188.308, 1.0h

prefuse Tutorial

beta release 2006.07.15

Wolfgang Aigner

aigner@ifs.tuwien.ac.at
http://ieg.ifs.tuwien.ac.at/~aigner/

27.11.2006

What is *prefuse*?

Extensible software framework to create interactive information visualization applications

Java

Open Source

Why *prefuse*?

OS independent (pure Java)

available demos and examples

good documentation

active community

many built-in components
- I/O methods
  - CSV, (tab-)delimited text, GraphML (XML), TreeML (XML)
- data structures
  - table, graph, tree
  - provides indexing, queries
- components for color, size, and shape encodings
- layout components
- distortion techniques
- animation (e.g., smooth transitions)
- dynamic queries / interactive filtering
- integrated text search
- physical force simulation engine
- SQL-like expression language for writing queries

Download and Build

Download
- Homepage: **http://prefuse.org**
- Beta release
- Download link in the upper left corner
- Unzip

Build
- Set "JAVA_HOME" environment variable
  - ANT build script (build.xml)
  - Ant = Java Build system
- highly recommended in general!
- Start Ant script via build.sh (Linux/Mac) or build.bat (Win)
- 1) Build classes and jars
  - Option "all"
- 2) Build API documentation
  - Option "api"

```
JAVA_HOME='/usr'; export JAVA_HOME
sh build.sh api
```

How to run demos

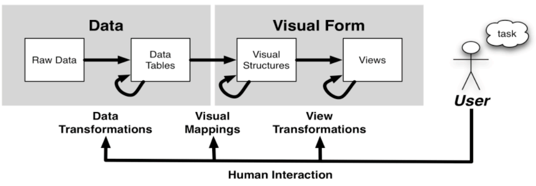
10 Available demos
- AggregateDemo - Vis of groups of graph nodes
- Congress - Scatterplot of annual income of congressmen in different states
- DataMountain - Spatial arrangement of book covers
- FisheyeMenu - Focus+Context list of numbers
- GraphView - Graph vis with adjustable parameters for layout
- RadialGraphView - Radial graph of a social network
- ScatterPlot - Scatterplot of Iris dataset
- TreeMap - Treemap vis
- TreeView - DOITree
- ZipDecode - Vis of US zip codes

1) Go to "build" folder

2) Launch demo of interest

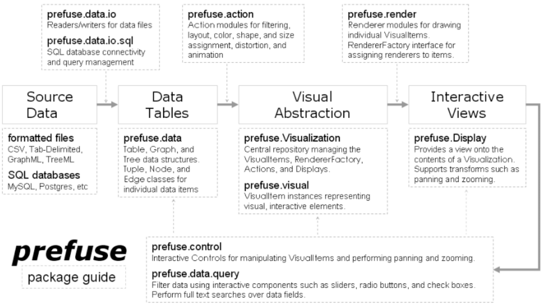
```
java -cp prefuse.jar:demos.jar prefuse.demos.<name of demo here>
```

InfoVis Reference Model



Raw Data: idiosyncratic formats
Data Transformations: Mapping raw data into an organization appropriate for visualization
Data Tables: relations (cases by variables) + metadata
Visual Mappings: Encoding abstract data into a visual representation
Visual Structures: spatial substrates + marks + graphical properties
View Transformations: Changing the view or perspective onto the visual presentation
Views: graphical parameters (position, scaling, clipping, ...)
Human Interaction: User influence at any level

