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.

MOTIVATION

An integral part of most of control systems GUI applications are charts and plots. They are used to display current measurements of various signals, browse logged data or display results of diverse simulations. Another important feature often required by controls applications is a possibility to graphically modify displayed plots which typically represent functions to be played by the hardware (device settings).

Although there are many charting libraries available on the market, both open source and commercial, all of them provide either none or very limited support for functions edition. Extending or trimming functionality of these libraries is not an obvious choice as one has no influence on internal APIs and in case of commercial products there is even no access to the source code. Also considering the number of licences required for tens of controls applications and developers, and the level of support required by these developers, it is also not economically sound.

Taking into account all these aspects, it was decided to develop a new charting library with support for typical plotting types used in controls applications and with powerful and extensible editing facilities that would be used by all operational controls applications at CERN.

LIBRARY OVERVIEW

Chart

The central component of the library is the `Chart` class. The `Chart` is a graphical component (`JLayeredPane`) that initializes and coordinates the drawing process of all other chart elements i.e. plots and decorations (discussed later in this paper), scales, grids and a legend.

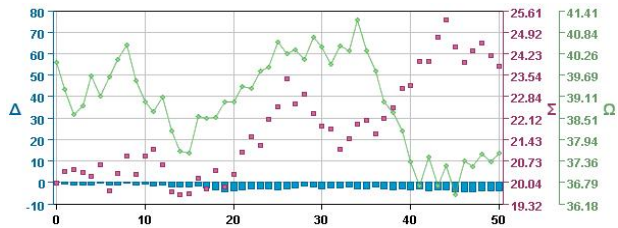

Figure 1: Chart component with three rendering types associated with three different Y scales.

All the drawings are made on a centrally placed chart `Area` (`JComponent`). In the centre of the area there is a plotting rectangle surrounded by scales.

The chart can contain a single X scale and one or more Y scales. The scale has many configurable attributes like title text, title position and rotation, foreground color, tics size, layout etc. Values that appear on the scale (steps and sub steps) are computed by the `StepsDefinition` associated with every scale. By default the `Scale` instances use `DefaultStepsDefinition` that computes steps using 1, 2 and 5 factors. The library includes also a `TimeStepsDefinition`, which computes appropriate time units, a `LogarithmicStepsDefinition` with configurable logarithm base and a `CategoryStepsDefinition` which is useful when one wants to display on the scale custom labels rather than numeric values.

Grids (instances of the `Grid` class) are associated with scales. All lines drawn by every grid follow step and substep values of corresponding scale. Similarly, like in case of the `Scale`, one can configure all visual attributes of the `Grid` like visibility, stroke or color.

The `Legend` component can be drawn in two different forms – as a floating panel that can be dragged on the plotting area (by default with a semi-translucent background) or as a panel placed on the side of the chart. Again, its visibility, position and paint attributes are configurable.

Data Model

In order to display a plot in the chart one has to first create an instance of the `DataSet` interface that represents a single series of data points (X, Y) of type double. There are several implementations of this interface provided by the library but the two most frequently used are `DefaultDataSet` and `ShiftingDataSet`. The generic `DefaultDataSet` is well suited for a typi-

cal "snapshot" data, allowing adding, removing and modifying all points. The `ShiftingDataSet` is dedicated for signals whose values change over time. It keeps data points in an internal circular buffer of specified size, so that when new points are added (with higher timestamp), the oldest are removed automatically.

The package contains also a `DataSet3D` interface and a default implementation that is meant for 3-dimensional charts. At the moment such 3D data sets can be displayed using contour plot (color-coded surface) – see Fig 3.

All data sets which should be displayed using the same plot type (e.g. as poly lines) must then be put into a `DataSource` which is simply a container for a group of `DataSets`. Such `DataSource` can then be connected to an appropriate chart renderer, which draws all series of data points.

Renderers

Chart renderers (classes that extend the `ChartRenderer` class) 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 polylines, the `BarChartRenderer` paints data points as bars, etc. Every renderer paints all the data sets that are contained in the associated `DataSource`.

Once the data source is linked with a selected type of renderer, the renderer should be registered in the chart. In general, many renderers can be added to a single chart therefore it is possible to mix plot types e.g. one can display certain data sets as bars and other data sets as points or as polylines – see Fig 1.

Interactors

Chart interactors (classes that extend the `ChartInteractor` class) are non-graphical components that can be used to interact with the chart. Every instance of an interactor that is 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: a `ZoomInteractor` that zooms in and out selected regions of the chart, a `DataPickerInteractor` that displays as a tooltip the coordinates, data set name and other custom information for the point the mouse is hovering over, or a `CursorInteractor` that can be used to graphically select (with a vertical or horizontal line) a single value or a data range - see Fig 4. There are also a number of edition interactors that will be discussed in the next part of this paper.

