Diagram UML Use Case Elements 1.0 Component Specification

1. Design

The Diagram UML Use Case Elements component provides the graphical diagram elements and edges representing the model elements specific to a use case diagram. SubsystemNodeContainer is a concrete NodeContainer, which takes information from Subsystem class from UML Model and from the GraphNode associated. ActorNode and UseCaseNode are concrete Node classes, which represent Actor and UseCase in UML Model respectively. IncludeEdge and ExtendEdge are concrete Edge classes, which represent Include and Extend in UML Model.

ActorNode and UseCaseNode are very similar, except the shape and position of compartments. So a common class named as BaseNode is defined. This class defines common properties like stroke color, fill color, and etc. BaseNode also contains name compartment, stereotype compartment and namespace compartment, but the location and value of these compartments are determined by concrete classes. Node classes can fire boundary changed event.

SubsystemNodeContainer is very like BaseNode, but since it should extend from NodeContainer, it can’t inherit attributes defined in BaseNode class. Besides similar functionality as BaseNode, SubsystemNodeContainer also support Drag-And-Drop, and it can fire edge adding or node adding events.

IncludeEdge and ExtendEdge are totally the same except their keyword meta-classes. BaseEdge is defined in this component with configurable keyword meta-class value. BaseEdge contains a name compartment and a stereotype compartment.

To allow popup menu, PopupMenuTrigger is defined. It is a mouse event listener, and it would show popup menu if some JComponent is right clicked. This listener is registered to all nodes and edges automatically to support popup menu.

To allow double-click-editing, TextBoxTrigger is defined. It is a mouse event listener. It will ask the diagram viewer to show edit box when double click event occurs. This listener should be registered automatically to the element which allows double-click-editing.

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 – ActorNode and UseCaseNode provide different implementations for abstract getPreferredGraphNodeSize method and notifyGraphNodeChange method from abstract BaseNode.

1.2 Industry Standards

JFC Swing

UML 2.0 Diagram Interchange

1.3 Required Algorithms

1.3.1 Compute the connection point to ActorNode.

The shape of an actor is not normal. To simplify the problem, we treat it as a rectangle. By the way, Poseidon also treats the shape of actor as a rectangle. The actual problem comes from name compartment and stereotype compartment. We can solve this problem level by level. Both name compartment and stereotype compartment also have rectangle shape. As a result, this problem becomes to find the closest point to three rectangles. To find closest point to a rectangle, we can consider a rectangle as four segments. Again, the problem becomes as easy as to find the closest point to a segment. 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

Here is the algorithm workflow:

```
shortestDistance := INF;
connectionPoint := (0, 0)
foreach segment in bounding rectangle of actor shape, name compartment and stereotype compartment.
  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
```

1.3.2 *Compute the connection point to UseCaseNode.*

Fortunately, both name compartment and stereotype compartment are contained in the UseCase node. We only need to find the closest point to an ellipse, which is the shape of UseCase node. To find the closest distance between a point and an ellipse is not an easy problem, and it is also very hard to implement. After searching in internet, we can find that the algorithm is neither easy nor efficient.

Here, we provide an approximate solution, which is very simple and also efficient. We draw a line between given point and the center of ellipse, and treat the intersection point of the line and ellipse as our desired point.

The most straightforward way works as follows. Developer is encouraged to use other algorithms, like using polar coordinates.

(1) Write the equation of ellipse and line as:

Line: $ax + by + c = 0$

Ellipse: $(x*x) / (a*a) + (y*y) / (b * b) = 1$

(2) Merge the two equations to get the result. It is a simple problem.

1.4 Component Class Overview

SubsystemNodeContainer:

This class represents a Subsystem element in UML Model.

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. Name compartment, stereotype compartment and namespace compartment are defined in this class. This class also registers EditBoxTrigger automatically to allow name editing.

BoundChangeEvent would be triggered by this node. It indicates the node's bound is changed. This class support Drag-And-Drop, and new node or edge adding event can be fired by this class.

This class is mutable, and not thread-safe.

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. Name compartment, stereotype compartment and namespace compartment are defined in this class. This class also registers EditBoxTrigger in constructor automatically to allow name editing.

BoundChangeEvent would be triggered by this node. It indicates the node's bound is changed.

This class is mutable, and not thread-safe.

ActorNode:

This class represents an Actor node is UML diagram. It is an extension of BaseNode, which defines the most features for a node. This class only provides some specific methods to support the unique shape of actor, and also to support the unique structure of actor GraphNode.

This class is not thread safe, because the BaseNode class is not thread safe.

UseCaseNode:

This class represents an UseCase node is UML diagram. It is an extension of BaseNode, which defines the most features for a node. This class only provides some specific methods to support the unique shape of usecase (an ellipse), and also to support the unique structure of usecase GraphNode.

