Programm architecture

AsbruFlow

Interactive Information Visualization to
Support Protocol-based Care

Stephan Hoffmann
0325733

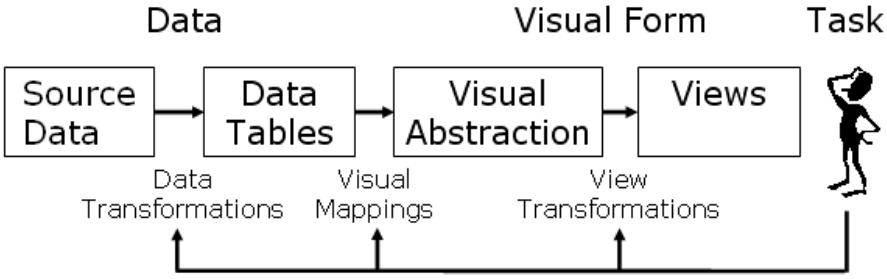
November 5, 2008

1 Introduction

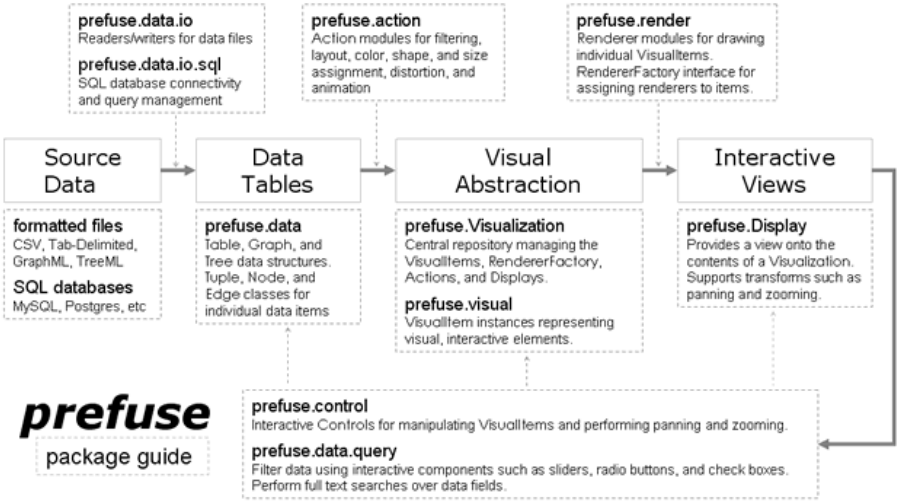
The aim of this project was to develop a interactive visualization toolkit to support protocol-based care. Based on the Software Prototype *CareVis*, this project is a approach to communicate the complex logic of *Asbruplans* to domain experts like physicians or nursing staff. Asbru can be used to express clinical protocols as skeletal plans that can be instantiated for every patient. Since a plan is modeled in XML, this representation is not well suited for physicians.
To illustrate an Asbruplan, it is neccessary to display different types of data: logical sequences, time-oriented data, flexible execution order, non-uniform element types and state characteristics of conditions.
AsbruFlow meets this requirements providing multiple views and using visualization methods well-known to domain experts. The tightly coupled views are based on the concepts of *clinical algorithm maps* and *LifeLines*.
This project was developed with JDK 1.6, Prefuse beta (release 2007.10.20) toolkit and the TimeVis API developed by Peter Weishapl.

2 Prefuse

Prefuse is a open source java-based toolkit designed to visualize connected information in form of graphs or trees. Additionally, prefuse offers some interaction techniques like Tooltips, dragging of visual elements, zooming and panning.
Prefuse is based on the *information visualization refernce model* shown below, which breaks up a information visualization process into different steps.



