to controls and visualize data from individual SWIC like devices. This display makes heavy use of both 2D and 3D charting widgets. It also has multiple modes of operations, which are implemented as groups of toggle-able widgets.
- AGS RF Cavity Tuning is another example of an application style display. In the AGS there are 10 RF cavities, all are controlled independently but by identical means. This repetitive behavior exploits the use of Global Property/Namespace enhancements, making what would be an extremely complicated display very simple.

ISBN 978-3-95450-139-7

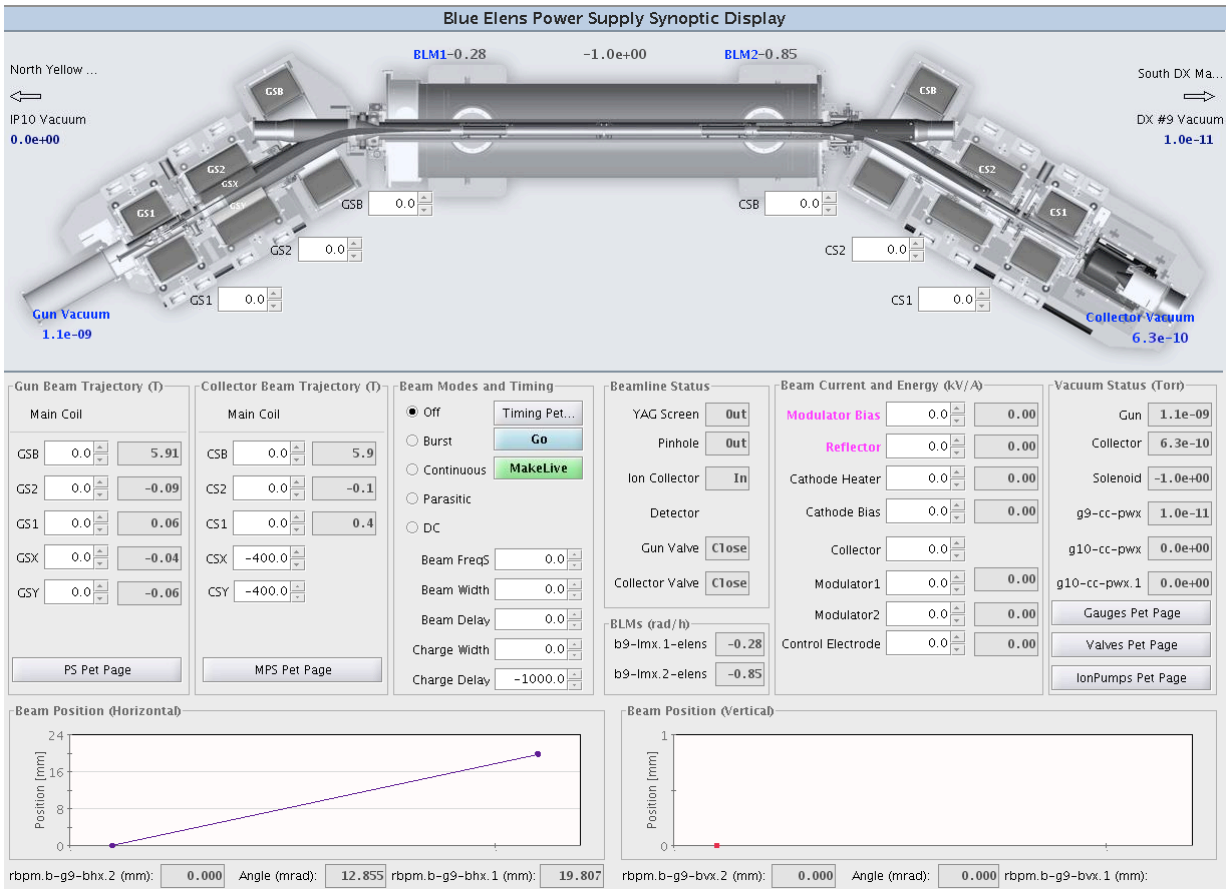

Figure 7: Elens synoptic display.

SUMMARY

Given a relatively modest amount of time spent on the controls integration and additional feature development, Synoptic Display deployment can already be declared as a successful venture. It has been used by both engineers from various groups and operators to develop both trivial as well as complex displays. Of course compared to other tools used around the Collider Accelerator Department it has still relatively low penetration – nevertheless we are optimistic about its future.

As we are gauging personnel interest in this tool, we are also thinking about potential improvements, which would stimulate its further growth. Here's a laundry list of upcoming Synoptic related projects:

- Web deployment with offline rendering for non-SVG components.
- Integration with existing Java applications, where Synoptic displays can be embedded within existing Swing applications. This integration will not be skin-deep. We are planning on implementing full data pathways and in and out of those displays.
- Additional high quality visualization widgets such as linear gauges, valves and thermometers.
- More layout and UI placement options. Right now in addition to the standard fixed layout, Synoptic Display supports container relative layouts on all four sides of the widget. We want to supplement this feature by adding more options, such as sibling relative layout.
- Improved error handling. Synoptic Display Viewer has already been supplemented with a message area, which displays all runtime errors. We want to improve on that by adding error contexts to individual widgets.

Synoptic Display is not an end all solution to all problems facing developers and engineers, who are building new applications for the accelerator complex. In cases where clients have to handle the bulk of the "business logic", it's downright unsuitable. On the other hand, it works great for projects with a middleware layer, which offloads the intricate logic from the user facing client tier.

REFERENCES

[1] http://controlsystemstudio.github.io/
[2] T. Bolshakov, A. Petrov and S. Lackey Synoptic Display – A Client-Server System for Graphical Data Representation, ICALEPCS (2003).
[3] http://www.quest.com/jclass/
[4] http://docs.oracle.com/javase/6/docs/technotes/guides/scripting/programmer_guide/

ISBN 978-3-95450-139-7