Diagram Elements 1.0 Component Specification

1. Design

The Diagram Elements component provides a general framework for representing graphically the Diagram Interchange graph nodes that can be added to the diagram view from the Diagram Viewer component. Some graph nodes may act as containers of other nodes, accepting nodes in certain compartments.

Node is the core class of this component. Node is a visual representation of GraphNode in UML Diagram Interchange. It is an extension of SWING JComponent. Node can contain zero or more SelectionCorners, which are embedded on predefined selection bound. The SelectionCorner is a concrete JComponent. It is added as child of Node. With this strategy, the selection corner can paint itself and also have its own mouse event.

Although user can register listeners on Node or SelectionCorner directly, some high level events are also provided. They are node dragged event, which tells the offset of node location, and selection corner dragged event, which tells the change of selection bound. There is a situation, in which events will be deal in a different way. When the diagram viewer is in the state of adding element, the Node may consume the node or pass the event to element behind it. The concrete implementation should implement the event consuming routine.

Because the shape of GraphNode is very different, concrete implementation has the responsibility to tell which kinds of selection corners are required. And also it should override the contains() and paintComponent() method to provide custom looking.

NodeContainer class is an extension of Node. It can contain other nodes. The nodes can be contained directly or be contained in certain compartments.

Because it is a base component, many values are configurable.

1.1 Design Patterns

Composition Pattern – NodeContainer extends Node, and it contains child nodes.

Observer Pattern – System events are listened in this component, and also custom events are triggered for application to listen.

1.2 Industry Standards

JFC Swing

UML 2.0 Diagram Interchange

1.3 Required Algorithms

1.3.1 Get actual location of Node

This algorithm is used to retrieve the component location of Node.

Because the relative position to diagram viewer of visual node must be kept.

The location of component must be determined automatically.

```
// the visual node's position is relative value, need to get the absolute
// position of the visual node

    double x = 0;

    double y = 0;
```

```
    for (GraphElement node = graphNode; node.getContainer() != null; node
= node.getContainer()) {
        x += node.getPosition().getX();
        y += node.getPosition().getY();
    }
    // the component position should be calculated from the relative
   // position of visual node to component node and the absolute
   // position of visual node
    point.x = (int) Math.round(x) - relativePosition.x;
    point.y = (int) Math.round(y) - relativePosition.y;
    return point;
```

1.4 Component Class Overview

Node:

This class represents a graph node in UML diagram. It is an extension of JComponent. All the concrete implementation should draw the visual look by overriding paintComponent() and contains() methods. The selection corners are added as a child component of this node, so we don't need to draw it. But the selection corners are configurable by specifying corners type and selection bound.

When a user changes the size of this Node, there are several values to be set. User should call setSize(), setRelativePosition() and setSelectionBound() to make the node look good.

This class provides two kinds of events. User can register this event as other events. They are selection corners dragged event and node dragged event. Besides the event registration, when the diagram viewer is in the state of adding element, it will treat the event differently.

This class is mutable, and not thread safe.

NodeContainer:

This class represents a node container, which can contain Node instances. It is an extension of Node. It can contain node directly, or contain node under certain compartments.

Besides the functionality provided in Node class, this class only defines the methods to manipulate the contained nodes.

This class is mutable, and not thread safe.

SelectionCorner:

This class represents a selection corner in diagram. It is a concrete SWING JComponent. A selection corner is a circle which can be embedded on element's bound or edge's waypoints.

The radius of circle, center of circle, stroke color and fill color are all configurable via API.

This class is mutable and not thread safe.

LocationEvent:

This class represents a location event. A location event indicates the change of the node location. It contains two attribute, old location and new location. It can be used before or after the location changing.

This class is immutable and thread safe.

SelectionBoundEvent:

This class represents a selection bound event. A selection bound event indicates the change of the node's selection bound. It contains two attribute, old selection bound and new selection bound. It can be used before or after the selection bound changing.

This class is immutable and thread safe.

SelectionCornerType:

This enumeration represents all the types of selection corners. It defines eight values according to eight directions: East, South, West, North, Northeast, Southeast, Northwest, and Southwest.

Enum is thread-safe.

DragEventAdapter:

This is an adapter to handle a series of dragging event. Because we can only get current point of dragging event, last point must be recorded to calculate the offset value. Since the dragging process begins with a mouse press, this class will extend from MouseAdapter and implement MouseMotionListener.

MouseDragged event is left as abstract to compel the user to implement it.

This class is mutable and not thread-safe.

SelectionCornerMouseListener:

This listener listens to the SelectionCorner's mouse event. It will wrap the low level mouse dragged event, and trigger a selection corner moved event.

This class is package private and final; it will be registered in the Node's constructor automatically. So the selection corner moved event could be triggered by default.

This class is mutable and is not thread safe.

NodeMouseListener:

This listener listens to the Node's mouse event. It will wrap the low level mouse dragged event, and trigger a node dragged event. It also reacts to the mouse pressed event to set the node selected.

This class is package private and final; it will be registered in the Node's constructor automatically. So the node dragged event could be triggered by default.

This class is mutable and is not thread safe.

CornerDragListener:

This interface defines the contract that every selection corner drag event listener must follow.

It contains only one method to process the selection corner dragged event with a single SelectionBoundEvent parameter.