The next figure below illustrates how the different classes of the prefuse toolkit implement the reference model:

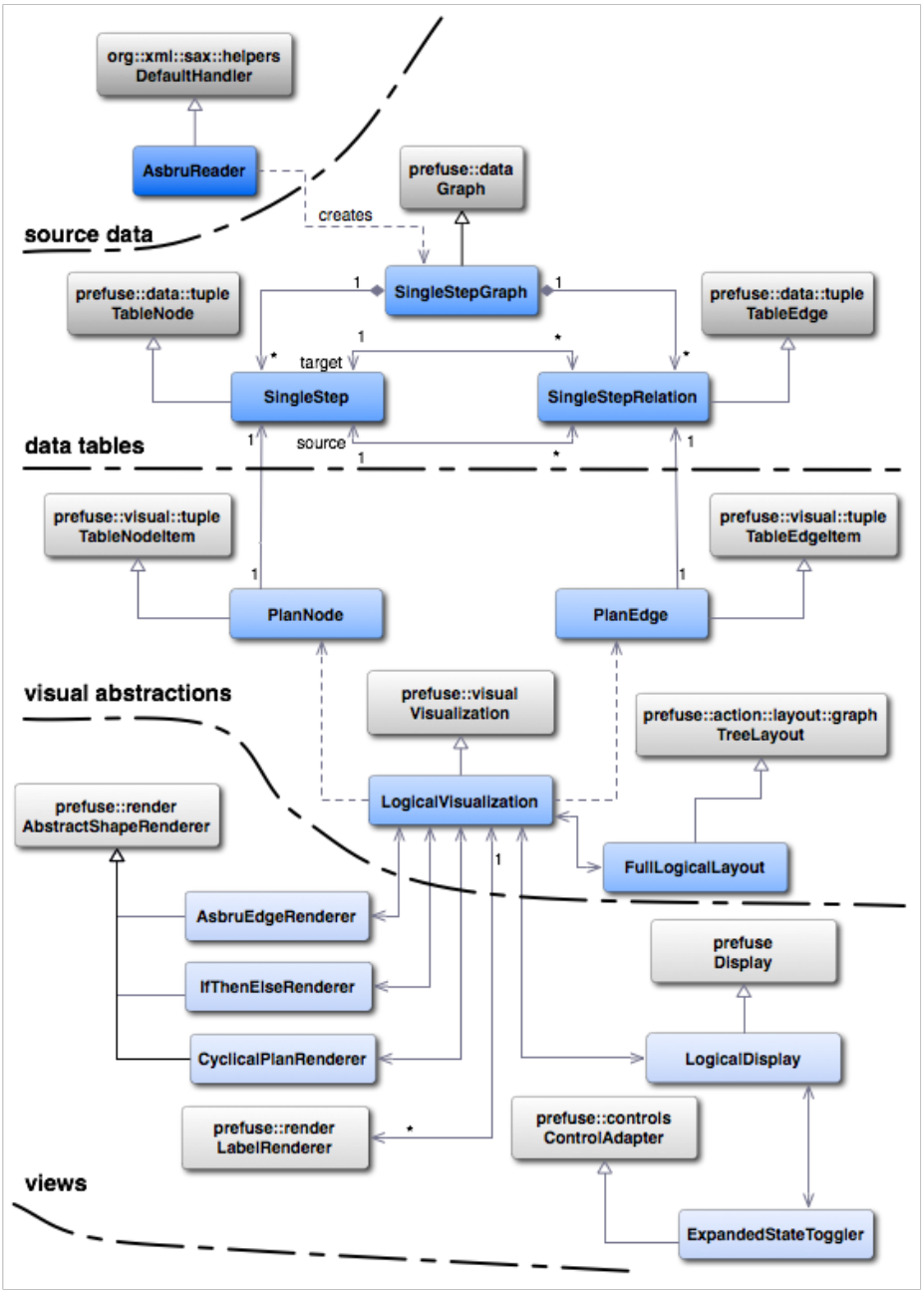


3 LogicalView

The LogicalView shows the logical sequences, hierarchical decomposition, non-uniform element types and conditions of an Asbru plan.
There are 2 different modes for this view, a Overview + Detail Mode and a FishEyeMode.