This class is not thread safe, because the BaseNode class is not thread safe.

BaseEdge:

This class is the base Edge of this component. It is an extension of Edge class served for the purpose of providing all the common functionalities required by Include and Extend edges.

It contains three kinds of properties. They are name compartment, stereotype compartment, and keyword meta-class.

This class also adds EditBoxTrigger to name compartment automatically to allow name editing. PopupMenuTrigger is also registered automatically to support popup menu.

This class is mutable, and not thread-safe.

IncludeEdge:

This class represents an Include edge in UML diagram. It is an extension of BaseEdge, which defines all features required by an Include edge. The only specific configuration of Include edge is keyword meta-class. Keyword meta-class value for Include edge is defined as "include".

This class is not thread safe, because the BaseEdge class is not thread safe.

ExtendEdge:

This class represents an Extend edge in UML diagram. It is an extension of BaseEdge, which defines all features required by an Extend edge. The only specific configuration of Extend edge is keyword meta-class. Keyword meta-class value for Extend edge is defined as "extend".

This class is not thread safe, because the BaseEdge class is not thread safe.

SimpleArrow:

This class represents the simple arrow used in edges.

It can draw a simple arrow at specified location and angle, and it also can determine whether given point is in the arrow triangle.

This class is immutable, but the super class may not be thread-safe.

TextField:

Text field 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, and not thread-safe.

ActorConnector:

This class is the connector used to connect to ActorNode.

If name, stereotype and namespace compartments are visible, they will be taken in consideration. For simplicity (like Poseidon), the actor shape is treated as a rectangle.

This class is immutable, and thread-safe.

UseCaseConnector:

This class is the connector used to connect to UseCaseNode. It is relatively simple, as the shape of UseCase is an ellipse, and compartments are contained in ellipse, which we don't take consideration.

This class is immutable, and thread-safe.

[Event sub-package]
PopupMenuTrigger:

This is an 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 automatically.

This class is immutable, and thread-safe.

EditBoxTrigger:

This class can trigger edit box of diagram viewer when some component is double clicked. A TextField instance should be given to this class to tell which text field should be edited.

For example, this trigger can be registered to a UseCase node, and the text field representing name compartment will be associated with this trigger. As a result, when the use case node is clicked, the name compartment will be editable.

This class is immutable and thread-safe.

EditBoxListener:

This listener is used to listen to events from edit box in diagram viewer. It must be attached to a TextField. 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 expected be used internally. It will be created in EditBoxTrigger#mouseClicked method, and will be registered to the edit box automatically.

This class is immutable, and thread-safe.

BoundaryChangedEvent:

This is an event object used to indicate bound of node (including node container) is changed. It contains four properties. They are the Nodes 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.

BoundaryChangedListener<<interface>>:

This interface defines the contract that every boundary change event listener must follow. It contains only one method to process the boundary changed event with a single BoundaryChangeEvent parameter.

The implementations of this interface not required to be thread-safe. They could be used in a thread safe manner in this component.

TextChangeEvent:

This is an event object used to indicate text of text field is changed. It contains three properties. They are the TextField whose text is changed, the old text value, the new text value. Note, the TextField 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. This kind of listener can be registered to TextField instances.

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

For example, application can register a listener to a TextField, which represents name compartment.

The implementations of this interface not required to be thread-safe. They could be used in a thread safe manner in this component..

NodeAddEvent:

This is an event object used to indicate a new node will be added to SubsystemNodeContainer. It contains only contains one location property, which tells where the new element should be added.

SubsystemNodeContainer can be retrieved by getSource(), and added node type can be retrieved from diagram viewer.

This class is immutable, and thread-safe.

NodeAddListener<<interface>>:

This interface defines the contract that every node adds event listener must follow. Note, the event would be triggered before the node is actually added. This kind of listener can be registered to SubsystemNodeContainer instances.

It contains only one method to process the node add event with a single NodeAddEvent parameter.

The implementations of this interface not required to be thread-safe. They could be used in a thread safe manner in this component.

EdgeAddEvent:

This interface defines the contract that every node adds event listener must follow. Note, the event would be triggered before the node is actually added. This kind of listener can be registered to SubsystemNodeContainer instances.

It contains only one method to process the node add event with a single NodeAddEvent parameter.

The implementations of this interface not required to be thread-safe. They could be used in a thread safe manner in this component.

EdgeAddListener<<interface>>:

This interface defines the contract that every edge-add event listener must follow. Note, the event would be triggered before the edge is actually added. This kind of listener can be registered to SubsystemNodeContainer instances.

It contains only one method to process the edge-add event with a single EdgeAddEvent parameter.

