ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB028

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 easing and improving a quick 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, and user-interaction such as zooming and dataset editing. While third-party extensions address some of the shortcomings [8,9], the rendering performance and other missing features issues remain. Many of the needed enhancements require workarounds around the original JavaFX API, were impeded by method override restrictions (ie. '`final`' keyword), lack of abstract interfaces, or the low-level technology choice of rendering inefficient '`Scene`' graphs of large and complex '`Node`' elements rather than, for example, using native '`double`'-based data point arrays.

ARCHITECTURE

Based on earlier designs and analysis of missing functionalities, performance bottlenecks, and long-term maintenance risks for the necessary workarounds, we decided that it was worth to re-engineer a new scientific charting library that preserves the functionality of established libraries, while being conforming to JavaFX standards, and while addressing performance bottlenecks and API issues of the JavaFX implementation. Visual examples illustrating some of its functionalities are given in Figures 1 and 2.

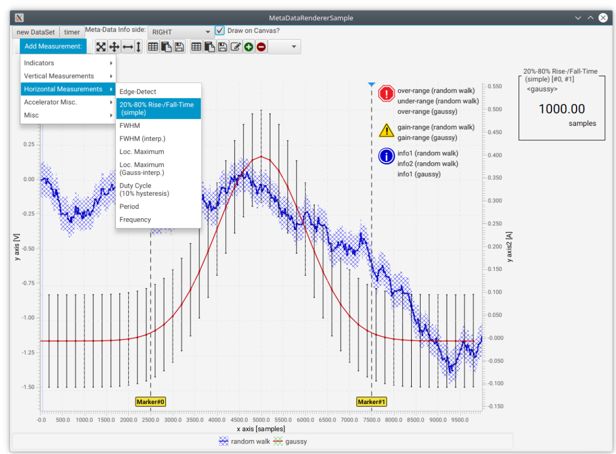


Figure 1: ChartFx example showing error-bar and error-surface representations, display of mock meta-data, `ChartPlugin` interactors and data parameter measurement indicators (here: '20%-80% rise-time' between 'Marker#0' and 'Marker#1'. More examples are available at [1].

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 sanitiza-

* R.Steinhagen@GSI.de

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB028

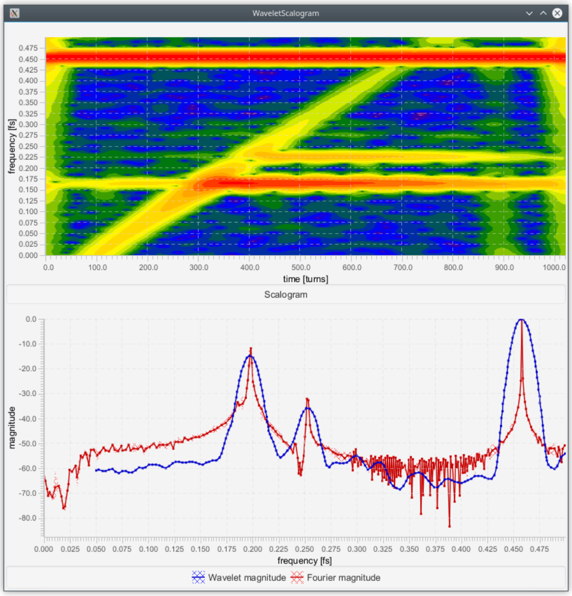


Figure 2: ChartFx example showing a heatmap representation of 2D wavelet transform betratron oscillation (N.B. chirp-type excitation), corresponding FFT and projected wavelet magnitude spectrum that have been all computed using the built-in math sub-library and `DataSet`.

tion, and transformation between real-world data and screen pixel coordinates; `Renderer` responsible for the specific data point drawing process; and `ChartPlugins` to provide methods for interacting with the chart or data.

The modules adhere to Java 1.8 compliant abstract observable interfaces, rather than using specific class implementations as used in JavaFX's `Chart` class. A common abstract and one or more example implementations thereof derived are provided for each interface. This abstraction establishes a more stable API and at the same time allows to reimplement, override, extend or modify existing functionality at the level of detail and experience of the individual developer. This also simplifies long-term maintenance or exchange of technologies in the underlying code without breaking dependencies to other modules or existing user-level applications.

The use of the `Canvas` pane is the key to the performance and provides substantially better hardware graphics acceleration[1]. This facilitates also further performance improvements for very large datasets by efficiently reducing data points prior to rendering that are drawn on top of each other or in such a close proximity that the user cannot visually discern their difference (e.g. less than three pixels on a HD screen). The `ReducingLineRenderer` and `ErrorDataSetRenderer` provide example implementations. The first performing a straight-forward data reduction in x, while the latter also considers reductions in y and accounts for propagation of measurement uncertainties.

[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]

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 χ^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[2]);
- `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.

