[TOPCODER]
SOFTWARE

Diagram Elements 1.0 Requirements Specification

1. Scope

1.1 Overview

The Diagram Elements component provides a general framework for representing graphically the Diagram Interchange graph nodes and edges 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. 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

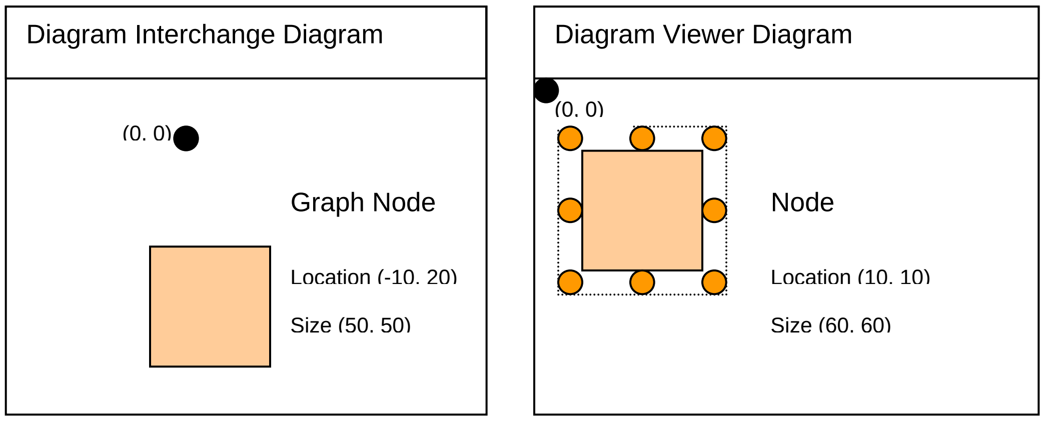
1.2.1 Node graphic component

The Node is the base JComponent class for the graph nodes that can be added to a diagram view from Diagram Viewer component. A Node corresponds to a GraphNode from Diagram Interchange component (a method to access the graph node should be provided). The GraphNode can have contained GraphNodes, which are not necessarily represented as Nodes. These could be compartments (like a name compartment, a body compartment ...). The container graph nodes are graph nodes, which can accept other graph nodes directly on them or on a contained graph node compartment (but this will be detailed below).

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

The size and location from the GraphNode object that the Node represents will be different things. For instance, the graph node location can have negative values. The size will be the exact size of the node.

Here is a visual example:



Since the diagram has only this node, and since the diagram view has a margin of 10 pixels (we assume), the node location on the diagram view will be at (10, 10). The Node, when selected will need a margin of 5 pixels to draw the 'selection' corners. When it is not selected, the corners will not be visible, and the actual position of the visible node will be at (15, 15) - this is what the user sees. The size of the Node will be (60, 60), in order to be able to properly show the actual node and the 'selection' corners. The size of the drawn Node will be equal with the size from the GraphNode: (50, 50).

Note that the concrete graphical nodes might require extra space, besides the actual size of the graph node for other reasons, not just the 'selection' corners. This is just an example, to explain why the size and location of the Node are different from the size and location of the GraphNode.

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

However, the relative position of the visual nodes shown to the user must correspond exactly to the relative positions in the graph node structure. The sizes also have to match - the size of the visual nodes shown to the user must be equal to the size of the graph nodes.

1.2.2 Selection corners

The Node will have selection corners, which are activated when the Node is selected (mouse pressed on them). The Node 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 node will add itself, without removing the other selected nodes. To test whether the Node is selected or not, when the Node draws itself, the Node will check the list of selected elements from the DiagramViewer.

The selection corners will be round circles drawn on top of each corner and at the middle of each side. Each corner has the ability to resize the Node, if it is dragged by the user. The mouse cursor should be changed, to reflect the resize action that can occur. The behavior is the normal one used when resizing an element or window in other applications.

The resize 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 should be validated by the application - minimum size...; the event could be changed - snapped to grid ...).

The node's size should not be set to the new size. There will be a listener that will set the size 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. This comment is just so you do not combine a default drawing with the corner drawing, since the concrete nodes might look very different: circles, UML packages, UML actors...

The selection corners should be drawn around the visual node. The node should provide a method to set the relative position of the visual node according to the (0, 0) point of the JComponent Node. This is required in order to allow other tool buttons... to be added by the concrete nodes.

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.

1.2.3 NodeContainer

Some nodes might contain other nodes. The concrete nodes that can contain other nodes should extend this class. The contained nodes can be contained directly, or in some contained compartments. The compartments' positions are exactly as specified in the GraphNode (the relative position to the actual Node), so the actual position of the contained nodes will be computed by translating the position of the Node with the position of the compartments.

For instance, the Package will accept nodes in the BodyCompartment, which could be located at (0, 20). The contained GraphNode has a location of (10, 10). This means the contained Node will have a relative position to the container Node of (10, 30).

The contained Nodes will not be added to the container Node using the built-in JComponent's container hierarchy. Both the container nodes and the contained nodes belong to the Diagram view. The containment hierarchy is just simulated by setting the proper order of the components in the Diagram view.

Therefore, the actual location of the contained node will be the sum between the location of the container node and the relative position to this one.

The node should provide methods to access the contained nodes: all the contained nodes, and the nodes in a certain compartment.

1.2.4 Dragging

The nodes 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 perform the dragging.

1.2.5 Different state for events

The Node 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 nodes 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 Node 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 node immediately behind it (considering the diagram view hierarchy, not the graph node container hierarchy).

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 elements.

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.elements

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 1.0
- Diagram Edges 1.0
- Diagram Interchange 1.0

**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.