Programm architecture

Project “TrainVis“

By: Markus Martin
Hansjörg Pripamer
Georg Prohaska

1 Introduction

The goal of this project was to create a tool that can visualize a train schedule using four different views: A normal textual table, two different types of bar chart diagrams and finally a technique that uses lines that represent a train (after EJ. Marey). For further information about these visualization-techniques read the designdocument.
Since it was predetermined that we had to use the prefuse-toolkit, the programming language for the project had to be java. Our team consisted of three people and therefore a revision control system to synchronize the programming process was necessary. We used Subversion for that matter. Also all of us used the same integrated development environment (IDE) namely NetBeans which does support the Subversion-system.

2 Toolkits

The two "big" frameworks we used are *Prefuse* and *TimeVis*. Then there is another small toolkit we employed. Its name is *JCalendar* and it provides functionalities like a date-chooser (menu component) or classes to manage temporal data (especially dates).

2.1 Prefuse

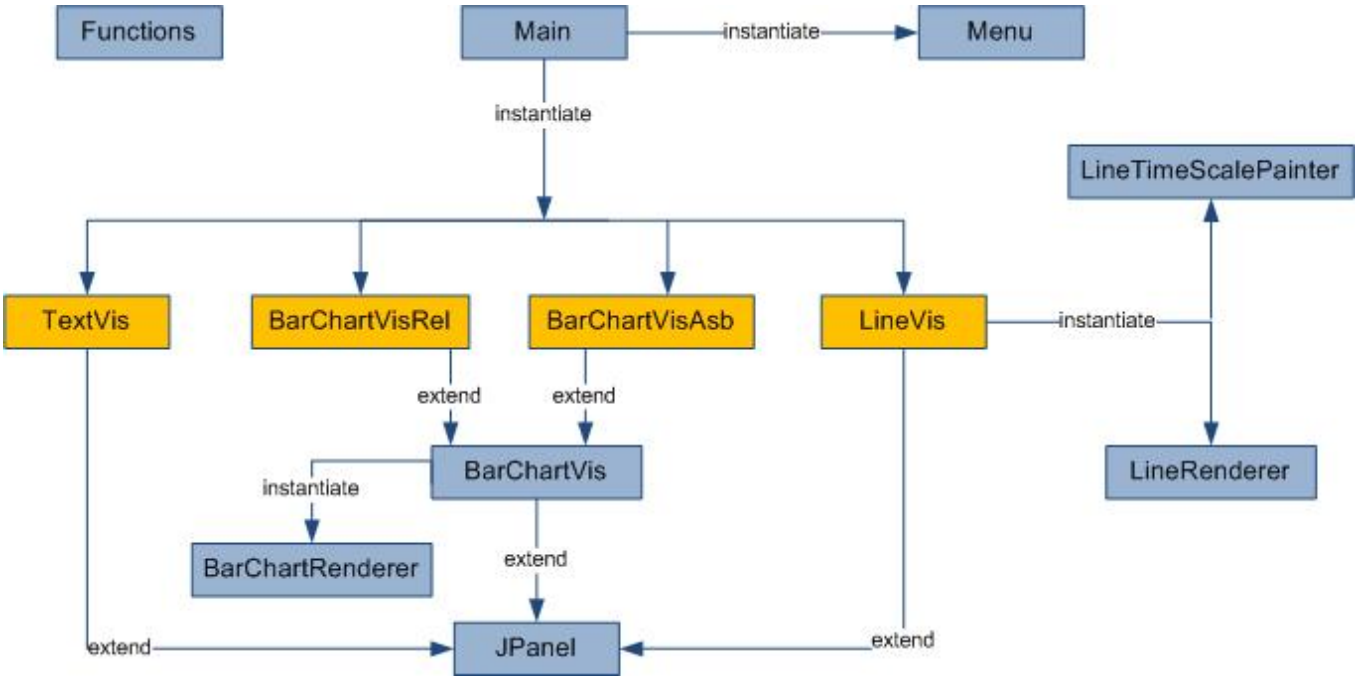
Prefuse is a "Java-based toolkit for building interactive information visualization applications" (http://prefuse.org). As mentioned before the usage of this toolkit was specified by our tutor and so the project is based on its functionality.
Whenever a data table is loaded a new internal data table is created by using the *makeData*-method in the *Function* class. In this new table every row specifies a train that goes from one station to another at ONE point in time. So if a train goes from A to B every day of the schedule there is a separate row for each day. This big table is also the biggest deficiency of the application since it cuts down the performance. There is surely room for improvement in this matter.
The internal table is used by the four main classes to create the different visualizations. In the LineVis-class for example each row is used to create a *VisualItem* which can be rendered as one line. We implemented two different *Renderers* to draw the bars of the bar chart diagrams and the lines of the Marey diagram: *BarChartRenderer* and *LineRenderer*.
We used the action-model of prefuse to manipulate the data tables. The *BorderFilter* class (an inner class of *LineVis*) for instance renders all visualitems invisible that cross a specific border so that the lines don't overlap the axis labels.

