Diagram UML Class Edges 1.0 Component Specification

1. Design

The Diagram UML Class Edges component provides the graphical diagram edges representing the model relations specific to a class diagram.

The edges provided are the association, the generalization, the abstraction and the dependency. The edges are very similar, only the way the edge ends looks and the way the line is drawn is different. The edges extend from the base edge from the Diagram Edges component, providing the visual aspect and concrete graphical edge ends.

The rendering of the class elements is made by using Swing library.

1.1 Design considerations

The design of this component simply implements the concrete Edges for abstract ones defined in the Diagram Elements component. The main purpose of design – is to encapsulate rendering of the class edges. And as required – additional reactions for the mouse events were implemented.

AssociationEdge, GeneralizationEdge, AbstractionEdge and DependencyEdge are concrete Edge classes, which represent Association, Generalization, Abstraction and Dependency classes from UML Model respectively. They take information from associated GraphEdge object too.

AssociationEdge, GeneralizationEdge, AbstractionEdge and DependencyEdge are very similar, except the slight difference in endings. So a common class named as BaseEdge is defined. This class defines common name compartment and stereotype compartment, but the location and value of these compartments are determined by concrete classes.

The GeneralizationEdge, AbstractionEdge and DependencyEdge classes use simple edge endings: ClosedArrow or OpenArrow. These endings just draw their graphics, do not have any corresponding test compartments and do not react to mouse events.

But for AssociationEdge a special group of “active” endings was defined. The base class ActiveEdgeEnding contains name and multiplicity compartments, has some configuration properties, and allows selection and a special edge ending popup menu. The concrete active edge endings are: NoArrow, DirectedAssociation, Aggregation, AggregationBiDirectioinal, Composition, and CompositionBiDirectional. These endings can be only assigned to the AssociationEdge.

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 edges automatically to support popup menu. A similar PopupMenuEndingTrigger is defined and registered to the all active edge ending to allow another popup menu, specific for edge endings.

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. These elements are all text fields of edges and edge endings (except stereotypes).

The EdgeSelectionTrigger and EdgeEndingSelectionTrigger were implemented to support edge and active edge ending selection. These listeners should be registered to the all concrete edges and active edge endings.

1.2 Design Patterns

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

Template Method Pattern – the BaseEdge class defines abstract notifyGraphEdgeChanged method, which is differently implemented in concrete child classes.

1.3 Industry Standards

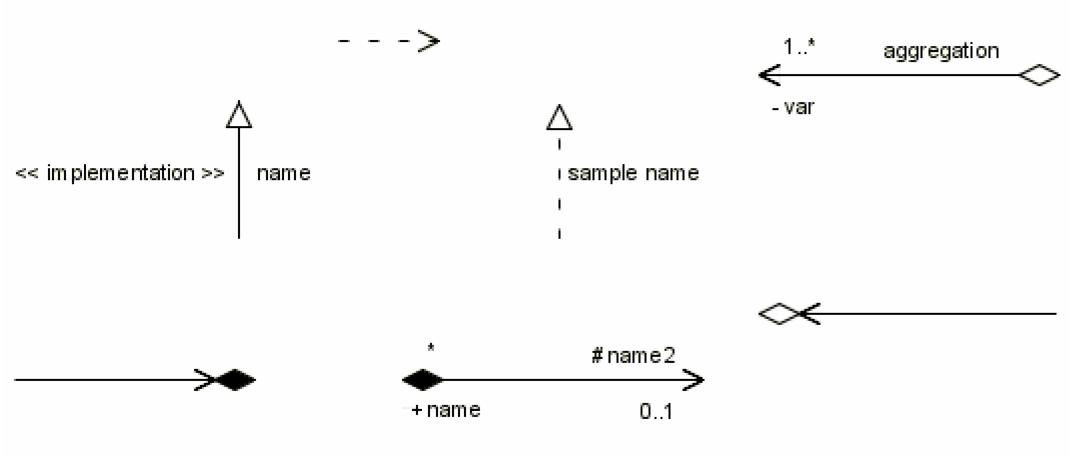
JFC Swing

UML 2.0 Diagram Interchange

1.4 Required Algorithms

1.4.1 How to Draw the Edges

A simple Poseidon-like drawing style implemented for the all edges. Each edge is based on edge line (continuous or dashed) and edge ending. Several text fields can appear to the edge and edge endings. A quick reference picture is below.



1.4.2 How to Get Information for Edges

This component is tightly coupled with two others: Diagram Edges and Diagram Interchange. These components provide all required information for each edge.

First component, Diagram Edges, represents a graphical side of the edge. Several base classes are inherited from the JComponent. The Edge class defines common functionality for the all concrete edges. For example, the edge body is drawn in the paintComponent method of this class. The lineStyle, leftEnding and rightEnding parameters for the edge are stored and reused in this class. The Edge class constructor registers both endings are the child graphical components, so they will be updated automatically. This class also has a map of textFields, corresponding to the name and stereotypes compartments. This map should be filled by appropriate compartments in the concrete edge class constructor. And the last important feature of the Edge class is working with selection. The selection waypoint values retrieved by the Diagram Edges component from the corresponding GraphEdge UML model element automatically.