The implementations of this interface not required to be thread-safe. They could be used in a thread safe manner in this component.

1.5 Component Exception Definitions

IllegalGraphElementException:

This exception is used to indicate some GraphNode or GraphEdge is illegal in specific situation. For example, a UseCase GraphNode is given to ActorNode 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.

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 cared 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 the list, so thread safety is not a problem here.

2. Environment Requirements

2.1 Environment

- Development language: Java1.5
- Compile target: Java1.5

2.2 TopCoder Software Components

- Diagram Viewer 1.0

It is used to hold the nodes and edges defined in usecase diagram.

- Diagram Interchange 1.0

It provides the standard UML diagram information, including size, position and etc.

- Diagram Elements 1.0

It defines the base classes for nodes, and node containers.

- Diagram Edges 1.0

It defines the base classes for edges.

- UML Model components 1.0

These components define all the UML Model elements, like UseCase, Actor, and etc.

- UML Model Manager 1.0

This component defines the Subsystem interface.

- Base Exception 1.0

IllegalGraphNodeException extends BaseException defined in the component.

Not used components from Requirement Specification

- Configuration Manager 2.1.5

In this component, only GraphNode property name/key mapping requires configuration. Such configuration is provided as a java.util.Map instance to the constructor of BaseNode, BaseEdge. The functionality to load configuration from file should be done in the application level, because all kinds of nodes/edges need such configuration in whole application.

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.usecaseelements

This package contains all the usecase diagram elements.

com.topcoder.gui.diagramviewer.uml.usecaseelements.event

This package contains all the listeners and events objects.

3.2 Configuration Parameters

None

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, this demo only shows part usage of this component, and the un-revealed part is much similar to following part.

```
// create custom properties mappings.
Map<String,String> properties = new HashMap<String,String>();
properties.put("StrokeColor", "stroke_color_property_key");
properties.put("FontSize", "font_size_property_key");
//...

// create a custom transfer handler class
public class CustomTransferHandler extends TransferHandler {
  // override 'importData' to provide custom transfering
  public boolean importData(JComponent comp, Transferable t) {
    // transter 't' to this component,
    // maybe add a new element to diagram
  }
}

// create Subsystem with properties mapping and transfer handler
SubsystemNodeContainer subsystem = new SubsystemNodeContainer(
  <<graphNode>>, properties, new CustomTransferHandler());

// create a class to listen to node add event.
public class SubsystemChildAddListener implements NodeAddListener {
  public void nodeAdd(NodeAddEvent e) {
    // query the diagram viewer which kind of node should be added.
    Node element = << new element>>

    // specify the location of the element
    element.setLocation(e.getLocation());

    // get the subsystem instance from event
    SubsystemNodeContainer subsystem = (SubsystemNodeContainer)
e.getSource();

    // add the node to this subsystem.
    subsystem.addNode("BodyCompartment", <<new element>>);
  }
}

// register a listener to the subsystem container to add child
subsystem.addNodeAddListener(new SubsystemChildAddListener());

// create a class to listen to text change event.
public class NameChangeListener implements TextChangeListener {

  public void textChange(TextChangeEvent e) {
    // retrieve the TextField, and subsystem node.
    TextField textField = e.getSource();
    SubsystemNodeContainer subsytem = textField.getParent();

    // get subsystem model element.
    GraphNode graphNode = subsystem.getGraphNode();
    Subsystem modelElement = graphNode.getModelElement();

    // actually change the name
    modelElement.setName(e.getNewText());

    // get preferred graphNode size
    Dimenstion newSize = node.getPreferredGraphNodeSize();

    // change size of graphNode according to new name
    graphNode.setSize(newSize);

    // notify the size of graphNode is changed.
    // the comment node will be updated accordingly.
    node.notifyGraphNodeChange();
  }
}

// register the text change listener to the name compartment.
// the name can be changed, and node will be resized automatically
subsystem.getNameCompartment().addTextChangeListener(
  new NameChangeListener());

// create an include edge.
IncludeEdge includeEdge = new IncludeEdge(<<graphEdge>>);

// create a class used to change the location of edge ending
public class IncludeEdgeEndingDragListener
   implements WayPointDragListener {
  public void wayPointDragged(WayPointDragEvent e) {
    // check whether the ending is dragged....

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

    // retrieve the UseCase node link to this edge.
    UseCaseNode usecaNode = <<...>>

    // get connector from comment node
    Connector connector = usecaseNode.getConnector();

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

    // update the ending in diagram interchage.
    edge.getGraphEdge().setEnding(p);
  }
}

// add the ending change listener.
includeEdge.addWayPointListener(new IncludeEdgeEndingDragListener());
```

5. Future Enhancements

Provide more beautiful looking style.