Decorations and Annotations

The library also supports two other types of components that can be added to the chart: data point annotations and chart decorations – see Fig 2.

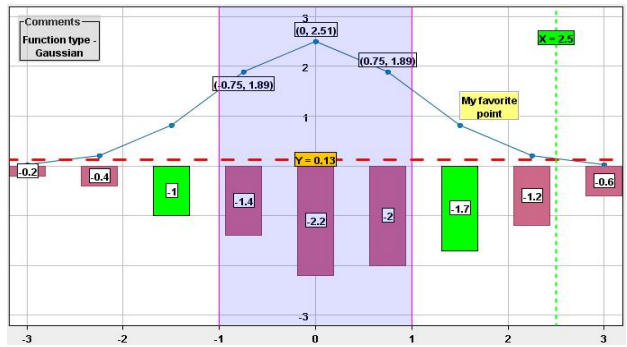

Figure 2: Chart with data indicators and annotations.

Annotations (implementations of the `DataAnnotation` interface) can be typically used to display information about all or predefined data points e.g. Y value of this point or a custom label associated with it. Annotations can be configured in terms of the look and feel, the text that is displayed and placement with respect to the data point.

Decorations (extensions of the `ChartDecoration` class) on the other hand are dedicated for custom text or drawings which are not related to specific data sets or points. The library includes two custom decorations. The first one `ChartAnnotation`, can be used to display a text label on a chart in a specified place e.g. to show information about currently displayed plots. The second one `DataIndicator`, is dedicated to marking a single value, a range of values or a window of values (data range on X and Y axis). This functionality is typically useful when one wants to highlight limits of a displayed signal (to see whether they are exceeded or not) or values which have some special meaning for the displayed data.

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, legend ..) can be configured using appropriate tags in a CSS file.

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

Example of CSS file defining chart properties.

DataViewer

The library includes also a graphical component called `DataViewer`, which simplifies layout and display of charts in case there are several of them to be shown in a single application.

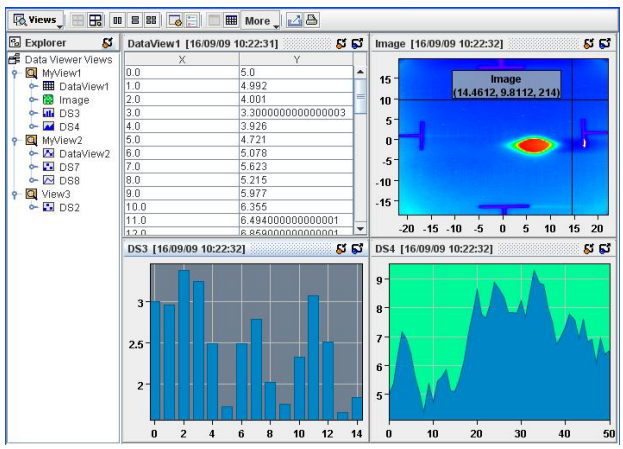

Figure 3: DataViewer component with three views.

The `DataViewer` is essentially a panel that may contain a number of views (`DVView`) where each consecutive view may contain one or more charts. Only one `DVView` can be visible at a time but a user has a possibility to browse through available views and select the one that should be displayed at a given moment. Additionally the user can 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.

FUNCTIONS EDITION

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. Typically, only one edition interactor is active at a time so that one can add, remove, select and change points with few mouse clicks and movements.

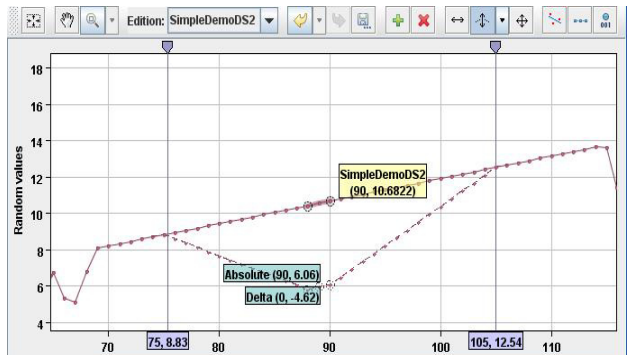

Figure 4: Function edition by dragging selected points.

All interactors that are included in the library 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.

The function edition is heavily used at CERN in applications dedicated for settings management i.e. generation and trimming. Examples of such function settings are magnet strengths, power converter currents or radio frequency voltages that are typically functions of time.

EXTENSIBILITY

The library offers a rich 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 or decorations can be implemented and easily integrated with the chart if necessary.

CONCLUSIONS

Since 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).

Today, *JDataViewer* is used by almost all operational applications (requiring charting) in the Controls group at CERN and it is now being adopted by the LHC experiments and other laboratories like GSI.