Diagram Edges 1.0 Component Specification

1. Design

The Diagram Edges component provides a general framework for representing graphically the Diagram Interchange graph edges that can be added to the diagram view from the Diagram Viewer component. The edges have configurable edge ends and text fields attached to the actual edge and to the edge ends. The component provides the general behavior of the elements: dragging of elements, updating edge paths, moving edge text fields.

Edge is the core class of this component. Edge is a visual representation of GraphEdge in UML Diagram Interchange. It is an extension of SWING JComponent. Edge can contain two or more way points, which are highlighted as selection corners when the edge is selected. The SelectionCorner is a concrete JComponent. It is added as child of Edge. With this strategy, the selection corner can paint itself and also have its own mouse event.

The edge line can have different styles due to its meaning. The LineStyle class is provided for this purpose. We can configure the dash length and the blank spaces between dashes. Edge can have zero to two endings, which can also have different styles due to the edge meaning. The ending is defined by EdgeEnding abstract class. User should extend this class to provide custom looking of ending.

TextFields can be attached to Edge, which can be anchored to the endings or to the line of edge. Build-in text dragging functionality is provided. User can register listener to listen the events on TextFields. It is very easy because TextField is a kind of SWING JLabel.

Although user can register listeners on Edge or SelectionCorner directly, some high level events are also provided. They are edge dragged event, which tells the offset of point on edge, and way point dragged event, which tells the change of way point. 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 Edge may consume the event or pass the event to element behind it. The concrete implementation should implement the event consuming routine.

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

1.1 Design Patterns

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 absolute position of element.

This algorithm is used to retrieve the absolute position of any element in diagram.

Because we set the size of Edge as large as the parent diagram viewer, the position in the Edge component has the same value with the absolute position in UML diagram representation. But the position value store in UML diagram is relative value. So we must calculate the absolute position.

```
// the visual node's position is relative value
// get the ABSOLUTE position of the visual node
double x = 0, y = 0;
GraphElement element = edgeNode;
while (element.getContainer() != null) {
  x += element.getPosition().getX();
  y += element.getPosition().getY();
  element = element.getContainer();
}
```

1.3.2 *Compute the distance between point and segment.*

It is a basic geometry problem. Please see ***lbackstrom***'s tutorial on TopCoder website.

http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=geometry1

This algorithm used in several places of this component.

One of the usage is to calculate the anchorage of text fields. This works as follows:

```
shortestDistance := INF;
anchorage := (0, 0)
foreach segment in edge
  calculate the distance between the point of segment.
  if current distance is less than shortestDistance
    shortestDistance := current distance
    anchorage := the nearest point on segment.
set the anchorage to text field
```

The second usage is in RectangleConnector. It has the similar process. You can iterate through the four edges of rectangle to find the nearest point.

1.3.3 *Keep the relative position of text fields to the edge.*

There are two situations here.

One is the edge ending is moved. This can only affect one segment of the edge. All the text fields anchored on the ending or the affected segment should rotate with the segment. The other is one waypoint is moved. This can affect two segments. Text fields anchored on the two affected segments should be rotated with the segments.

Rotating text field according to a segment is also a common geometry problem. It just be like rotating a triangle. Again, you can find answer from the tutorial.

This algorithm will be used in Edge#notifyAddWayPoint(), notifyRemoveWayPoint() and notifyChangeWayPoint() methods. In all these methods, the way points are changed. You should adjust the text fields according the waypoints.

1.4 Component Class Overview

Edge:

This class represents an edge in UML diagram. It is an extension of JComponent. Actually, it only draws the line by itself. All the selection corners, line endings, and anchored text fields are represented as its children. The line also can be configured by line style. Besides the line style, there are two other configurable properties. Active width represents the active area of the edge, and align deviation defines the deviation in which two segments will be aligned automatically.

This class provides two kinds of events. User can register this event as other events. They are waypoint dragged event and edge 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..

LineStyle:

This class represents the line style. A line can be continuous or dashed. The dashes can be configurable. The length of the dash and the length of the blank space between the dashes can both be configured.