3.1 Overview + Detail Mode

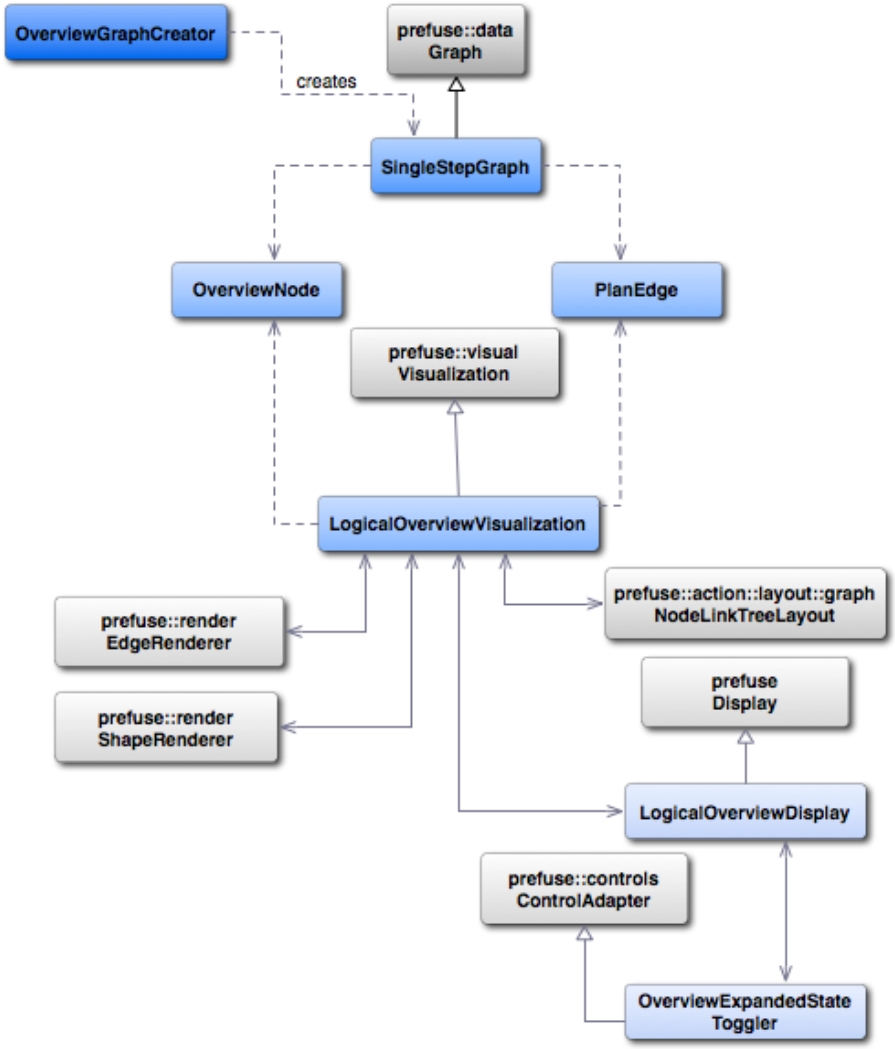
This mode uses a small window containing a downscaled, simplied tree overview where the current position within the plan is highlighted. The current expanded plan is displayed in the *LogicalPlanViewPanel* in an enclosing frame. If a plan within this enclosing frame gets expanded by the user (by clicking on the triangle), this plan gets displayed in the enclosing frame.
The planbody is visualized with prefuse. The mapping to the InfoVis Pipeline and which prefuse classes are utilised in this visualization is shown below.



The class *AsbruReader*, which is a XML Parser for Asbru files, creates a *SingleStepGraph* consisting of *SingleSteps* connected with *SingleStepRelations*. The visual representations of this graph is a *VisualGraph* consisting of the visual representations of *SingleSteps* and *SingleStepRelations*: *PlanNodes* and *PlanEdges*. These visual elements get visualized by the *LogicalVisualization*, applying a *FullLogicalLayout* on them, does some coloring and is responsible for the animations.

The actual rendering of the planbody happens in the *LogicalDisplay*, using a couple of different renderers, depending of the element type. The display hold some *ControllListeners* for user-interaction(zooming, panning, dragging) and also a special *ExpandedStateToggler* to recognize selection or expanding of a plan.

The simplied tree overview is also visualized with prefuse, the architecture is shown below.