Basic Architecture


prefuse.data.io
Readers/writers for data files
prefuse.data.io.sql
SQL database connectivity and query management
prefuse.action
Action modules for filtering, layout, color, shape, and size assignment, distortion, and animation
prefuse.render
Renderer modules for drawing individual VisualItems. RendererFactory interface for assigning renderers to items.
Source Data
Data Tables
Visual Abstraction
Interactive Views
formatted files
CSV, Tab-Delimited, GraphML, TreeML
SQL databases
MySQL, Postgres, etc
prefuse.data
Table, Graph, and Tree data structures. Tuple, Node, and Edge classes for individual data items
prefuse.Visualization
Central repository managing the VisualItems, RendererFactory, Actions, and Displays.
prefuse.visual
VisualItem instances representing visual, interactive elements.
prefuse.Display
Provides a view onto the contents of a Visualization. Supports transforms such as panning and zooming.
prefuse
package guide
prefuse.control
Interactive Controls for manipulating VisualItems and performing panning and zooming.
prefuse.data.query
Filter data using interactive components such as sliders, radio buttons, and check boxes. Perform full text searches over data fields.

Source Data & Data Tables

Source Data

- Raw data
  - formatted files
  - databases
  - etc.

Data Tables

- internal storage and management for data read from source data
- table columns are typed
- each row contains a data record (tuple)
- each column contains values for a named data field with a specific data-type
- graphs and tables are internally also stored as tables (nodes, edges)
- Data records do not contain any visual information like assignment attributes or color settings.
- Instead, own visual analogs are created

Visual Abstraction

Visualizations are created by generating visual representations of data records (VisualItems) in data tables.

central data structure: **Visualization**

- manages mappings between source data and VisualItems
  - Table<-->VisualTable
  - Graph<-->VisualGraph
  - Tree<-->VisualTree
- manages **VisualItems**
  - visual representation of data elements
  - interactive visual object
  - properties of source data + visual properties
    - Location, color, size, shape, font
- specialization of VisualItems into NodeItems and EdgeItems for graphs

Process is called **Visual Mapping**

Visual Abstraction 2

Specific visual mappings are provided by **Action** modules

Actions are independent processing modules that operate on the VisualItem instances in a Visualization

- setting item visibility, computing layouts, assigning color values, etc.
- can be grouped into **ActionLists**
- Actions can be run once or repeatedly over a time interval, controlled by an ActivityManager

Built-in Actions

Assignment

- ColorAction
- DataColorAction
- SizeAction
- DataSizeAction
- ShapeAction
- DataShapeAction

Filter

- VisibilityFilter
- GraphDistanceFilter
- FisheyeTreeFilter

Layout

- AxisLayout
- AxisLabelLayout
- GridLayout
- CircleLayout
- StackedAreaChart
- RandomLayout
- SpecifiedLayout
- CollapsedStackLayout
- CollapsedSubtreeLayout

Graph/Tree Layout

- BalloonTreeLayout
- ForceDirectedLayout
- FruchtermanReingoldLayout
- NodeLinkTreeLayout
- RadialTreeLayout
- SquarifiedTreeMapLayout

Distortion

- BifocalDistortion
- FisheyeDistortion

Animation

- VisibilityAnimator
- LinearAnimator
- PolarAnimator
- ColorAnimator
- FontAnimator
- SizeAnimator

Interactive Views

actual drawing of VisualItems is done by **Renderers**

- responsible for drawing items and computing item bounds

choice of Renderer is done by **RendererFactory** that is assigned to a Visualization

Display component acts as a camera onto the contents of a Visualization

- is the component where the actual drawing takes place
- draws all the items within its current view, and can be panned, zoomed, and rotated
- first-class user interface components
- can be added into Java applications and applets

single Visualization can be associated with multiple Display instances

- multiple views, overview + detail, small multiples

Interactive Views 2

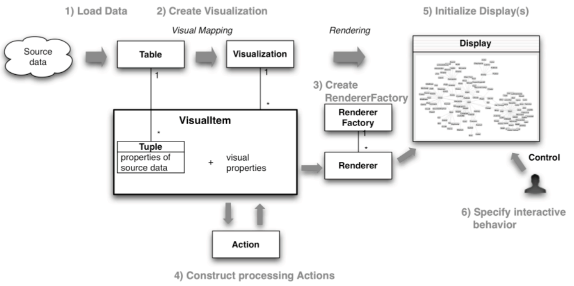
Controls for user interaction

- process mouse or keyboard actions on the Display and on individual VisualItems
- built-in controls
- selecting focus items
- dragging items around
- panning
- zooming
- rotating