This class is mutable and not thread-safe.

EdgeEnding:

This class represents an ending of Edge. It extends JComponent, and also adds some other attributes. As their names, angle attribute defines the angle of this edge ending, and ending point attribute represents the ending point of this edge.

The direction from west to east has zero angles, and it is enlarged anti-clock wisely.

This class is abstract, concrete implementation should override the contains() and paintComponent() method to provide custom looking.

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.

TextField:

TextField class represents the text fields anchored on Edge. It extends SWING JLabel class, and adds some other attributes and operations to accommodate our usage.

It contains a reference of GraphNode, which represents this text field in UML Diagram. The GraphNode also provides the font definition of this text field. Anchorage attribute represents the anchored point on the edge. AnchorType attribute defines how this text field is anchored to the edge. A Boolean flag is also defined to tell whether this text field is selected or not.

This class is mutable and not thread safe.

AnchorType:

This enumeration represents all the anchor types of text fields. It defines three values: LeftEnding, RightEnding and Line.

Enum is thread-safe.

EdgeMouseListener:

This listener listens to the Edge's mouse event. First, it will wrap the low level mouse dragged event, and trigger an edge moved event. Second, if the diagram viewer is in the state of adding new element, the event will be treated differently.

This class is package private; it will be registered in the Edge's constructor automatically. So the edge moved event could be triggered by default.

This class is mutable and is not thread safe.

TextFieldMouseListener:

This listener listens to the TextField's mouse event. It will react to the dragging event to move the position of TextField.

This class is package private; it will be registered in the TextField's constructor automatically. So the build-in text fields dragging functionality could be provided.

This class is mutable and is not thread safe.

WayPointMouseListener:

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

This class is package private; it will be registered in the Edge's constructor automatically. So the waypoint moved event could be triggered by default.

This class is mutable and is not thread safe.

WayPointEvent:

This class represents event which is related to waypoint on edge. It can be position changing of wayPoint or creating a new wayPoint.

This class is immutable and thread-safe..

WayPointListener <<interface>>:

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

It contains only one method to process the waypoint dragged event with a single WayPointEvent parameter.

EdgeDragListener <<interface>>:

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

It contains only one method to process the edge dragged event with a single WayPointEvent parameter..

Connector <<interface>>:

This interface defines the contract every connector must follow. Edge should be connected to the closest point of node. However, nodes have different shapes, so we need to compute the connection point differently.

It contains only one method, which receives the nearest waypoint of edge and will return the connector waypoint.

RectangleConnector <<connectors sub-package>>:

This is an implementation of connector interface. It is the default connector for rectangle shaped element.

This class is mutable, and not thread-safe.

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.

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 the parent container used to hold the edge in this component.

- Diagram Interchange 1.0

  This component provides the data structures which hold all the standard UML diagram informations.

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.edges [main package]

com.topcoder.gui.diagramviewer.edges.connectors [connectors package]

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

```
// implement a concrete EdgeEnding

 public class TriangleEdgeEnding extends EdgeEnding {
 /**
  * <p>
  * Construct a TriangleEdgeEnding instance.
  * </p>
  */
 public TriangleEdgeEnding() {
 }

 /**
  * <p>
  * Construct a TriangleEdgeEnding with angle and ending point given.
  * </p>
  *
  * <p>
  * Note, the angle should be in radian representation.
  * The endingPoint is cloned.
  * </p>
  *
  * @param angle the angle of edge ending
  * @param endingPoint the ending point of edge.
  *
  * @throws IllegalArgumentException if endingPoint is null.
  */
 public TriangleEdgeEnding(double angle, Point endingPoint) {
     super(angle, endingPoint);
```

```
    }

    /**
     * <p>
     * This method draws the arrow at the specified ending point.
     * </p>
     *
     * @param g the graphics to paint on
     */
    protected void paintComponent(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;

        Point endingPoint = getEndingPoint();

        double x1 = endingPoint.getX();
        double y1 = endingPoint.getY();

        Polygon poly = new Polygon();
        poly.addPoint((int) x1 - 10, (int) y1 - 5);
        poly.addPoint((int) x1, (int) y1);
        poly.addPoint((int) x1 - 10, (int) y1 + 5);

        AffineTransform tx = AffineTransform.getRotateInstance(getAngle(), x1, y1);
        g2d.fill(tx.createTransformedShape((Shape) poly));
    }

    /**
     * <p>
     * This method returns false always because it is not interested at any swing events.
     * </p>
     *
     * @param x the x-coordinate value
     * @param y the y-coordinate value
     *
     * @return false always
     */
    public boolean contains(int x, int y) {
        return false;
    }
}
```

