[TOPCODER]
SOFTWARE

Diagram UML Activity Elements 1.0 Requirements Specification

1. Scope

1.1 Overview

The Diagram UML Activity Elements component provides the graphical diagram elements and edges representing the model elements specific to an activity diagram.

1.2 Logic Requirements

1.2.1 InitialNode Node

This class is a concrete Node. It takes its information from the Pseudostate class with the kind equal to Pseudostate.INITIAL from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The stereotype compartment could be hidden. The 'isVisible' attribute of the compartment graph node is for this property. There should be methods to tell the interface that the compartment's visibility has changed. The initial node will be resized and an event will be triggered, to signal the size change (the reason is also passed, as a string). The node should also have a method that computes the preferred size, according to the new visible/hidden compartment.

The name compartment will not be visible if there is no name.

The stereotype compartment do not react to any events. They let the events pass to the component, which reacts as described below.

The whole component will react to a mouse double-click event, by showing the edit control of the DiagramViewer in order to edit the name of the node. The text of the Name compartment will not be shown while the edit control is up (though the node will not be resized). The event should provide to the DiagramViewer the position where to show the edit control, so it fits on top of the Name compartment, and the initial text. It will also register a listener to receive the event that the text was entered or cancelled in the edit control. It will remove the listener after receiving the event.

The new name will not be set. An event is generated instead, with the old and new name, and with the node and graph node for which the name is set. The node will also provide a method to check the size that will be required for the node if the name would be set (this is required, as other representations of the same model element could be resized as a side effect). The application will register for the event and, eventually it will set the new name. The component should make it easy for the application to set the new name, by performing the resize needed when the name is changed. The method for setting the name will perform the resize of the name, and it will generate a resize event (the reason is also passed, as a string).

The graphical component will not be implemented as a drag and drop DropTarget, as it is not a container. However, it should not interfere with the drag and drop action initiated by the user. The user should be able to drop the element on top of the node and the event should be handled by the diagram behind it (as the intention of the user is to add the element to the diagram).

The node should define a minimum size. It should also have a method that computes the preferred size, according to the name compartment and the stereotype compartment. It should also provide a method to compute the preferred size if the name or the stereotypes change.

The node will show only the four selection corners, when it is selected (not the ones at the middle of the edges). The corners will be shown around the round shape.

The round shape is the base of the component. It should remain at the same place if the stereotype compartment is shown or hidden.

In case the diagram viewer's flag for adding new elements from the toolbar is on, the node should react to mouse events differently, by letting the events pass to the element behind.

There will be a connector for the edges defined for the node. It will behave differently than the connector for the rectangle shaped elements. The last segment of the edge should point to the closest point on the round shape. The name compartment and the stereotype compartment should be taken into consideration, when they are active.

There should be a way to set a popup that will be shown if a popup trigger mouse event action occurs. There will be a popup general for the component.

The node should have several properties configurable through graph node:

- the stroke color (defaults to black)
- the fill color (defaults to black)
- the font color (defaults to black)
- the font family (defaults to Arial)
- the font size (defaults to 10).

The component should receive events only in the round shape and in the text compartments.

1.2.2 ObjectFlowNode Node

This class is a concrete Node. It takes its information from the ObjectFlowNode from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above, with a few differences.

The stereotype compartment and the name compartment are contained in the shape.

All the selection corners are shown.

The connector for the edges is the default connector for the rectangle shaped elements.

1.2.3 ActionState Node

This class is a concrete Node. It takes its information from the ActionState from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (ObjectFlowNode), with a few differences, related to the shape.

1.2.4 SendSignalAction Node

This class is a concrete Node. It takes its information from the SimpleAction with a tag definition attached (TagDefinition("SendSignalAction").value="True") from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (ObjectFlowNode), with a few differences, related to the shape.

1.2.5 AcceptEventAction Node

This class is a concrete Node. It takes its information from the SimpleAction with a tag definition attached (TagDefinition("AcceptEventAction").value="True") from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (ObjectFlowNode), with a few differences, related to the shape.

1.2.6 ForkNode Node

This class is a concrete Node. It takes its information from the Pseudostate with the kind equal to Pseudostate.FORK from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (InitialNode), with a few differences, related to the shape.

In addition, the node has only the selection corners from the middle of the vertical edges (only the width can change).

The connector for the edges works a bit differently. The incoming transitions go directly to the point at the top. The outgoing transitions go from the closest point of the two outer points at the bottom.

1.2.7 JoinNode Node

This class is a concrete Node. It takes its information from the Pseudostate with the kind equal to Pseudostate.JOIN from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (InitialNode), with a few differences, related to the shape.

In addition, the node has only the selection corners from the middle of the vertical edges (only the width can change).

The connector for the edges works a bit differently. The incoming transitions go to the closest point of the two outer points at the top. The outgoing transitions go directly from the point at the bottom.

1.2.8 DecisionNode Node

This class is a concrete Node. It takes its information from the Pseudostate with the kind equal to Pseudostate.CHOICE from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (InitialNode), with a few differences, related to the shape.

1.2.9 MergeNode Node

This class is a concrete Node. It takes its information from the Pseudostate with the kind equal to Pseudostate.JUNCTION from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (InitialNode), with a few differences, related to the shape.

1.2.10 FlowFinalNode Node

This class is a concrete Node. It takes its information from the FinalState with a tag definition attached (TagDefinition(“FinalNodeType”).value=”FlowFinalNode”) from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (InitialNode), with a few differences, related to the shape.

1.2.11 FinalNode Node

This class is a concrete Node. It takes its information from the FinalState from the UML Model and from the GraphNode associated with it.

The structure of the GraphNode that corresponds to it is given in section 3.2.1.

The node is similar to the one above (InitialNode), with a few differences, related to the shape.

1.2.12 Transition Edge

This class is a concrete Edge. It takes its information from the Transition class from the UML Model and from the GraphEdge associated with it.

The structure of the GraphEdge that corresponds to a Transition is given in section 3.2.1.

The stereotype compartment could be hidden. The 'isVisible' attribute of the compartment graph node is for this property. There should be methods to tell the edge that the compartment's visibility has changed.

The transition has a continuous line, with the exception of when it is connected to an ObjectFlowNode, in which case it is a dashed line.

There should be two Transition end types defined:

- target end: none
    - o ----
- source end: a simple arrow, with the color of the edge
    - o ---->

The edge supports the name and the stereotypes as text fields attached to the edge.

Selecting the text fields of the edge will result in selecting the edge.

Double clicking on a text fields will bring the diagram's viewer editing text field in front, with the text that should be edited. The name and the guard name are used for this in this. The format could be "name [guardName]". Each one of those two names can be missing.

There should be a way to set popups that will be shown if a popup trigger mouse event action occurs. There will be a popup general for the component.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool to display the class diagram related elements on a diagram in the diagram viewer.

1.5 Future Component Direction

None.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The design must follow the interface found in the class diagram with the component interfaces.

The designer is encouraged to add to the existing interface, but not to remove anything.

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.gui.diagramviewer.uml.activityelements

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?

The structure of the Diagram Interchange elements should be respected. The names of the compartments are provided in this file.

UML Tool - Activity Diagram Elements Compartments.rtf

3.2.2 TopCoder Software Component Dependencies:

- Diagram Viewer 1.0
- Diagram Elements 1.0
- Diagram Edges 1.0
- UML Model Manager 1.0
- UML Model components
- Diagram Interchange 1.0
- Configuration Manager 2.1.5 - recommended

**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.