REDESIGN OF THE JAVAFX CHARTS LIBRARY IN VIEW OF REAL-TIME VISUALISATION OF SCIENTIFIC DATA

Ralph J. Steinhagen*, Harald Bräuning, Alexander Krimm, Timo Milosic
GSI Helmholtzzentrum, Darmstadt, Germany

Abstract

The accurate and performant graphical representation of accelerator- or beam-based parameters is crucial for commissioning and operation in any modern accelerator. Based on earlier GSI and CERN designs a new JavaFX based ChartFx scientific charting library has been developed [1] that preserves the feature-rich and extensible functionality of established earlier Swing-based libraries [2,3] while addressing the performance bottlenecks and API issues. The library has been optimized for real-time data visualization at 25 Hz for data sets with a few 10 thousand up to 5 million data points common in digital signal processing applications.

Relying on modular open interface abstractions, the library allows the exchange of the underlying technology if necessary, while easing its use by casual developers as well as allowing more-inclined developers to modify, add or extend missing functionalities. This contribution provides a performance and functionality comparison with other existing Java-based charting libraries.

INTRODUCTION

Charts are one of the most visible but at the same time often underappreciated accelerator control system components even though these are crucial for ease of access to the data and to improve the intuitive understanding of complex or large quantities of data. In turn the understanding gathered through charts is used to efficiently control, troubleshoot or improve the accelerator performance.

Java has been adopted for FAIR’s accelerator settings supply and other control subsystems [4-6]. The possibility of reusing server code on the client sides initially made the use of Java’s Swing UI framework a favourable choice. Albeit lacking native support for charting, a number of complementary third party libraries have been developed, some with mixed quality and a very limited scope.

The JDataViewer is one of the more notable free, open-source and powerful exceptions to the rule and being extensively used at CERN and GSI [2,3]. Unfortunately, its API is bound to the – by today’s standard aging – Swing-based API that is to be phased-out from the JDK latest by 2026 [7].

While its successor JavaFX provides a basic chart implementation, it presently still lacks significant functionalities and performance for real-world scientific control room applications, typically requiring real-time update of large data sets with 10k or more data points, visualisation of error-bars and surfaces, data parameter measurement indicators (here: ‘20%-80% rise-time’ between ‘Marker#0’ and ‘Marker#1’). More examples are available at [1].

The user-side API was kept compatible to JavaFX’s Chart API in order to preserve a low entry-level learning curve, easy interaction with other JavaFX UI code, and to keep the required boilerplate code to a minimum. The internal interfaces follow a more modular separation-of-concerns paradigm and split the library into functional sub-modules: DataSet containing the measurements data itself as well as relevant meta information; a math sub-library that allows performing common signal processing using the DataSet interface; Chart with a Canvas pane as back-end for drawing the data; Axis responsible for the scaling, data sanitization,
Chart Functionalities and Features

The library offers a wide variety of plot types common in the scientific signal processing field, a flexible plugin system as well as online parameter measurements commonly found in lab instrumentation. Some of its features include:

- **DataSet**: basic XY-type datasets, extendable by `DataSetError` to account for measurement uncertainties, `DataSetMetaData`, `EditableDataSet`, `Histogram`, or `DataSet3D` interfaces;
- **math sub-library**: FFTs, Wavelet and other spectral and linear algebra routines, numerically robust integration and differentiation, IIR- & FIR-type filtering, linear regression and non-linear $\chi^2$-type function fitting;
- **Chart**: providing euclidean, polar, or 2D projections of 3D data sets, and a configurable legend;
- **Axis**: one or multiple axes that are linear, logarithmic, time-series, inverted, dynamic auto-(grow)-ranging, automatic range-based SI and time unit conversion;
- **Renderer**: scatter-plot, poly-line, area-plot, error-bar and error-surfaces, vertical bar-plots, Bezier-curve, stair-case, 1D/2D histograms, mountain-range display, true contour plots, heatmaps, fading `DataSet` history, labelled chart range and indicator marker, hexagon-map, meta data (i.e. for indicating common measurement errors, warnings or infos);
- **ChartPlugin**: data zoomer with history, zoom-to-origin, and option to limit this to X and/or Y coordinates, panner, data value and range indicators, cross-hair indicator, data point tool-tip, `DataSet` editing, table view, export to CSV and system clipboard, online axis editing, data set parameter measurement such as rise-time, min, max, rms, etc.

