# Diagram Edges 1.0 Requirements Specification

## 1. Scope

### 1.1 Overview

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.

### 1.2 Logic Requirements

#### *1.2.1 Edge graphic component*

The Edge is the base JComponent class for the graph edges that can be added to a diagram view from Diagram Viewer component. An Edge corresponds to a GraphEdge from Diagram Interchange component (a method to access the graph edge should be provided).

The Edge's size and location attributes inherited from JComponent will represent the graphical bounds of the Edge relative to the diagram view. The location will always have positive values. The size will be bigger than the actual edge displayed, as the selection corners have to belong to the edge and in order to be able to paint them the Edge needs to allocate the extra margin space.

The location of the Edge needs to be positive because the JComponent can draw only on areas located on the positive side.

However, the relative position of the visual edges' waypoints shown to the user must correspond exactly to the relative positions in the graph node structure.

The designer can choose the appropriate way to represent the edge: either as a component with the bounds set so the elements of the edge can be contained, or as a component big as the diagram.

Note that the edge contains the text fields also, which can be moved far away from the edge. Therefore, this aspect needs to be considered.

#### *1.2.2 Accepting events*

The Edge will accept events only along the actual line, with a configurable width along the line (as one pixel is not enough, making it hard to be clicked by the user).

The edge will also receive events in the edge ends and the attached text fields, but these might be handled implicitly by the actual elements.

#### *1.2.3 Edge line*

The Edge is made of segments connecting the edge waypoints. The line can be continuous or dashed. The dashes should be configurable (the length of the dash and the length of the blank space between the dashes).

#### *1.2.4 Edge ends*

The Edge may have different looking edge ends. They can also have different sizes. The edge ends should be drawn on the same direction as the line, on top of the line.

### *1.2.5 Selection corners*

The Edge can have selection corners, which are activated when the Edge is selected (mouse pressed on them). The Edge will react to the mouse event by setting itself in the list of selected elements in the DiagramViewer. If Ctrl is pressed when the mouse event occurs, the edge will add itself, without removing the other selected elements. To test whether the Edge is selected or not, when the Edge draws itself, the Edge will check the list of selected elements from the DiagramViewer.

The selection corners will be round circles drawn on top of each corner (waypoint), also at the end of edge ends. Each corner has the ability to move the Edge waypoint, if the user drags it. The line will merge two segments into one if the segments are on the same direction (with a configurable deviation).

The drag event should also generate an event and the application should be able to register listeners for it (the event affects the graphical model; the event could be validated by the application - the edge ends, for instance may not be allowed to be moved, as the position for these will be determined automatically...; the event could be changed - snapped to grid ...).

The waypoints should not be set to the new waypoints. There will be a listener that will set the waypoints after the event is processed.

The drawing of the corners should be kept in a different method, in order to be used by the concrete classes.

The size, the fill color and the stroke-color of the corners should be configurable. The shape of the corners could also be configurable, but it is not required.

### *1.2.6 Dragging*

The edges should react to mouse dragging events. Similar to the selection corners, the dragging should not be performed directly. The event should be triggered, so the listeners can take the proper actions. Eventually, the listener will split the segment in two segments and add the new waypoint.

There could be a different behavior - working with right angle segments. This is not required. It isn't even required to leave the possibility open. But it should be considered, and it is a big enhancement only to allow an easy future development for this.

### *1.2.7 Different state for events*

The Edge should check the flag of the DiagramViewer to see if the diagram is not in the state for adding an element. If that is the case, the edges should have a validator that tells whether they can consume the mouse events or not. Note that no event will be processed as in the normal state.

Given the fact that the validation might be quite different (depending on the actual elements), all that is required here for the Edge is to provide the means for the concrete subclasses to:

- either consume the event (ignoring or adding the new element),
- or to pass the event to the component immediately behind it (considering the diagram view hierarchy, not the graph node container hierarchy).

### *1.2.8 Text fields*

There will be text fields anchored to the edges and to the edge ends. They will be represented only with text and no background. When selected, there should be a line drawn that shows the connection of the text to the anchorage point on the edge.

The text fields can be dragged. The dragging will be built in this component. The dragging will trigger an event, however. The dragging will occur on the whole surface of the text field.

  

The text anchored on the line will maintain the same distance to the segment that is the nearest to it. It will maintain the relative position to the ends of the segment.

The text anchored on the edge end will maintain the same distance to the edge end.

There should be a way to set and get the text from the text fields.

There can be multiple text fields anchored to an edge end, or to the line. There should be at least two default positions, in case there are multiple text fields attached.

There should be an option to show or hide the text fields when not selected. When selected, the text fields should be visible.

The text field should be selectable. There should be a visual indicator that the text field is selected - a background color, but not opaque.

When a text field is selected, an event should be triggered.

#### *1.2.9 Connectors to Nodes*

The edge ends are computed according to the shape of the nodes, the position of the waypoints closer to the edge ends and according to a certain behavior desired by the user.

This component should connect the edge to the closest point of the node. However, some nodes are ovals (use cases), some are accepting lines only in some areas (sequence diagrams lifelines), and so this component will define an interface that computes the connector point (the one where the edge connects to the node). The interface will receive the nearest waypoint of the edge and will return the connector waypoint.

There will be an implementation for each custom shaped element.

A default connector will be provided for rectangle shaped elements.

### 1.3 Required Algorithms

None.

### 1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool as the base classes for the diagram viewer's edges.

### 1.5 Future Component Direction

None.

## 2. Interface Requirements

#### *2.1.1 Graphical User Interface Requirements*

None.

#### *2.1.2 External Interfaces*

None.

#### *2.1.3 Environment Requirements*

- Development language: Java 1.5
- Compile target: Java 1.5

#### *2.1.4 Package Structure*

com.topcoder.gui.diagramviewer.edges

  

# 3. Software Requirements

## 3.1 Administration Requirements

*3.1.1 What elements of the application need to be configurable?*

None.

## 3.2 Technical Constraints

*3.2.1 Are there particular frameworks or standards that are required?*

None.

*3.2.2 TopCoder Software Component Dependencies:*

- Diagram Viewer
- Diagram Interchange

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

*3.2.3 Third Party Component, Library, or Product Dependencies:*

None

*3.2.4 QA Environment:*

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

## 3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

## 3.4 Required Documentation

*3.4.1 Design Documentation*

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

*3.4.2 Help / User Documentation*

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.