Dynamic Query Bindings

- create a binding between a column of table data and an expression Predicate (or query) over that column
- bindings can automatically generate appropriate user interface components (e.g., sliders, radio buttons, check boxes, text search boxes, etc)

Application Building Overview



1) Load Data

2) Create Visualization

Visual Mapping

Rendering

5) Initialize Display(s)

Source data

Table

Visualization

Display

3) Create RendererFactory

Renderer Factory

VisualItem

Tuple

properties of source data

+ visual properties

Renderer

Control

6) Specify interactive behavior

Action

4) Construct processing Actions

1) Load data

data sources

- File
- Database
- Custom

prefuse data structures

- Table
- Graph
- Tree

2) Create Visualization

maps loaded data to Visual Abstraction

Tables, Graphs, and/or Trees are added to the Visualization

manages **VisualItems**

- visual representation of data elements
- interactive visual object
- properties of source data + visual properties

3) Create RenderFactory and register with Visualization

is responsible for assigning Renderers to VisualItems

Renderers do the actual drawing of VisualItems

DefaultRendererFactory

- EdgeRenderer for any EdgeItems
  - straight-line edges by default
- ShapeRenderer for all other items
  - draws items as basic shapes such as squares and triangles

4) Construct processing Actions (Visualization Operators)

operate on the visual abstraction

e.g., setting the location, color, size, and shape of visual items or animating these properties between different configurations

5) Initialize Display(s)

for viewing and manipulating visual items

6) Specify interactive behavior

by adding Controls to the Displays

Search and filtering over data items can be added using "dynamic query bindings"

Coordinates in prefuse

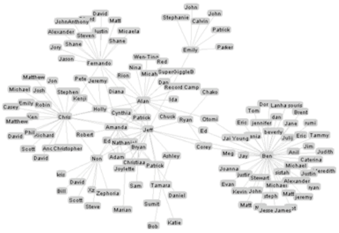
two different coordinate systems

- absolute coordinates
  - device-independent, logical coordinates
  - all visual attributes like positions or sizes are defined in absolute coordinates
- view coordinates
  - device-dependent (screen) coordinates

Transformations between absolute and view coordinates are done automatically by Java painting routines

Example 1

Built-in example network visualization "Example.java"



1) Load data

data source: file "socialnet.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
 <!--  An excerpt of an egocentric social network  -->
 <graphml
xmlns="http://graphml.graphdrawing.org/xmlns">
 <graph edgedefault="undirected">

 <!-- data schema -->
 <key id="name" for="node" attr.name="name"
attr.type="string"/>
 <key id="gender" for="node" attr.name="gender"
attr.type="string"/>

 <!-- nodes -->
 <node id="1">
  <data key="name">Jeff</data>
  <data key="gender">M</data>
  </node>
 <node id="2">
  <data key="name">Ed</data>
  <data key="gender">M</data>
 </node>
 <node id="3">
  <data key="name">Christiaan</data>
  <data key="gender">M</data>
 </node>
 ...

 <!-- edges -->
 <edge source="1" target="2"></edge>
 <edge source="1" target="3"></edge>
 <edge source="1" target="4"></edge>
 <edge source="1" target="5"></edge>
 <edge source="1" target="6"></edge>
 ...