NodeDragListener:

This interface defines the contract that every node drag event listener must follow.

It contains only one method to process the node dragged event with a single LocationEvent parameter.

1.5 Component Exception Definitions

No custom exception is defined in this component. Only IllegalArgumentException can be thrown for null arguments. Why there is no custom exception occurs? First this follows other JComponents in JFC Swing. Second, actually, custom exception can’t occur in the implementation.

1.6 Thread Safety

This component is not thread-safe, because most of the classes in this component are mutable except the event classes. Thread-safety is not required. Like many other standard swing methods, thread-safety should be cared by users. And there is one issue we want to discuss further. Event listeners list is used by both the main application thread (adding or removing listeners), and the event dispatching thread (using the listeners). This problem can be solved by listenerList field in JComponent, which is thread-safe. We add the listeners to the list, so thread safe is not a problem here.

If user wants to use this component in a thread manner, instances from this component should be synchronized before their methods are called.

2. Environment Requirements

2.1 Environment

- Development language: Java1.5
- Compile target: Java1.5

2.2 TopCoder Software Components

- Diagram Viewer 1.0

  This component provides a diagram viewer which is used to hold the Node and NodeContainer.

- Diagram Interchange 1.0

  This component defines the data structure of standard UML diagram information. A Node will retrieve the graph information from GraphNode defined in the component.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

NONE

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.diagramviewer.elements

3.2 Configuration Parameters

NONE

3.3 Dependencies Configuration

Put the dependent components under class path.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.

- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Add this package and the related components to the class path.

4.3 Demo

4.3.1 The supporting classes

```
public class RectNode extends Node {

    /**
     * Creates a <code>RectNode</code> with the given graph node.
     *
     * @param graphNode
     *            the <code>GraphNode</code> instance associated to this node
     */
    public RectNode(GraphNode graphNode) {
        // corner types and relative position are specified by default
        super(graphNode, new Point(5, 5), getBound(), getDefaultCornerTypes());

        // make the size larger than actual bound
        super.setSize(getBound().width + 10, getBound().height + 10);
    }

    /**
     * Gets the default selection corner types.
     *
     * @return the default selection corner types
     */
    private static final Collection<SelectionCornerType> getDefaultCornerTypes()
{
        Collection<SelectionCornerType> types = new
ArrayList<SelectionCornerType>();
        types.add(SelectionCornerType.EAST);
        types.add(SelectionCornerType.SOUTH);
        types.add(SelectionCornerType.WEST);
        types.add(SelectionCornerType.NORTH);
        types.add(SelectionCornerType.NORTHEAST);
        types.add(SelectionCornerType.NORTHWEST);
        types.add(SelectionCornerType.SOUTHEAST);
        types.add(SelectionCornerType.SOUTHWEST);
        return types;
    }

    /**
     * Gets the default selection bound.
     *
     * @return the default selection bound
     */
    private static final Rectangle getBound() {
        return new Rectangle(100, 100, 100, 100);
    }

    /**
     * Overrides to paint custom look.
     *
     * @param g
     *            the <code>Graphics</code> to paint on
     */
    protected void paintComponent(Graphics g) {
        // draw a rectangle
```

```
    }

    /**
     * Overrides to define custom shape.
     *
     * @param x
     *            the x coordinate of point
     * @param y
     *            the y coordinate of point
     * @return true if the given point is in the circle, otherwise false
     */
    public boolean contains(int x, int y) {
        return false;
    }

    /**
     * Simply return false.
     *
     * @param event
     *            ignore
     * @return always false
     */
    protected boolean consumeEvent(MouseEvent event) {
        return false;
    }

    public class DefaultContainer extends NodeContainer {

        /**
         * The default constructor.
         */
        public DefaultContainer() {
            super(new GraphNode(), new Point(), new Rectangle(), new
ArrayList<SelectionCornerType>());
        }

        /**
         * Simply return false.
         *
         * @param event
         *            ignore
         * @return always false
         */
        protected boolean consumeEvent(MouseEvent event) {
            return false;
        }
    }
```

4.3.2 Use the Node

```
            // create a node instance

            RectNode node = new RectNode(new GraphNode());


            // add it to diagram viewer

            // <<to be implemented>>


            // register the node drag event
```

```
node.addNodeDragListener(new MockNodeDragListener());

// register the selection corner drag event
node.addCornerDragListener(new MockCornerDragListener());
```

4.3.3 *Use the NodeContainer*

```
// create a node instance
RectNode node = new RectNode(new GraphNode());

// create a NodeContainer instance
NodeContainer container = new DefaultContainer();

// manipulate node directly contained in it

// add a directly contained node
container.add(node);

// get all directly contained nodes
container.getNodes();

// remove a directly contained node
container.remove(node);

// clear all directly contained nodes
container.clearNodes();

// manipulate nodes contained in compartment

// add a node to certain compartment
container.add("Compartment1", node);

// get all nodes contained in certain compartment
container.getNodes("Compartment1");

// remove a node from certain compartment
container.remove(node);
```

```
// clear all nodes in certain compartment
container.clearNodes("Compartment1");

// get all compartments
container.getCompartments();
```

5. Future Enhancements

Trigger event after location changing. Trigger event after bound changing. Add other selection corner shape.