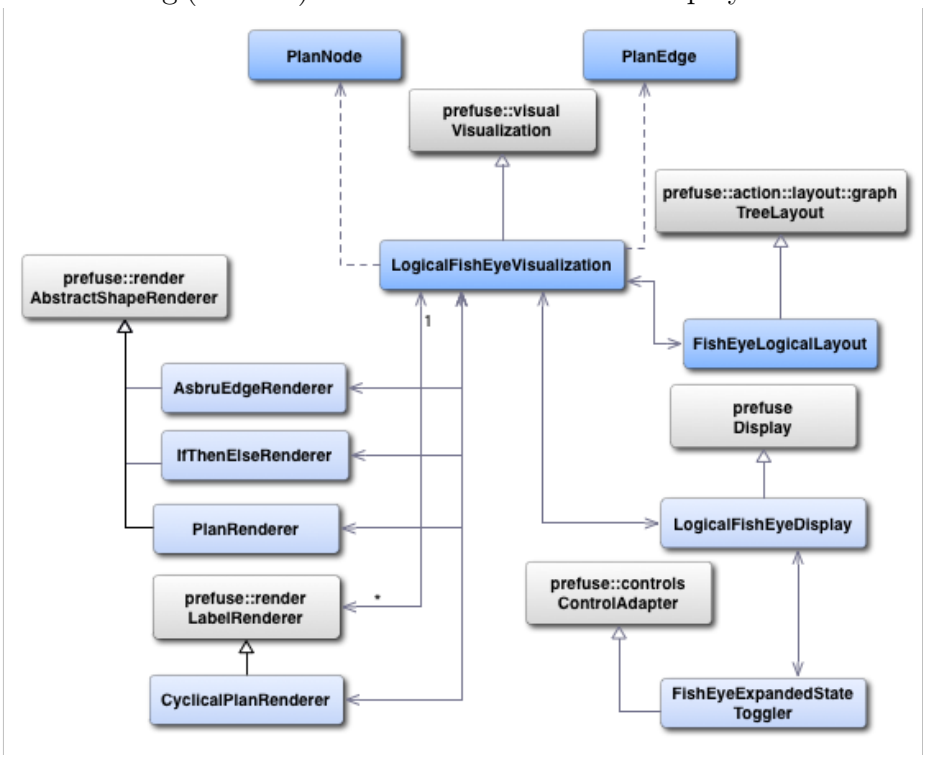


Since this visualization only displays the plantree without other elements such as ask, if-then-else assignments,.., a OverviewGraphCreator is responsible for the creation of the SingleStepGraph. This visualization is much simpler than the detailed visualization, so the standard prefuse renderers

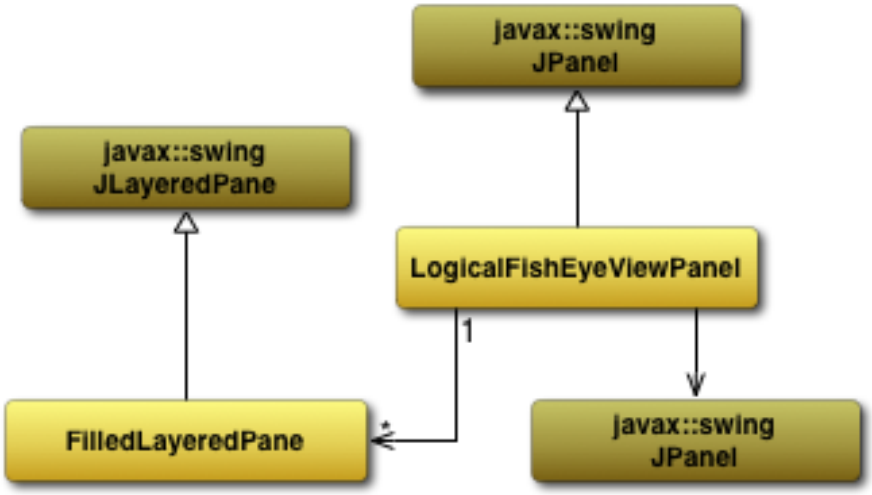
and layout-algorithm are adequate. The OverviewDisplay also has a ExpandedStateToggler that recognises clicks on the nodes.

