2009 ICALEPCS

International Conference on Accelerator and Large Experimental Physics Control Systems
October 12 – 16, 2009, Kobe International Conference Center, Kobe , Japan

CERN

JDATAVIEWER – JAVA-BASED CHARTING LIBRARY

G. Kruk, M.Peryt, CERN, Geneva, Switzerland

Abstract

The JDataViewer is a Java-based charting library developed at CERN, with powerful, extensible and easy to use function editing capabilities. Function edition is heavily used in Control System applications, but poorly supported in products available on the market. The JDataViewer enables adding, removing and modifying function points graphically (using a mouse) or by editing a table of values. Custom edition strategies are supported: developer can specify an algorithm that reacts to the modification of a given point in the function by automatically adapting all other points. The library provides all typical 2D plotting types (scatter, polyline, area, bar, HiLo, contour), as well as data point annotations and data indicators. It also supports common interactors to zoom and move the visible view, or to select and highlight function segments. A clear API is provided to configure and customize all chart elements (colors, fonts, data ranges ...) programmatically, and to integrate non-standard rendering types, interactors or chart decorations (custom drawings). Last but not least, the library offers class-leading performance.

1. Chart Component

The central component of the library is the `Chart` class. The chart is a graphical component that initializes and coordinates the drawing process of all other chart elements i.e. plots and decorations, scales, grids and the legend.

The chart can contain a single X scale and one or more Y scales. Steps and sub-steps of every scale are computed by the associated `StepsDefinition`. The default implementation computes steps using 1, 2 and 5 factors, but there are also implementations for logarithmic, time and category (custom) scales [Fig. 1, 2, 3, 4].

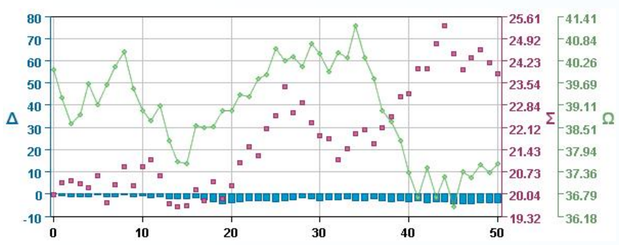

Figure 1: Chart component with three rendering types associated with three different Y scales

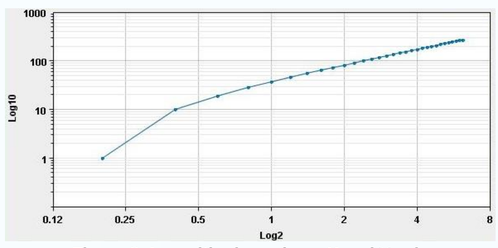

Figure 2: Logarithmic scale on X and Y axis

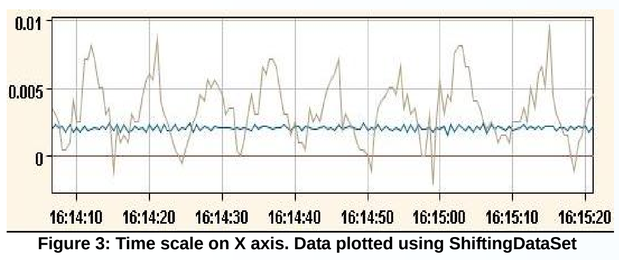

Figure 3: Time scale on X axis. Data plotted using ShiftingDataSet

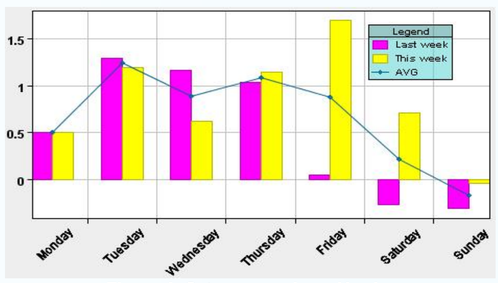

Figure 4: Category scale on X axis

2. Data Model and Renderers

In order to display a plot on the chart one has to first create an instance of the `DataSet` interface that represents a single series of data points (X, Y). There are several implementations of this interface provided by the library but the two most frequently used are `DefaultDataSet` for "snapshot" data [Fig. 1, 2, 4] and `ShiftingDataSet` for continuous signals [Fig. 3]. The package contains also a `DataSet3D` which can be displayed as contour plot [Fig. 5, 6].

Once data sets are created they can be connected to appropriate chart renderers which are responsible for the drawing process of data points. There are several types of renderers implemented in the library and each draws data in a specific way e.g. the `PolylineChartRenderer` draws data sets as poly lines, the `BarChartRenderer` paints data points as bars, etc. Many renderers can be added to the chart and each of them paints plots from all associated data sets [Fig. 1, 4].

