Diagram UML Auxiliary Elements 1.0 Component Specification

1. Design

The Diagram UML Auxiliary Elements component provides the comment, free text and polyline graphical diagram elements.

All the nodes in this component have fill color, stroke color, font color, font size, and font family properties. So BaseNode class is defined to handle those properties. Besides the properties, BaseNode class also provides node bound change event, which is used to indicate the node's bound is changed for some reason.

Although the GraphNodes representing comment and free text are different, they are very similar in a user perspective. User can enter text into both of them with double clicking to activate the edit box. So the FreeTextNode class and CommentNode class have a common base class named TextNode. It provides all the common function of comment and free text. Besides the common aspects, CommentNode has a wrapper which has a folded corner, and FreeTextNode is free as the name. Another difference is the place to retrieve text from.

PolyLine is also represented as a concrete Node. With the help of the function already implemented in BaseNode class, this class is easy to be implemented.

CommentLinkEdge is provided to link other diagram elements to CommentNode. This is a simple Edge with no text field attached and with no arrow in the ending.

1.1 Design Patterns

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

Template Method Pattern – The method used to retrieve text to display in TextNode is defined as a template method. Concrete class will retrieve the text in their own way.

1.2 Industry Standards

JFC Swing

UML 2.0 Diagram Interchange

1.3 Required Algorithms

1.3.1 Compute the connection point to CommentNode.

This problem comes from the folded corner. Because of the folded corner, we can't treat it as a simple rectangle. The shape of comment node should be a polygon, so we just need to find the closest point to the polygon edges.

This works as follows:

shortestDistance := INF;

connectionPoint := (0, 0)

foreach segment in edge

  calculate the distance between the point and segment.

  if current distance is less than shortestDistance

    shortestDistance := current distance

    connectionPoint := the nearest point on segment.

Return the connection point

To find the minimum distance between a point and a segment is a basic geometry problem. Please see ***lbackstrom***'s tutorial on TopCoder website.

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

1.4 Component Class Overview

BaseNode:

This is the base Node of this component. It defines the common behaviors of all nodes in this component.

It contains four properties. They are fill color, stroke color, font color, and font. Especially, JComponent#setFont and getFont methods are reused to support the font property. A PropertyMapping instance is also contained to provide the property name mappings.

BoundaryChangeEvent would be triggered by this node. It indicates the node's boundary is changed.

This class is mutable, and not thread-safe.

TextNode:

This abstract class defines a Node which contains only text values. It provides some methods to display the text, and to measure the text.

TextChangeEvent could be triggered by this class. It indicates the text is expected to be changed.

This class is mutable, and not thread-safe.

CommentNode:

This node represents a Comment in UML diagram. A comment is displayed as a rectangle with one folded corner. The comment text resides in the rectangle.

The major function in this class implementation is to provide a custom look to CommentNode. All other functions are defined in base TextNode class.

This class is mutable, and not thread-safe.

FreeTextNode:

This node represents free text in UML diagram. Free text is displayed in a rectangle with no border. The free text resides in the rectangle.

The major function in this class implementation is to provide a custom look to FreeTextNode. All other functions are defined in base TextNode class.

This class is mutable, and not thread-safe.

PropertyMapping:

This class contains the property mappings. A property may have different key for the same meaning, because property names are not standardized. This class maps a property name to an actual property key. It can load such mappings from configuration.

This class is mutable, and not thread safe.

PolyLineNode:

This node represents a polyline in diagram interchange. It extends from base node to use the properties defined in it.

This class is immutable but its base class is not thread-safe.

CommentLinkEdge:

This class represents CommentLink edge. It is rather simple that neither edge ending nor text field is required. This edge has a dashed line.

This class is not thread safe, since the super class is not thread safe.

PopupMenuTrigger:

This is a event listener which listens to mouse right clicked event. If the event occurs, popup menu would be shown. This event listener will be registered to every Node or Edge in this component.

This class is immutable, and thread-safe.

TextChangeEvent:

This is an event object used to indicate text of text node is changed. It contains three properties. They are the Node whose text is changed, the old text value, the new text value. Note, the Node property can be retrieved by getSource().

This class is immutable, and thread-safe.

TextChangeListener<<interface>>:

This interface defines the contract that every text change event listener must follow. Note, the event would be triggered before the text is actually changed.

It contains only one method to process the text change event with a single TextChangeEvent parameter.

EditBoxListener:

This listener is used to listen to events from edit box in diagram viewer. It must be attached to a TextNode. It would fire a text change event when new text is entered, or display the original text if new text is canceled.

This class is immutable, and thread-safe.

CommentConnector:

This a connector defined for comment. The getShape() method of the node should return an instance of Polygon. Otherwise, the connection point can be an unexpected value.