 </graph>
 </graphml>
```

format: GraphML

parse file into **Graph** data structure using the **GraphMLReader**

```
Graph graph = null;
 try {
    graph = new
GraphMLReader().readGraph("socialnet.xml");
 } catch ( DataIOException e ) {
    e.printStackTrace();
    System.err.println("Error loading graph.
Exiting...");
    System.exit(1);
 }
```

2) Create Visualization

create **Visualization** object

add graph to visualization

```
// add the graph to the visualization as the data group "graph"
// nodes and edges are accessible as "graph.nodes" and "graph.edges"
 Visualization vis = new Visualization();
 vis.add("graph", graph);
```

3) Create RenderFactory and register with Visualization

create a new **LabelRenderer** (to see text labels on the nodes)

create a new **DefaultRendererFactory**

- uses the new label renderer as the default renderer for all non-edge items
- all **EdgeItems** will use the default **EdgeRenderer**

```
// draw the "name" label for NodeItems
 LabelRenderer r = new LabelRenderer("name");
 r.setRoundedCorner(8, 8); // round the corners

 // create a new default renderer factory
 // return our name label renderer as the default for all non-EdgeItems
 // includes straight line edges for EdgeItems by default
 vis.setRendererFactory(new DefaultRendererFactory(r));
```

4) Construct processing Actions

setting up visual encodings by creating Action modules that process the VisualItems in the Visualization

a) ColorActions

b) Animated Layout

c) Add ActionLists to Visualization

a) ColorActions

each **VisualItem** supports three color values by default

- stroke color
- fill color
- text color

assign colors based on the gender of people in the social network

- pink for females and baby blue for males
- create a **DataColorAction** that computes the color assignment
- constructor
- name of the data group to process (in this case graph.nodes)
- name of the data field on which to base the encoding (in this case gender)
- data type of the field (nominal, ordinal, numeric)
- color field to set (stroke, fill, text)
- optional color palette

assign the colors for the node text to black and the stroke color for edges to a light gray

create an **ActionList** instance that groups all the color assignment actions into a single executable unit

a) Color Actions 2

```
// create our nominal color palette
 // pink for females, baby blue for males
 int[] palette = new int[] {
     ColorLib.rgb(255,180,180), ColorLib.rgb(190,190,255)
 };
 // map nominal data values to colors using our provided palette
 DataColorAction fill = new DataColorAction("graph.nodes", "gender",
     Constants.NOMINAL, VisualItem.FILLCOLOR, palette);
 // use black for node text
 ColorAction text = new ColorAction("graph.nodes",
     VisualItem.TEXTCOLOR, ColorLib.gray(0));
 // use light grey for edges
 ColorAction edges = new ColorAction("graph.edges",
     VisualItem.STROKECOLOR, ColorLib.gray(200));

 // create an action list containing all color assignments
 ActionList color = new ActionList();
 color.add(fill);
 color.add(text);
 color.add(edges);
```

b) Animated Layout

All Action instances can either be parameterized to run once (the default), or to run repeatedly within a given time duration

- continuous update by setting parameter to "INFINITY"

add a **ForceDirectedLayout** to assign the spatial positions of the elements of the graph

add a **RepaintAction** to signal that any Displays should be repainted after the layout has been recomputed

```
// create an action list with an animated layout
 // the INFINITY parameter tells the action list to run indefinitely
 ActionList layout = new ActionList(Activity.INFINITY);
 layout.add(new ForceDirectedLayout("graph"));
 layout.add(new RepaintAction());
```

c) Add ActionLists to Visualization

```
// add the actions to the visualization
 vis.putAction("color", color);
 vis.putAction("layout", layout);
```

5) Initialize Display(s)

create a Display for the visualized data

```
// create a new Display that pull from our Visualization
 Display display = new Display(vis);
 display.setSize(720, 500); // set display size
```

6) Specify interactive behavior

add three interactive controls to the Display

- **DragControl** for dragging VisualItems around with a left-click mouse drag
- **PanControl** for moving the Display region with a left-click mouse drag on the Display background
- **ZoomControl** for zooming the display in or out with a vertical right-click mouse drag

default settings of the Controls

- mouse button used to trigger the control and other settings can be changed by using alternative constructors.

