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.

2. Data Model and Renderers
In order to display a plot on the chart one has to first create an instance of the DataModel interface which 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. 2, 3, 4] and ShiftingDataSet for continuous signals [Fig. 2]. The package contains also a DataSet3D which can be displayed as contour plot [Fig. 5, 6].

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.

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 is identified about all mouse and keyboard events occurred 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 DataPointInteractor that displays as a tooltip the coordinates.

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, browse or minimize or maximize selected charts and change a current rendering type of selected plots e.g. from a polyline to bars or a table [Fig. 9, 10].

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 (e.g. as a code repetition) or by frameworks that automatically generate chart components.

7. 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 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 (10-50Hz).