Diagram UML Activity Elements 1.0 Component Specification

1. Design

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

This design depends on several other components that are still under design, so it may be updated and modified in the final fix phase.

There are 11 nodes and 1 edges in this components, the nodes are similar in most behavior and only different in something like shape, so two abstract super class are abstracted to provide the common logic and let the individual nodes do the different details.

1.1 Design Patterns

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

Strategy pattern — The define of the concrete node classes can be regard as strategies.

1.2 Industry Standards

JFC/Swing

UML 2.0 Diagram Interchange

1.3 Required Algorithms

This component has no complex algorithm, only some straightforward initializations related, and the way to render the node and edge should be chose with some consideration.

1.3.1 Build the structure of the GraphNode for InitialNode, ForkNode, JoinNode, DecisionNode, MergeNode, FlowFinalNode and FinalNode, the structure should be in this form:

```
Uml1SemanticModelBridge.element = <UML:Pseudostate>
        SimpleSemanticModelElement.typeInfo = "StereotypeCompartment"
        SimpleSemanticModelElement.typeInfo = "Name"
```

The structure of StereotypeCompartment should be in this form:

```
SimpleSemanticModelElement.typeInfo = " StereotypeCompartment"
        [SimpleSemanticModelElement.typeInfo = “KeywordMetaclass" - For interface...]
        SimpleSemanticModelElement.typeInfo = “StereotypeStart"
        Uml1SemanticModelBridge.element = <UML:Stereotype>
                SimpleSemanticModelElement.typeInfo = "Name"
        SimpleSemanticModelElement.typeInfo = "StereotypeSeparator"
        Uml1SemanticModelBridge.element = <UML:Stereotype>
                SimpleSemanticModelElement.typeInfo = "Name"
        Repeat...
        SimpleSemanticModelElement.typeInfo = "StereotypeEnd"
```

The building process should be as follow steps:

1. Create instances of GraphNode as the structure described, and make the relationships by calling method Node#addContained()

2. Create instances of SemanticModelBridge, they could be either instances of Uml1SemanticModelBridge or instances of SimpleSemanticModelElement, then set them to the corresponding GraphNode by calling method Node#setElement().

1.3.2 Build the structure of the GraphNode for ObjectFlowNode, ActionState, SendSignalAction and AcceptEventAction, the structure should be in this form:

```
Uml1SemanticModelBridge.element = <UML:ObjectFlowNode>
    SimpleSemanticModelElement.typeInfo = "NameCompartment"
        SimpleSemanticModelElement.typeInfo = "NameAndType"
            SimpleSemanticModelElement.typeInfo = "Name"
            SimpleSemanticModelElement.typeInfo = "TypeSeparator"
                Uml1SemanticModelBridge.element = <UML:Class> (a Classifier)
                    SimpleSemanticModelElement.typeInfo = "Name"
```

The process should be similar with 1.3.1.

1.3.3 Build the structure of the GraphEdge for Transition, the structure should be in the form:

```
Uml1SemanticModelBridge.element = <UML:Transition>
    SimpleSemanticModelElement.typeInfo = "Name"
    SimpleSemanticModelElement.typeInfo = "TransitionDescription"
        SimpleSemanticModelElement.typeInfo = "GuardStart"
        Uml1SemanticModelBridge.element = <UML:Guard>
            SimpleSemanticModelElement.typeInfo = "Name"
        or SimpleSemanticModelElement.typeInfo = "NullGuard"
        SimpleSemanticModelElement.typeInfo = "GuardEnd"
    SimpleSemanticModelElement.typeInfo = "StereotypeCompartment"
```

The process should be similar with 1.3.1.

The <UML:Transition> should be an instance of com.topcoder.uml.model.statemachines.Transition.

1.3.4 Render the nodes and edges

Most of the nodes and edges in this component have a irregular shape, so there are two method to do the rendering, one is define an array that contains each point's graphics information, another is using a image such as a GIF file to render the shape. The developer can choose any method to implement the rendering process.

1.4 Component Class Overview

BaseNode (abstract class):

This class is the base Node of this component, it defines the common behaviors of all nodes in this component.

It contains four properties: fill color, stroke color, font color, and font. (JComponent#setFont and getFont methods are reused to support the font property.).

It implements the MouseListener to react to a mouse double-click event, by showing the edit control of the DiagramViewer in order to edit the name of the node.

This class is mutable, not thread safe.

InitialNode:

This class represents the initial node in the activity diagram, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

ObjectFlowNode:

This class is a concrete Node. It takes its information from the ObjectFlowNode from the UML Model and from the GraphNode associated with it, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

ActionState:

This class is a concrete Node. It takes its information from the ActionState from the UML Model and from the GraphNode associated with it, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

SendSignalAction:

This class is a concrete Node. It takes its information from the SimpleState with a tag definition attached (TagDefinition("SendSignalAction").value="True") from the UML Model and from the GraphNode associated with it, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

AcceptEventAction:

This class is a concrete Node. It takes its information from the SimpleState with a tag definition attached (TagDefinition("AcceptEventAction").value="True") from the UML Model and from the GraphNode associated with it, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

ForkNode:

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, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

JoinNode:

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, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

DecisionNode:

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, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

MergeNode:

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, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

FlowFinalNode:

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, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

FinalNode:

This class is a concrete Node. It takes its information from the FinalState from the UML Model and from the GraphNode associated with it, it extends the BaseNode to share the common initialization and operation, but overrides the methods to provide its own rendering and structure.