```
display.addControlListener(new DragControl()); // drag items around
display.addControlListener(new PanControl());  // pan with background left-drag
display.addControlListener(new ZoomControl()); // zoom with vertical right-drag
```

7) Launching the visualization

add the **Display** to a new application window

- create a new **JFrame** instance

run the color assignment action list

start continuously-running layout list

```
// create a new window to hold the visualization
 JFrame frame = new JFrame("prefuse example");
 // ensure application exits when window is closed
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.add(display);
 frame.pack();           // layout components in window
 frame.setVisible(true); // show the window

 vis.run("color");  // assign the colors
 vis.run("layout"); // start up the animated layout
```

Compile & Run

Compile

```
javac -cp lib/prefuse.jar Example.java
```

Run

```
java -cp lib/prefuse.jar:. Example
```

Example 2

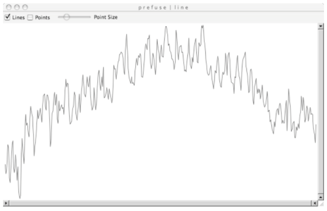
Line Chart

Climate data (Avg. temparatures over the course of a year)

draw points and lines

- points can be drawn directly using axis layouts
- lines have to be created from points "manually"

range sliders for both axes



Documentation & Help

Manual

- http://prefuse.org/doc/manual/

FAQ