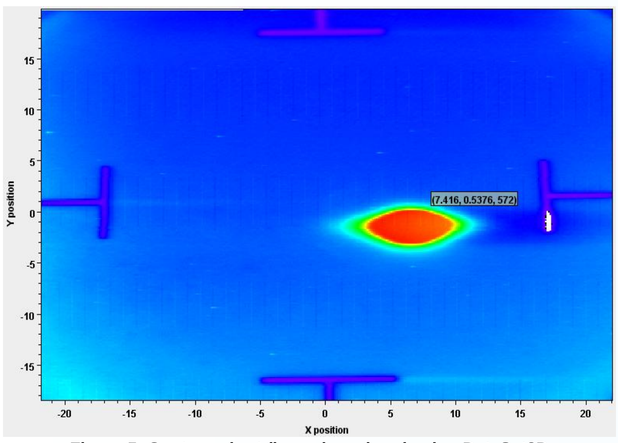

Figure 5: Contour chart (image) rendered using DataSet3D

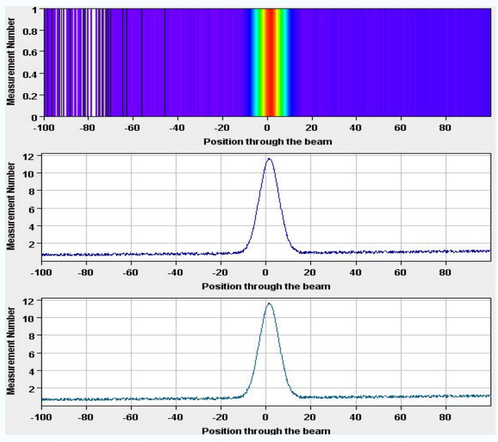

Figure 6: Three charts with synchronized X axis

3. Annotations and Decorations

The package supports two other types of components that can be added to the chart: annotations and decorations [Fig. 7, 8].

Annotations can be used to display information about data points e.g. Y value of each point or a custom label associated with it. Decorations are dedicated for custom text or drawings which are not related to specific data sets. Two custom decorations are included: `ChartAnnotation` that can be used to display a custom label on a chart and `DataIndicator` that is dedicated to marking a single value or a range of values. This functionality is typically useful when one wants to highlight limits of a signal (to see whether they are exceeded or not) or values which have some special meaning .for the displayed data.

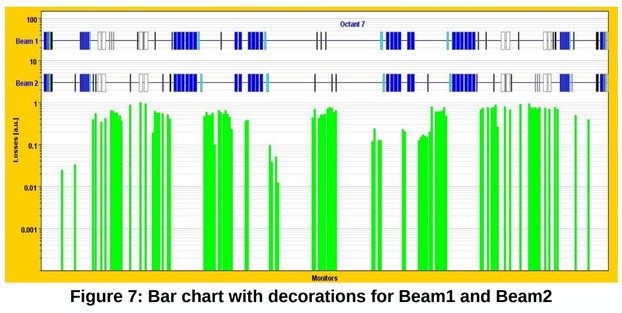

Figure 7: Bar chart with decorations for Beam1 and Beam2

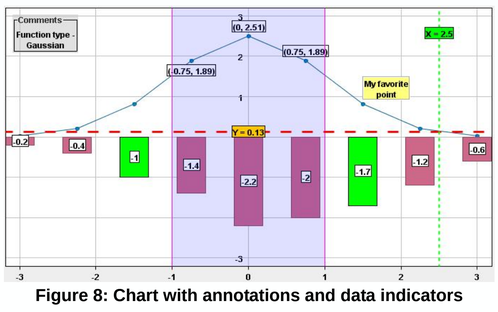

Figure 8: Chart with annotations and data indicators

4. Interactors and Functions Edition

Chart interactors are non-graphical components that can be used to interact with the chart. Every instance of an interactor registered in the chart is notified about all mouse and keyboard events received by the chart, so that it can perform appropriate actions. The library provides the most commonly used interactors like `ZoomInteractor` that zooms in and out selected regions of the chart or a `DataPickerInteractor` that displays as a tooltip the coordinates.

One of the key features offered by the package is a graphical and a tabular edition of functions. This functionality has been implemented via a set of edition interactors that are well integrated with the `Chart` e.g. `AddPointsInteractor`, `RemovePointInteractor`, `ChangePointInteractor`, or `AlignPointsInteractor`. Every interactor can define its control components – typically buttons or a drop down list - which are used to activate them or to modify their properties. All interactors fire events every time an interaction is performed, providing additional information when applicable e.g. nature of the interaction or coordinates of modified points. Therefore one can subscribe to such events and execute custom actions.

The chart provides also a method to create a toolbar containing all control components defined by the associated interactors and *undo*/*redo* operations that can be performed on the edited function [Fig. 9, 10].