2.2 TimeVis

"TimeVis provides a framework that enables Java developers to build applications to visualize any kind of temporal data without much effort." The use of this framework was also specified by our tutor. We used it to assign the horizontal position of our visualitems on the panel. Each main-class (*TextVis, LineVis, BarChartVisRel, BarChartVisAbs*) has its own *TimeScale*-object which determines the timescale that is currently displayed. To synchronize the scales of all views we used a message-system that will be explained in the next section.

Classes like *TimeScalePainter* and *MouseTracker* facilitated the creation of a good-looking environment for our visualizations.

3 Class diagram and program structure



The four most important classes of the project are *TextVis, LineVis, BarChartVisRel*, and *BarChartVisAbs*. All of them derive from the Java-Swing class *JPanel*. Each of them encapsulates one form of visualization. You only have to pass consistent data tables to the constructor of these classes and everything will be displayed in the panel.
The *Main* class contains the *main* method of the application. At first the user interface and the default data tables are loaded. Therefore a *Menu* object is instantiated which represents the toolbar of the application. As soon as everything necessary (consistent data tables) is existent the four visualization classes (*TextVis, LineVis, BarChartVisRel*, and *BarChartVisAbs*) are instantiated and loaded into a *JTabbedPane*.
We used the *ChangeListener* class and the messaging system of swing to communicate between the user interface and the four panels. Every time an UI-component is used to a message is fired to the tabbed pane which passes it to each visualization object. For example when checkbox for the stations is selected a _SHOWSTATIONS_ message is fired that contains the new value of the checkbox. This system is also used within the visualization classes: Whenever the current timescale is changed by zooming or panning the currently active panel fires a _STARTDATE_ and _ENDDATE_ message to let the menu and the other panels know.
Now let's have a closer look at the visualization classes:

3.1 LineVis

This class accomplishes the Marey train-visualization. As described in section 2.1 the internal table created by the *makeData* method is used as a *VisualTable*. So each row of the table represents one *VisualItem*. The *TimeLayout* class (a class of the *TimeVis* framework) is used to calculate the horizontal positions of the items. For the drawing of the timescale in the background of the panel the *LineTimeScalePainter* class is used which is a slightly modified version of the *TimeScalePainter* class (also a part of *TimeVis*).

The rendering of each VisualItem takes place in the *LineRenderer* class which derives from the *AbstractShapeRenderer* class of prefuse. It may seem a bit awkward that this class needs to have the *Visualization* object passed in its constructor. This was necessary since in the Marey diagram a horizontal line is drawn between to stations if the stopover of the train takes a little more time. That's why the information contained by one *VisualItem* is not sufficient for its rendering.

3.2 BarChartVis

This class is an abstract class for both bar chart visualizations. Most of the work regarding the visualizations is done here. It provides functionality for zooming and panning. It also sets up the *Display* object and all necessary *ActionLists* and the *BarChartRenderer* that are needed to draw the visualization. Furthermore the process of train-selection is managed here by handling all mouse events. That's also where the tool tips for each item are created.

3.3 BarChartRel

This class displays a duration bar chart. The start of every bar lies in the origin of the x-axis. Here the *makeRelData* method is used to create the internal data table because in this type of visualization there is only one *VisualItem* for each train necessary. Furthermore this class catches all necessary messages from the tab pane and executes the according actions.

3.4 BarChartAbs

Here all the trains are displayed in a time line chart. Since most of the work is done in *BarChartVis* this class only need to process the incoming messages.

3.5 TextVis

Here the data is just displayed as a normal text table. As in the other panels the messages are handled here. A *JTextPane* is used to display the table.