3.2 Fisheye Mode

The current (sub)plan represents the focus which is displayed in full detail. The surrounding (context) elements are shrunk and displayed with less detail.



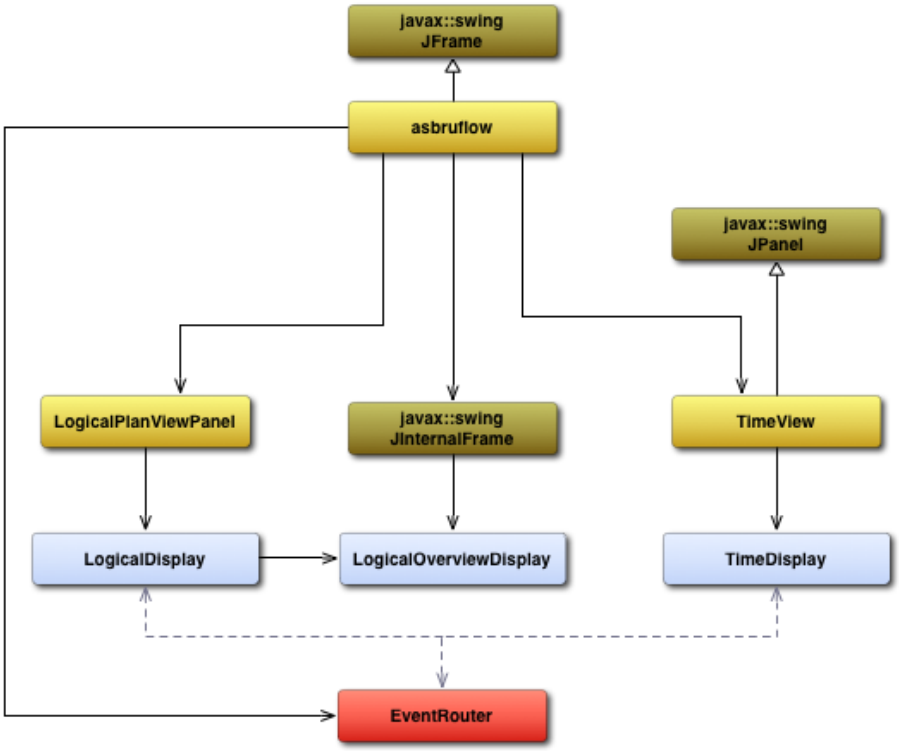
The architecture of this visualization looks nearly the same as the detail visualization in the O+D mode, the only difference is the usage of the *FishEyeLogicalLayout* for element positioning and the *PlanRenderer* instead the standard prefuse LabelRenderer. This is necessary because in this mode, a (sub)plan cannot only be expanded or selected, but also opened.
This mode needs 3 displays, each for every opened plan. The backmost plan-body is displayed in a *LogicalFishEyeDisplay* domiciled in a *FilledLayeredPane* in the backmost layer. If a plan gets opened in this display, a JPanel gets displayed on the next Layer, which contains the enclosing frame of the opened plan. This plan's body in turn is displayed in a *LogicalFishEyeDisplay* domiciled in a *FilledLayeredPane* in the backmost layer again. The next opened plan's enclosing frame is held in a JPanel again, the foremost *LogicalFishEyeDisplay* domiciled in a normal JPanel.



The size and position of the panels above the backmost layer is decided by the *FishEyeLogicalLayout* in the visualization one layer behind.

4 Overall architecture

The main class of the programm is derived from *JFrame*, this class creating the viewpanels for the LogicalView. Either LogicalFishEyeViewPanel or LogicalPlanViewPanel, in this case also creating a JInternalFrame for the overview display.
It also creates a JPanel for the temporal view, containing a prefuse display for the illustration of an Asbruplan. The architecture for this visualization is nearly the same as in the TimeVis Prototype and can be read there.



As can be seen, a *EventRouter* is instantiated in the main class too. The *EventRouter* is responsible to couple the logical and temporal view. E.g. a plan gets selected in one view, this event is sent to the *EventRouter*, who routes this event to the other view. Also double-clicking plans or dragging it to another view is supported.