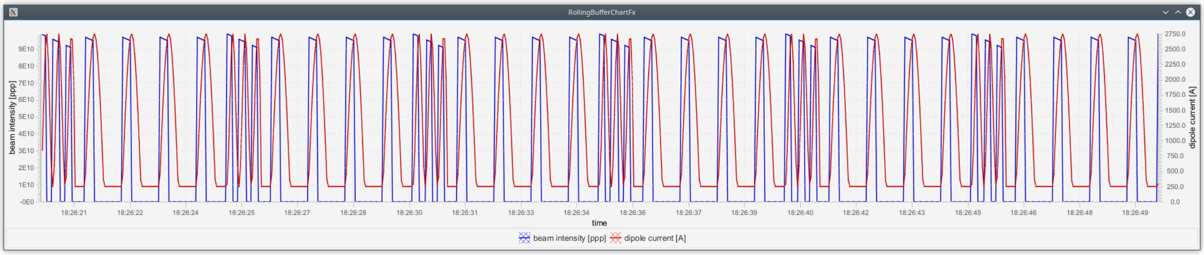
Chart Performance

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

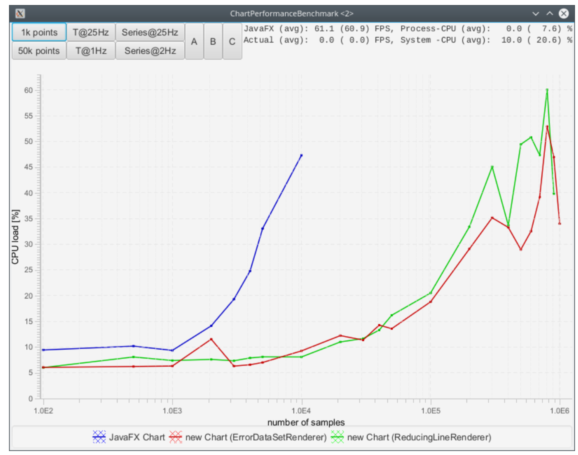
[2] such as over- or under-ranging, device or configuration errors etc.

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB028



(a) Performance test scenario with two independent graphs, independent auto-ranging y-axes, and common scrolling time-series axis.

(b) Test results at 25 Hz update rate.



(c) 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 by 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.

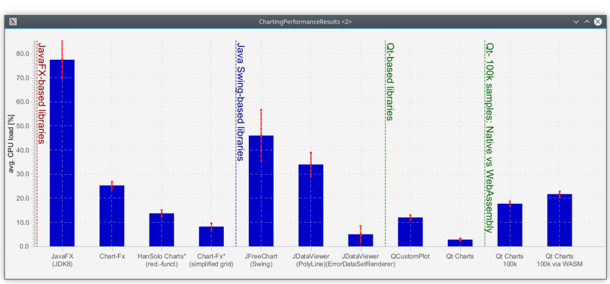


Figure 4: Chart performance comparison for popular JavaFX, Java-Swing, C++/Qt and WebAssembly-based implementations [1, 2, 8, 9, 11–16]. The last 'Qt Charts' entries show results for 100k data points being updated at 25 Hz.

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 JavaFX 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 re-designing 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 Baggiolini at CERN for their valuable insights, discussions and feedback on this topic.

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-THPRB028

REFERENCES

[1] H. Bräuning and R. J. Steinhagen, "Chart-Fx Project at GitHub" GSI Helmholtzzentrum, Darmstadt, Germany. [Online] https://github.com/GSI-CS-CO/chart-fx

[2] G. Kruk and M. Peryt, "JDataViewer - Java-based Charting Library," in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'09)*, Kobe, Japan, Oct. 2009, pp. 856–858. [Online] https://cds.cern.ch/record/1215878

[3] Z. Makonnen, "JDataViewer 3D Extension: Design, Development and Usability Test," Master's thesis, Université de Genève, Geneva, Switzerland, Mar 2012. [Online] http://cui.unige.ch/isi/icle-wiki/_media/papers:thesis:master_thesis_makonnen.pdf

[4] S. Deghaye, M. Lamont, L. Mestre, M. Misiowiec, W. Sliwinski, and G. Kruk, "LHC Software Architecture (LSA) – Evolution toward LHC Beam Commissioning," in *Proc. 11th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'07)*, Oak Ridge, TN, USA, Oct. 2007, pp. 307–309.

[5] R. Müller, J. Fitzek, and D. Ondreka, "Evaluating the LHC Software Architecture for Data Supply and Setting Management within the FAIR Control System," in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'09)*, Kobe, Japan, Oct. 2009, paper THP012, pp. 697–699.

[6] D. Ondreka, J. Fitzek, H. Liebermann, and R. Müller, "Settings Generation for FAIR," in *Proc. of International Particle Accelerator Conference (IPAC'12)*, New Orleans, LA, USA, May 2012, paper THPPR001, pp. 3963–3965.

[7] Oracle Inc. *Java Client Roadmap Update*. [Online] https://www.oracle.com/technetwork/java/javase/javaclientroadmapupdate2018mar-4414431.pdf

[8] G. Kruk, O. D. S. Alves, and L. Molinari, "JavaFX Charts: Implementation of Missing Features," in *Proc. 16th International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct 2017, pp. 866–868, doi:10.18429/JACoW-ICALEPCS2017-TUPHA186

[9] G. Kruk, "ExtJFX," CERN. [Online] https://github.com/extjfx/extjfx

[10] D. Lemmermann, "JavaFX Tip 20: A lot to show? Use Canvas!" [Online] https://dlsc.com/2015/06/16/javafx-tip-20-a-lot-to-show-use-canvas/

[11] G. Grundwald, "HanSolo Charts," [Online] https://github.com/HanSolo/charts

[12] Object Refinery Limited, "JFreeChart," 2005-2017. [Online] http://www.jfree.org/jfreechart/

[13] E. Eichhammer, "QCustomPlot," 2018. [Online] https://www.qcustomplot.com/

[14] Qt Company Ltd., "Qt Charts," 2019. [Online] https://doc.qt.io/qt-5/qtcharts-index.html

[15] WebAssembly Community Group, "Webassembly specification," Tech. Rep., May 2019. [Online] https://webassembly.org/

[16] Qt Company Ltd., *Qt documentation - qt for WebAssembly*, 2019. [Online] https://doc.qt.io/qt-5/wasm.html