// Implement a generalization Edge.

// like (-----------|>)

```
     public class GeneralizationEdge extends Edge implements WayPointListener,
     EdgeDragListener {
    /**
     * <p>
     * Construct a GeneralizationEdge with graph edge and right ending given.
     * </p>
     *
     * @param graphEdge the associated graphEdge
     * @param rightEdgeEnding the right ending of this edge
     *
     * @throws IllegalArgumentException if graphEdge is null, or the size of way points
     * of the graph edge is less than two
     */
    public GeneralizationEdge(GraphEdge graphEdge, TriangleEdgeEnding rightEdgeEnding) {
        super(graphEdge, new LineStyle(1, 0), null, rightEdgeEnding);

        this.addEdgeDragListener(this);
        this.addWayPointDragListener(this);
    }

    /**
     * <p>
     * This method implements the abstract method of Edge class.
     * </p>
     *
     * @param e the mouse event
     *
     * @return false always
     */
    protected boolean consumeEvent(MouseEvent e) {
        // always ignore the event
        return false;
    }

    /**
     * <p>
     * This method implements the contract of WayPointListener interface.
     * </p>
     *
     * <p>
     * This method prints the received event to the console.
     * </p>
     *
     * @param e the way point event
     */
    public void wayPointDragged(WayPointEvent e) {
        System.out.printf("Receive a way point dragged event, "
            + "the old position is %s, the new position is %s, the offset is %s.\r\n",
         e.getOldPosition(),e.getNewPosition(), e.getOffset());
    }

    /**
     * <p>
     * This method implements the contract of EdgeDragListener interface.
     * </p>
     *
     * <p>
     * This method prints the received event to the console .
     * </p>
     *
     * @param e the way point event
     */
    public void edgeDragged(WayPointEvent e) {
        System.out.printf("Receive a edge dragged event, "
            + "the old position is %s, the new position is %s, the offset is %s.\r\n",
         e.getOldPosition(),e.getNewPosition(), e.getOffset());
    }

}
```

// The normal usage of the Edge class

```
        GraphEdge graphEdge = new GraphEdge();
        GraphNode graphNode = new GraphNode();
        Rectangle rectangle = new Rectangle();

        // add there waypoints
        graphEdge.addWaypoint(TestHelper.createDiagramInterchangePoint(100, 100));
        graphEdge.addWaypoint(TestHelper.createDiagramInterchangePoint(200, 200));
        graphEdge.addWaypoint(TestHelper.createDiagramInterchangePoint(300, 400));

        TriangleEdgeEnding triangleEdgeEnding = new TriangleEdgeEnding();
        // create an edge
        Edge edge = new GeneralizationEdge(graphEdge, triangleEdgeEnding);

        // register listeners to listen events
        WayPointListener wayPointListener = new SimpleWayPointListener();
        EdgeDragListener edgeDragListener = new SimpleEdgeDragListener();

        edge.addWayPointDragListener(wayPointListener);
        edge.addEdgeDragListener(edgeDragListener);

        // manipulate text fields
        // add a text field on the line
        edge.addTextField(graphNode, "class1->class2", AnchorType.Line);
```

```
// remove a text field
edge.removeTextField(graphNode);

// use a connector
edge.setLeftConnector(new RectangleConnector(rectangle));
```

5. Future Enhancements

Trigger event after waypoint changing. Trigger event after edge dragging.