- http://prefuse.org/doc/faq/

Forum

- http://sourceforge.net/forum/forum.php?forum_id=343013

API Documentation

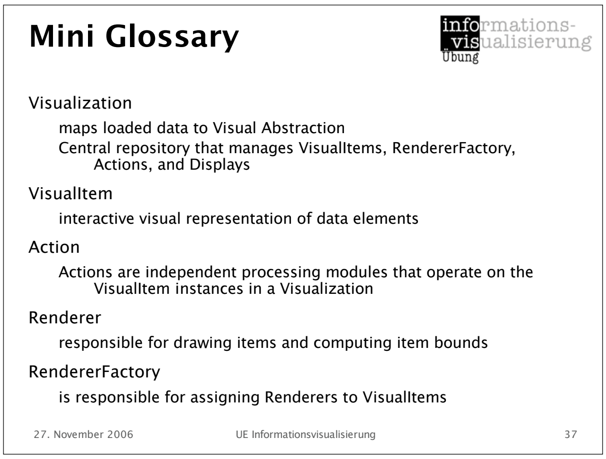
- http://prefuse.org/doc/api/

Demos

- included in download package

InfoVis:Wiki page

- http://www.infovis-wiki.net/index.php/Prefuse



Mini Glossary

Visualization

- maps loaded data to Visual Abstraction
- Central repository that manages VisualItems, RendererFactory, Actions, and Displays

VisualItem

- interactive visual representation of data elements

Action

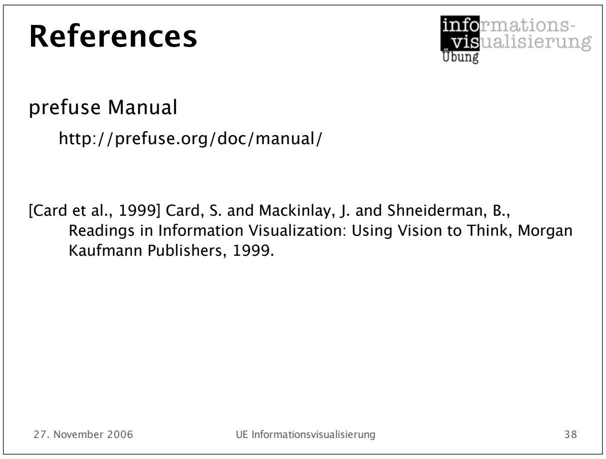
- Actions are independent processing modules that operate on the VisualItem instances in a Visualization

Renderer

- responsible for drawing items and computing item bounds

RendererFactory

- is responsible for assigning Renderers to VisualItems

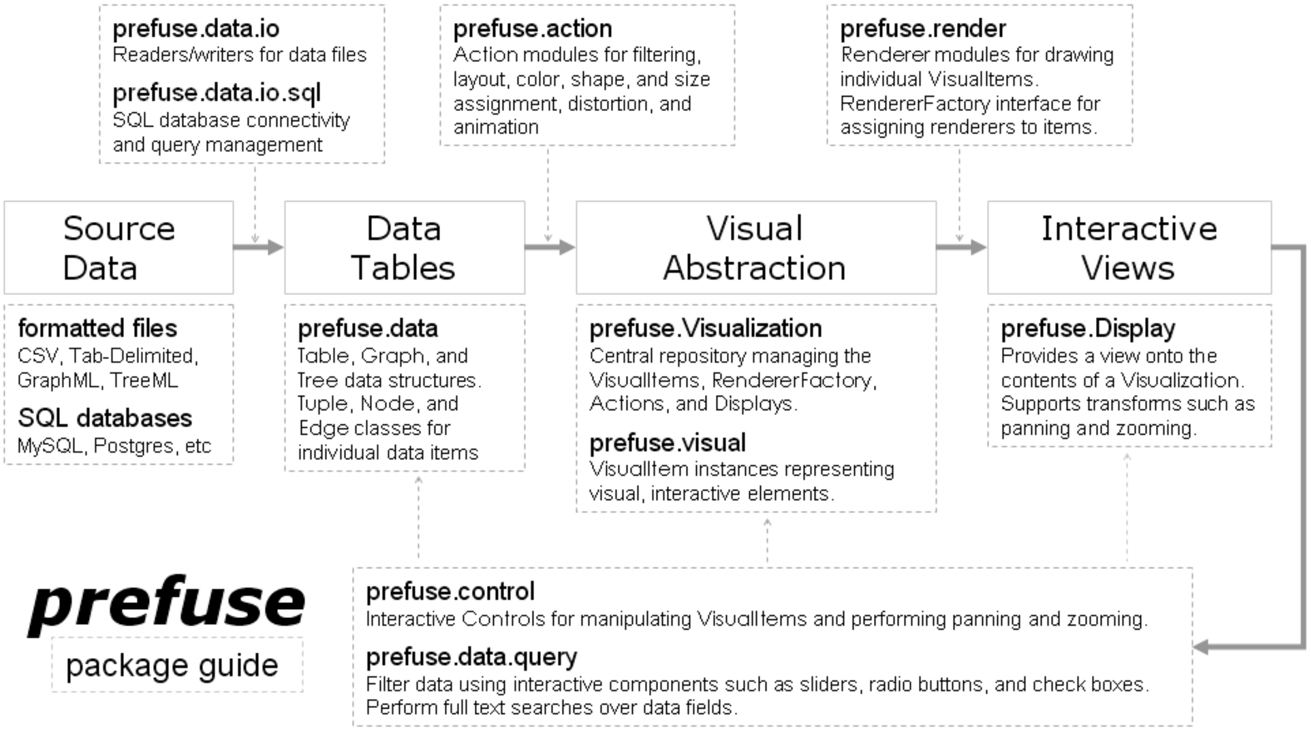


References

prefuse Manual

- http://prefuse.org/doc/manual/

[Card et al., 1999] Card, S. and Mackinlay, J. and Shneiderman, B., Readings in Information Visualization: Using Vision to Think, Morgan Kaufmann Publishers, 1999.



prefuse.data.io
Readers/writers for data files

prefuse.data.io.sql
SQL database connectivity and query management

prefuse.action
Action modules for filtering, layout, color, shape, and size assignment, distortion, and animation

prefuse.render