In order to provide some of the scenegraph-level functionality while using a Canvas as graphics backend, the functionality of each module was extended to be readily customized through direct API methods as well as through external CSS-type style sheets.

Chart Performance

Besides the extended functionality outlined above, the ChartFx optimisation goal also included achieving real-time update rates of up to 25 Hz for data sets with a few 10k up to 5 million data points. In order to optimise and compare the performance with other charting libraries, especially those with only reduced functionality, a reduced simple oscilloscope-style test case has been chosen that displays two curves with independent auto-ranging y-axes, common sliding time-series axis, and without further `ChartPlugins` as visualised in Figure 3(a).

The direct performance comparison between the ChartFx and JavaFX charting library for update rates at 25 Hz and

---

1 While this was not known to the authors at the time of the initial ChartFx implementation, this shortcoming of JavaFX was also noted in [10,11]

2 such as over- or under-ranging, device or configuration errors etc.
Performance test scenario with two independent graphs, independent auto-ranging y-axes, and common scrolling time-series axis.

Test results at 25 Hz update rate.

Test results at 2 Hz update rate.

Figure 3: Comparison between the JavaFX and ChartFx performance for a test cases with update rates at 25 Hz and 2 Hz. Note the logarithmic horizontal axis. Test system: Intel(R) Core(TM) i7 CPU 860 2.80GHz and GeForce GTX 670 GPU.

2 Hz is shown in Figures 3(b) and 3(c). While the ChartFx implementation already achieved a better functionality and a two orders of magnitude improved performance for very large datasets, the basic test scenario has also been checked against popular existing Java-Swing and non-Java based UI charting frameworks. Figure 4 provides a summary of the evaluated chart libraries for update rates at 25 Hz and 1k samples.

Figure 4: Chart performance comparison for popular JavaFX, Java-Swing, C++/Qt and WebAssembly-based implementations [1,2,8,9,11–16]. The last ‘Qt Charts’ entries show results for 100k data points being updated at 25 Hz.

The HanSolo-Chart [11] provides only static linear grids, axes and no data sanitization. The starred ChartFx result is for a reduced grid functionality without anti-aliasing and CSS-styling. The two JDataViewer reference results show the default (Polyline) and improved data-reducing ErrorDataSetRenderer. It is visible that the Swing use of the Java2D interface is at least a factor two better than the functionally equivalent Java use of the OpenGL interface. The native C++-based use of OpenGL is about an order of magnitude more performant than its Java counterpart.

CONCLUSION

While starting out to improve the JDK’s JavaFX Chart functionality and performance through initially extending, then gradually replacing bottle-necks, and eventually redesigning and replacing the original implementations, the resulting ChartFx library provides a substantially larger functionality and achieved an about two orders of magnitude performance improvement. Nevertheless, improved functionality aside, a direct performance comparison even for the best-case JavaFX scenario (static axes) with other non-JavaFX libraries demonstrated the raw JavaFX graphics performance – despite the redesign – being still behind the existing Java Swing-based JDataViewer and most noticeable the Qt Charts implementations. The library will be further maintained at GitHub and further used for existing and future JavaFX-based control room UIs at GSI. The gained experience and interfaces will provide a starting point for a planned C++-based counter-part implementation using Qt or another suitable low-level charting library.

ACKNOWLEDGEMENTS

We express our thanks and gratitude to Greg Krug and Vito Baggioni at CERN for their valuable insights, discussions and feedback on this topic.
REFERENCES