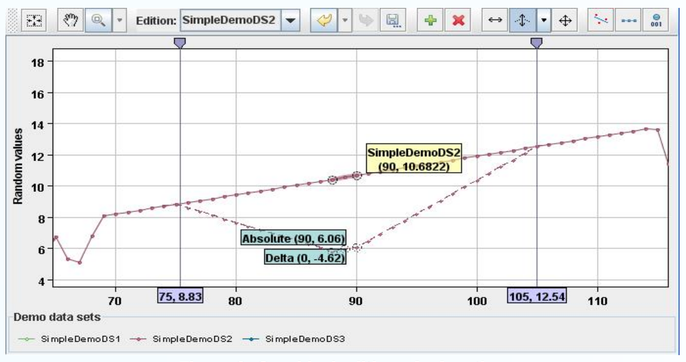

Figure 9: Graphical edition of a function

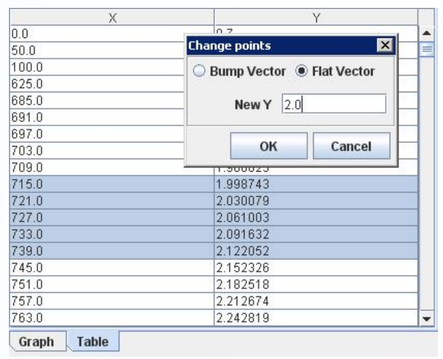

Figure 10: Tabular edition of a function

5. DataViewer Component

The `DataViewer` is a graphical component (panel) that simplifies layout and display of multiple charts. It may contain a number of views, where each consecutive view may include one or more charts. Only one view can be visible at a time but one has a possibility to browse through all available views and select the one that should be displayed. It is possible to dynamically modify layout, maximize or minimize selected charts and change a current rendering type of selected plots e.g. from a polyline to bars or to a table [Fig. 11].

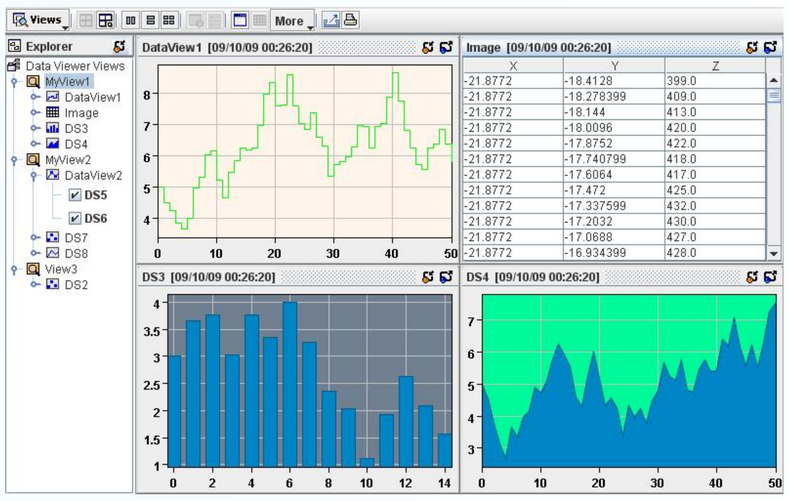

Figure 11: DataViewer component

6. Configuration using CSS

In typical applications, all chart attributes are configured directly in Java code using an API provided by the library. However in some cases it might be interesting to describe chart properties and properties of all its components using an external configuration file. Such functionality might be desired when the same look and feel of the chart should be applied in several applications (to avoid code repetition) or by frameworks that automatically generate chart components.

This possibility has been implemented in *JDataViewer* using the *Cascade Style Sheets (CSS)* format. Most of the chart properties and properties of its components (plots, grids, scales...) can be configured using appropriate tags in a CSS file.

```
chart {
  renderingType: 'POLYLINE';
  interactors: 'DATA_PICKER, ZOOM';
  legendTitle: "My Legend";
}
scale[axis='X'] {
  title: 'X coordinates';
  titleAlignment: LEFT;
  foregroundColor: BLUE;
}
dataset[index='1'] {
  renderingType: 'AREA';
  strokeColor: #FF0000;
}
```

Example of a CSS file defining properties of the chart and its components.

7. Extensibility

The library offers a reach set of generic and configurable components, both graphical and non-graphical. Although this is satisfactory for majority of controls applications, in some cases more specialized components or behaviour might be required.

Thanks to the clear API, all existing classes can be extended and tailored to specific needs. Also custom renderers, interactors and decorations can be implemented and easily integrated with the chart if necessary.

Conclusions

Since the initial implementation in 2005, the library has been very well perceived by developers. The number of applications depending on it, has been rapidly growing over the past few years. Diversity of uses has also increased over time, changing from trivial signals displayed offline, to applications used to edit function settings, and GUIs displaying data coming with a relatively high frequency (10-20Hz).