This class is immutable and thread-safe.

BoundaryChangeListener <<interface>>:

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

It contains only one method to process the bound changed event with a single BoundaryChangeEvent parameter..

BoundaryChangeEvent <<interface>>:

This is an event object used to indicate bound of node is changed. It contains four properties. They are the Node whose bound is changed, the old bound value, the new bound value, and some message. Note, the Node property can be retrieved by getSource().

This class is immutable, and thread-safe.

1.5 Component Exception Definitions

ConfigurationException:

This exception is used to indicate the configuration errors. It may indicate namespace in ConfigManager doesn't exist, or some required properties are not configured.

It can be thrown from the constructor of PropertyMapping.

IllegalGraphNodeException:

This exception is used to indicate some GraphNode is illegal in specific situation. For example, a name compartment GraphNode is given to stereotype compartment constructor. It could be thrown from the constructors, which accept a GraphNode as parameter.

Because this exception may be thrown in many places of application, we make it as a runtime exception. The other reason is that this exception can never happen in normal usage.

No other custom exception is defined in this component. Only IllegalArgumentException can be thrown for null arguments.

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 addressed 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 that list, and the listeners will be synchronized.

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 defines the diagram used to contain all the nodes and edges.

- Diagram Interchange 1.0

This component defines the data structure of standard UML diagram interchange. We will retrieve location, size, color, and other information from this component.

- Diagram Elements 1.0

  This component defines the base class for all nodes. Comment node and free text node will extend the base node in that component.

- Diagram Edges 1.0

  This component defines the base class for all edges. Comment link edge will extend the base edge in that component.

- Configuration Manager 2.1.5

  We will use this component to load property names configuration from file.

- Base Exception 1.0 - This class provides the base exception for the exceptions of our 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.uml.auxiliaryelements

3.2 Configuration Parameters

Parameter	Description	Values
[property name]	A fixed value which represents the key used to retrieve corresponding graph node property keys. This value should be known by application. It is OPTIONAL.	StrokeColor
[property value]	A custom value which represents the actual key used to retrieve graph node property value. This value may be different among different xmi files. It is OPTIONAL.	Stroke_color, Stroke_color_property, Or any other values.

Sample configuration: docs/sample.xml

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

Note, following section tells how to use PolylineNode.

```
        // create a polyline node
        PolylineNode polylineNode = new PolylineNode(new Polyline(), new
PropertyMapping());

        // implement a selection corner dragged listener
        // the event occurs before the boundary is actually changed.
        class PolylineCornerDragListener implements CornerDragListener {
            /**
             * @param e - the SelectionBoundEvent
             */
            public void cornerDragged(SelectionBoundEvent e) {
                // change the vertex of polyline according to new bound
                // update the visual looking of polyline
                ((PolylineNode) e.getSource()).notifyUpdate();
            }
        }

        // add selection corner listener to actually change the polyline boundary
        polylineNode.addCornerDragListener(new PolylineCornerDragListener());

        // implement boundary change listener
        // the event occurs after the boundary is actually changed.
        class PolyLineBoundaryListener implements BoundaryChangeListener {
            /**
             * @param e - the BoundaryChangeEvent
             */
            public void boundaryChanged(BoundaryChangeEvent e) {
                // changed the location of the nearby nodes
                // change the size of diagram viewer to contain it.
            }
        }

        // add the boundary changed listener
        polylineNode.addBoundaryChangeListener(new PolyLineBoundaryListener());
```