This class is mutable, not thread safe.

Transition:

This class is a concrete Edge, it will be shown as a line (continuous in general with the exception that when it is connected to an ObjectFlowNode is is a dashed line).

It contains three TextField fields to represent name, description and stereotype compartment.

This class is mutable, not thread safe.

TextField:

This class represents pure text compartment of Node or Edge.

It could be used to represent name compartment, stereotype compartment and etc.

Text field would be displayed as pure text with font and color inherited form parent node or edge.

There is no decorator around or on the text.

This class is mutable, not thread safe.

EllipseConnector:

This clas is the connector used to connect to a Node whose shape is ellipse.

This class is mutable, not thread safe.

RectangleConnector:

This clas is the connector used to connect to a Node whose shape is rectangle.

This class is mutable, not thread safe.

[event sub-package]:

TextChangedListener (interface):

This interface provides a mechanism to let the application be notified when the text compartment (instance of TextField) of the nodes is changed, there is only one method in this class and it will be called to process the name changed event, the implementation should change the TextField's text to the new one by calling the method setText().

The implementation is not required to be thread safe.

TextChangedEvent:

Encapsulates the related information when changing a Node's compartment, the Node property can be retrieved by getSource(), and the GraphNode can be retrieved from the Node.

This class is immutable so is thread safe.

1.5 Component Exception Definitionse.

IllegalGraphElementException:

This exception is used to indicate some GraphNode or GraphEdge is illegal in specific situation. For example, a ActionState GraphNode is given to InitialNode constructor. It could be thrown when retrieving graph information from GraphNode or GraphEdge.

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.

1.6 Thread Safety

This component is not thread safe since most of the classes in this component are mutable except the event classes. So thread-safety should be cared by users just like other standard swing components.

And there is one issue should be noticed, 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. The listeners is added to the list, so thread safety is not a problem here.

However, if this component is used in a multi-threaded environment, e.g., there are several worker threads to complete some kind of task; it will not cause any problems as long as the threads have no access to the state of this component, or all the request is posted via the event-dispatcher. Please refer to the javadoc of SwingUtilities class for how to do this.

If we really want to achieve thread safety of this component, despite this strongly opposed, we only need to make synchronization around the API that has access to the states to achieve thread safety.

2. Environment Requirements

2.1 Environment

- At minimum, Java1.5 is required for compilation and executing test cases.
- Java 1.5 or higher.

2.2 TopCoder Software Components

- Base Exception 1.0 was used to provide a base exception for custom exceptions.
- UML Model Core 1.0 was used to provide UML Model element.
- UML Model State Machines 1.0 was used to provide UML Model element.
- UML Model Activity Graphs 1.0 was used to provide UML Model element.
- UML Model Core Relationships1.0 was used to provide UML Model element.
- UML Model Data Types 1.0 was used to provide UML Model element.
- UML Model Common Behavior 1.0 was used to provide UML Model element.
- UML Model Core Extension Mechanisms 1.0 was used to provide UML Model element.
- Diagram Elements 1.0 was used to provide class Node and NodeContainer.
- Diagram Edges 1.0 was used to provide class Edge.
- UML Model Core Classifier 1.0 was used to provide UML Model element.
- UML Diagram Interchange 1.0 was used to provide UML diagram interchange data structure.

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

- JUnit : 3.8.2 : http://www.junit.org

- JFCUnit

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.gui.diagramviewer.uml.activityelements

3.2 Configuration Parameters

No configuration.

3.3 Dependencies Configuration

None needed.

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

Nothing special.

4.3 Demo

1. Create custom properties mappings:

```
Map<String, String> properties = new HashMap<String, String>();
properties.put("FillColor", "fill_color");
properties.put("StrokeColor", "stroke_color");
properties.put("FontColor", "font_color");
properties.put("FontFamily", "font_family");
properties.put("FontStyle", "font_style");
properties.put("FontSize", "font_size");
```

2. Create new nodes:

```
node = new InitialNode(graphNode, properties, new Point(), new Rectangle());
```

3. Register listeners:

```
// After creating the node, register a listener to receive text changed event:
InitialNode node = ...;
TextChangedListener listener = new TextChangedListener() {
    public void textChanged(TextChangedEvent event) {
        TextField compartment = (TextField) event.getSource();
        compartment.setText(event.getNewValue());
        }
};
// register the listener
node.getNameCompartment().addTextChangedListener(listener);

4. Show the node:
//Create a Diagram Viewer to contain this node
node.setVisible(true);
DiagramViewer viewer = new DiagramViewer(…);
viewer.add(node);
JDialog dialog = new JDialog((JFrame) null, "Demo", true);
dialog.setSize(500, 500);
dialog.getContentPane().add(viewer);
dialog.setVisible(true);
```

5. Future Enhancements

Provides more user friendly UI rendering.