Xcos Automatic Layout

JGraphX

Name: Chenfeng Zhu
Mentor: Mr. David Clément
Mentor: Mr. Paul Bignier

1 Introduction

Functionalities:

1) Graph Visualization
2) Graph Interaction:
   JGraphX supports dragging and cloning cells, re-sizing and re-shaping, connecting and disconnecting, dragging and dropping from external sources, editing cell labels in-place.
3) Graph Layouts:
   JGraphX supports a range of tree, force-directed and hierarchical layouts.
4) Graph Analysis

1.1 Core JGraphX Architecture

1.1.1 The JGraphX Model: mxGraphModel

The mxGraph model is the core model that describes the structure of the graph, the class is called mxGraphModel. **mxGraphModel is the underlying object that stores the data structure of the graph**.

Key API Methods:

- ✓ mxGraphModel.beginUpdate() - starts a new transaction or a sub-transaction.
- ✓ mxGraphModel.endUpdate() - completes a transaction or a sub-transaction.
- ✓ mxGraph.addVertex() - Adds a new vertex to the specified parent cell.
- ✓ mxGraph.addEdge() - Adds a new edge to the specified parent cell.

Changes made outside of this update scope take immediate effect and send out the notifications immediately.

1.1.2 The Transaction Model

There is a counter in the model that increments for every beginUpdate call and decrements for every endUpdate call. After increasing to at least 1, when this count reaches 0 again, the model transaction is considered complete and the event notifications of the model change are fired.

Below is a list of the methods that alter the graph model and should be placed, directly or indirectly, with the scope of an update:

- add(parent, child, index)
- remove(cell)
- setCollapsed(cell, collapsed)
- setGeometry(cell, geometry)
- setRoot(root)
- setStyle(cell, style)
- setTerminal(cell, terminal, isSource)
- setTerminals(edge,source,target)
- setValue(cell, value)
- setVisible(cell, visible)

Core API methods:

- ✓ mxGraph.insertVertex(parent, id, value, x, y, width, height, style) – creates and inserts a new vertex into the model, within a begin/end update call.
- ✓ mxGraph.insertEdge(parent, id, value, source, target, style) – creates and inserts a new edge into the model, within a begin/end update call.

1.1.3 The Cell Object: mxCell

mxCell is the cell object for both vertices and edges. mxCell duplicates many of the methods available in the model. The key difference in usage is that using the model methods creates the appropriate event notifications and undo, using the cell makes the change but there is no record of the change.

When creating a new cell, three things are required in the constructor, a value (user object), a geometry and a style. We will now explore these 3 concepts before returning to the cell.

1.1.3.1User Objects

The User object is what gives JGraphX diagrams a context, it stores the business logic associated with a visual cell.

1.1.3.2 Geometry: mxGeometry

The reason for a separate mxGeometry class, as opposed to simply having the mxRectangle class store this information, is that the edges also have geometry information.

The width and height values are ignored for edges and the x and y values relate to the positioning of the edge label. In addition, edges have the concept of control points. These are intermediate points along the edge that the edge is drawn as passing through. The use of control points is sometimes referred to as edge routing.

Core API methods:

- ✓ mxGraph.resizeCell(cell, bounds) – Resizes the specified cell to the specified bounds, within a begin/end update call.
- ✓ mxGraph.resizeCells(cells, bounds) – Resizes each of the cells in the cells array to the corresponding entry in the bounds array, within a begin/end update call.

1.1.3.2.1 Relative Positioning

For vertices in relative mode, (x,y) is the proportion along the parent cell's (width, height) where the cell's origin lies. (0,0) is the same origin as the parent, (1,1) places the origin at the bottom right corner of the parent.

Edge labels in relative mode are palced based on the positioning from the center of the edge. The x-coordinate is the relative distance from the source end of the edge, at -1, to the target end of the edge, at 1. The y co-ordinate is the pixel offset orthogonal from the edge.

1.1.3.2.2 Non-relative Positioning

1.1.3.3 Styles: mxStylesheet

The mxStylesheet holds one object, styles, which is a hashtable mapping style names to an array of styles:

Setting the Style of a Cell:

1) Apply a named style to a vertex.
2) Override a named or default style.
3) Use an unnamed style.

Core API methods:

- ✓ mxGraph.setCellStyle(style, cells) – Sets the style for the array of cells, encapsulated in a begin/end update.
- ✓ mxGraph.getCellStyle(cell) – Returns the style for the specified cell, merging the

styles from any local style and the default style for that cell type.
Create a New Global Style.

1.1.3.4Cell Types

There are two boolean flags on mxCell, vertex and edge, and the helper methods set one of these to true when the cell is created. isVertex(), isEdge() on mxIGraphModel are what the model uses to determine a cell's type, there are not separate objects for either type.

1.1.4 Group Structure

Grouping, within JGraphX, is the concept of logically associating cells with one another.

Core API methods:

- ✓ mxGraph.groupCells(group, border, cells) – Adds the specified cells to the specified group, within a begin/end update
- ✓ mxGraph.ungroupCells(cells) – Removes the specified cells from their parent and adds them to their parent's parent. Any group empty after the operation are deleted. The operation occurs within a begin/end update.

1.1.5 Complexity Management

1.1.5.1Folding

Folding is the collective term used for expanding and collapsing groups

Core API method:

- ✓ mxGraph.foldCells(collapse, recurse, cells) – States the collapsed state of the specificed cells, within a begin/end update.

Folding related methods:

- mxGraph.isCellFoldable(cell, collapse) – By default true for cells with children.
- mxGraph.isCellCollapsed(cell) – Returns the folded state of the cell

1.1.5.2Sub-Graphs, Drill-Down / Step-Up

Sometimes, as an alternative to expand/collapse, or possibly in combination with it, your graph will be composed of a number of graphs, nested into a hierarchy.

Core API methods:

- ✓ mxGraph.enterGroup(cell) – Makes the specified cell the new root of the display area.

- ✓ mxGraph.exitGroup() - Makes the parent of the current root cell, if any, the new root cell.
- ✓ mxGraph.home() - Exits all groups, making the default parent the root cell.

1.1.5.3 Layering and Filtering

Core API method:

- ✓ mxGraph.orderCells(back, cells) – Moves the array of cells to the front or back of their siblings, depending on the flag, within a begin/end update.

1.2 Visual Customization

Within the core JGraphX library, there are a number of mechanisms to define the appearance of cells. These split into vertex customizations and edge customizations

1.2.1 Vertex Customizations

Stencils.

1.2.2 Edge Customizations

Searching.

2 Demo

Develop some demos with JGraphX which implements the functionality required.

2.1 Simple Test with Style

This is a simple test with two Vertices and a defined style.
See org.zhuchenf.demo.Test2Block.java

2.2 Click to find an Optimal Route (unfinished)

This is a demo for optimal routing. There are several Vertices in random position. There are some edges links some of them. When clicking the edge and choosing "optimize", the edge would become an optimal edge.
See org.zhuchenf.demo.OptimizeRoute.java

2.3 Simple Test for Ordering (unfinished)