Renderer modules for drawing individual VisualItems. RendererFactory interface for assigning renderers to items.

Source Data

Data Tables

Visual Abstraction

Interactive Views

formatted files
CSV, Tab-Delimited, GraphML, TreeML

SQL databases
MySQL, Postgres, etc

prefuse.data
Table, Graph, and Tree data structures. Tuple, Node, and Edge classes for individual data items

prefuse.Visualization
Central repository managing the VisualItems, RendererFactory, Actions, and Displays.

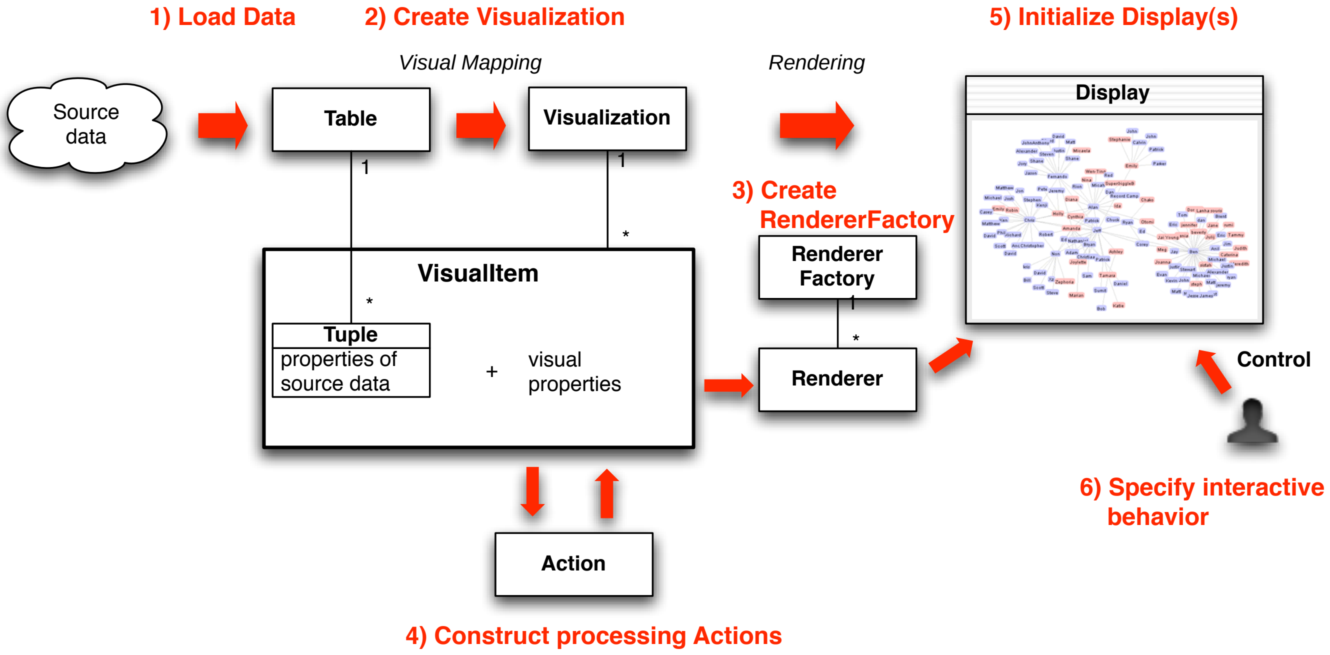
prefuse.visual
VisualItem instances representing visual, interactive elements.

prefuse.Display
Provides a view onto the contents of a Visualization. Supports transforms such as panning and zooming.

prefuse
package guide

prefuse.control
Interactive Controls for manipulating VisualItems and performing panning and zooming.

prefuse.data.query
Filter data using interactive components such as sliders, radio buttons, and check boxes. Perform full text searches over data fields.


1) Load Data
2) Create Visualization
5) Initialize Display(s)
Visual Mapping
Rendering
Source data
Table
Visualization
Display
1
1
*
3) Create RendererFactory
VisualItem
*
Tuple
properties of source data
+
visual properties
Renderer Factory
1
*
Renderer
Control
6) Specify interactive behavior
Action
4) Construct processing Actions