Note, following demo tells how to use CommentNode and CommentLinkEdge.

```
        // load configuration, and create property mappings
        TestHelper.clearConfig();
        TestHelper.loadXMLConfig(TestHelper.CONFIG_FILE);
        PropertyMapping propMapping = new PropertyMapping(VALID_NAMESPACE);

        // create graphNode for CommentNode
        GraphNode graphNode = new GraphNode();
        Uml1SemanticModelBridge usmb = new Uml1SemanticModelBridge();
        usmb.setElement((Element) new CommentImpl());
        graphNode.setSemanticModel(usmb);
        graphNode.addContained(new TextElement());
        graphNode.setPosition(new com.topcoder.diagraminterchange.Point());
        graphNode.setSize(new com.topcoder.diagraminterchange.Dimension());

        // create comment node with property mapping
        CommentNode commentNode = new CommentNode(graphNode, propMapping);

        // create a class to listen to text change event.
        class CommentChangeListener implements TextChangeListener {
            /**
             * @param e - the TextChangeEvent
             */
            public void textChanged(TextChangeEvent e) {
                // retrieve the CommentNode, GraphNode, Comment element
                CommentNode node = (CommentNode) e.getSource();
                GraphNode graphNode = node.getGraphNode();
                Comment comment =
                    (Comment) ((Uml1SemanticModelBridge)
graphNode.getSemanticModel()).getElement();

                // actually change the text
                comment.setBody(e.getNewText());

                // get preferred size
                Dimension newSize = node.getPreferredSize(e.getNewText());
                com.topcoder.diagraminterchange.Dimension newSize2 =
                    new com.topcoder.diagraminterchange.Dimension();
                newSize2.setWidth(newSize.width);
                newSize2.setHeight(newSize.height);

                // change size of graphNode according to new text
                graphNode.setSize(newSize2);

                // notify the size of graphNode is changed.
                // the comment node will be updated accordinly.
                node.notifyUpdate();
            }
```

```
        }

        // register the text change listener
        // the text can be changed, and node will be resized automatically
        commentNode.addTextChangeListener(new CommentChangeListener());

        // set up the graphEdge instance
        GraphEdge graphEdge = new GraphEdge();
        graphEdge.addWaypoint(TestHelper.createDiagramInterchangePoint(100, 100));
        graphEdge.addWaypoint(TestHelper.createDiagramInterchangePoint(200, 200));
        graphEdge.addWaypoint(TestHelper.createDiagramInterchangePoint(300, 400));

        // create a comment link edge.
        CommentLinkEdge commentLink = new CommentLinkEdge(graphEdge);

        // create a class used to change the location of edge ending
        class CommentLinkEdgeEndingDragListener implements WayPointListener {
            /**
             * @param e - the WayPointEvent
             */
            public void wayPointDragged(WayPointEvent e) {
                // check whether the ending is dragged....

                // retrieve the edge
                CommentLinkEdge edge = (CommentLinkEdge) e.getSource();

                // retrieve the CommentNode link to this edge.
                //CommentNode commentNode = edge.getLeftConnector();

                // get connector link to this edge.
                Connector connector;
                if (edge.getLeftEnding().getEndingPoint().equals(e.getOldPosition()))
{
                    connector = edge.getLeftConnector();
                } else {
                    connector = edge.getRightConnector();
                }

                // get the actual connection point
                Point p = connector.getConnectionPoint(e.getNewPosition());

                // update the ending in diagram interchage.
                com.topcoder.diagraminterchange.Point point = new
com.topcoder.diagraminterchange.Point();
                point.setX(p.getX());
                point.setY(p.getY());
                edge.getGraphEdge().addWaypoint(point);
            }
        }
```

```
        // add the ending change listener.
        // then it will be linked to the comment node appropriately.
            commentLink.addWayPointDragListener(new
        CommentLinkEdgeEndingDragListener());
```

Note, following demo tells how to use CommentNode and CommentLinkEdge.

```
        // load configuration, and create property mappings
        TestHelper.clearConfig();
        TestHelper.loadXMLConfig(TestHelper.CONFIG_FILE);
        PropertyMapping propMapping = new PropertyMapping(VALID_NAMESPACE);

        // create graphNode for FreeTextNode
        GraphNode graphNode = new GraphNode();
        SimpleSemanticModelElement ssme = new SimpleSemanticModelElement();
        ssme = new SimpleSemanticModelElement();
        ssme.setTypeInfo(FREE_TEXT);
        graphNode.setSemanticModel(ssme);
        graphNode.addContained(new TextElement());
        graphNode.setPosition(new com.topcoder.diagraminterchange.Point());
        graphNode.setSize(new com.topcoder.diagraminterchange.Dimension());

        // create freeText node with property mapping
        FreeTextNode freeTextNode = new FreeTextNode(graphNode, propMapping);

        // create a class to listen to text change event.
        class FreeTextChangeListener implements TextChangeListener {
            /**
             * @param e - the TextChangeEvent
             */
            public void textChanged(TextChangeEvent e) {
                // retrieve the FreeTextNode, GraphNode, TextElement
                FreeTextNode node = (FreeTextNode) e.getSource();
                GraphNode graphNode = node.getGraphNode();
                TextElement textElement = (TextElement)
graphNode.getContaineds().get(0);

                // actually change the text
                textElement.setText(e.getNewText());

                // get preferred size
                Dimension newSize = node.getPreferredSize(e.getNewText());
                com.topcoder.diagraminterchange.Dimension newSize2 =
                    new com.topcoder.diagraminterchange.Dimension();
                newSize2.setWidth(newSize.width);
                newSize2.setHeight(newSize.height);
```

```
            // change size of graphNode according to new text
            graphNode.setSize(newSize2);

            // notify the size of graphNode is changed.
            // the freeText node will be updated accordinly.
            node.notifyUpdate();
        }
    }

    // register the text change listener
    // the text can be changed, and node will be resized automatically
    freeTextNode.addTextChangeListener(new FreeTextChangeListener());
```

5. Future Enhancements

Provide more beautiful looking style.