The second component, Diagram Interchange, represents a UML model side of the edge. Such data as position, waypoints, and visibility can be directly retrieved from GraphElement, GraphEdge, DiagramElement classes. The DiagramElement class contains a collection of properties, which accessed through the getProperties method. These properties relate to the font family, color and so on information. To be familiar with such data, the two another documents should be used. The common description for the Diagram Interchange Specification can be found in the "DiagramInterchangeSpec.pdf" file (or from the external link: http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-01.pdf). It clearly describes the purpose and features of the Diagram Interchange component. Another document if "**UML Tool - Class Diagram Elements Compartments.rtf**" file (or it can be found on the TopCoder forums). This file defines properties of the concrete UML Model elements, such as Association, AssociationEnd and so on. First, the all elements of the UML diagram are linked in the graph. You can travel from leaves to the root and back on this hierarchy, because DiagramElement and GraphElement are linked together. GraphElement class has "containeds" list of child nodes, and each DiagramElement class has "container" reference to its parent. The related getters methods are implemented.

To get the type of the UML Model element, related to the GraphEdge, you should get the "semanticModel" from the GraphElement, then convert its type to the Uml1SemanticModelBridge, and next retrieve the UML Model element by "getElement" method. Then you can convert the type of this element to the concrete type (and check for null), retrieve simple properties of the element (such as isAbstract and so on). Also consult "**UML Tool - Class Diagram Elements Compartments.rtf**" file for getting name and multiplicity of the edge endings.

1.5 Component Class Overview

1.5.1 com.topcoder.gui.diagramviewer.uml.classedges namespace

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 Association, Generalization, Abstraction and Dependency edges.

There are name compartment and stereotype compartment properties. The objects for these compartments are created in the constructor.

This class also adds EditBoxTrigger to name compartment automatically to allow name editing. PopupMenuTrigger and EdgeSelectionTrigger are also registered automatically to edge and name/stereotype compartments to support popup menu and selection for the edge.

AssociationEdge:

This class is the concrete Edge of this component. It is an extension of BaseEdge class. The required information is retrieved from the Association class from the UML Model and from the GraphEdge associated with it. The special active edge endings will be assigned to the edge. The class uses a continuous line, an open ending and the several different other endings: simple arrow, empty diamond, filled diamond, empty diamond with simple arrow, filled diamond with simple arrow.

This class implements creating of the concrete line style for the edge and the concrete edge endings (with compartments). The drawing of the edge is implemented in the parent Edge class (in the Diagram Edges component). The edge endings (and their compartments) painted automatically as child graphical components.

GeneralizationEdge:

This class is the concrete Edge of this component. It is an extension of BaseEdge class. The required information is retrieved from the Generalization class from the UML Model and from the GraphEdge associated with it. No active edge endings are supported. The class uses a continuous line, an open ending and the closed arrow (triangle) ending.

This class implements creating of the concrete line style for the edge and the concrete edge endings (with compartments). The drawing of the edge is implemented in the parent Edge class (in the Diagram Edges component). The edge endings (and their compartments) painted automatically as child graphical components.

AbstractionEdge:

This class is the concrete Edge of this component. It is an extension of BaseEdge class. The required information is retrieved from the Abstraction class from the UML Model and from the GraphEdge associated with it. No active edge endings are supported. The class uses a dashed line, an open ending and the closed arrow (triangle) ending.

This class implements creating of the concrete line style for the edge and the concrete edge endings (with compartments). The drawing of the edge is implemented in the parent Edge class (in the Diagram Edges component). The edge endings (and their compartments) painted automatically as child graphical components.

DependencyEdge:

This class is the concrete Edge of this component. It is an extension of BaseEdge class. The required information is retrieved from the Dependency class from the UML Model and from the GraphEdge associated with it. No active edge endings are supported. The class uses a dashed line, an open ending and the simple arrow ending.

This class implements creating of the concrete line style for the edge and the concrete edge endings (with compartments). The drawing of the edge is implemented in the parent Edge class (in the Diagram Edges component). The edge endings (and their compartments) painted automatically as child graphical components.

TextField:

Text field represents pure text compartment of 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 edge. There is no decorator around or on the text.

It extends from com.topcoder.gui.diagramviewer.edges.TextField to allow to be added to Edge. After the Diagram Edges component is finally released, the methods and attributes already defined in base class should be removed in this class.

ClosedArrow:

This class represents the simple closed arrow (triangle) used in edges: |>.

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

OpenArrow:

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.

ActiveEdgeEnding:

This class is the base of active edge endings in this component. It is an extension of EdgeEnding class served for the purpose of providing all the common functionalities required by several edge endings for AssociatioinEdge.

It contains several compartments. They are name compartment and multiplicity compartment represented by TextField. There are several properties allowing to configure visibility when selected and to hide values for these compartments.

The reference to parent edge allows getting common properties (such as colors) from the related edge.

The selection feature is provided. This class also adds EditBoxTrigger to name compartment automatically to allow name editing. PopupMenuEndingTrigger and EdgeEndingSelectionTrigger are also registered automatically to edge ending and name/multiplicity compartments to support popup menu and selection for the edge ending.

NoArrow:

This class represents the no-arrow ending used in edges: ---.

It just draws nothing, but allows selection, name/multiplicity and edge ending popup menu, because it is a child of the ActiveEdgeEnding.

DirectedAssociation:

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. It also allows selection, name/multiplicity and edge ending popup menu, because it is a child of the ActiveEdgeEnding.

Aggregation:

This class represents the empty diamond used in edges: <>.

It can draw an empty (white) diamond at specified location and angle, and it also can determine whether given point is in the edge ending diamond. It also allows selection, name/multiplicity and edge ending popup menu, because it is a child of the ActiveEdgeEnding.

AggregationBiDirectional:

This class represents the empty diamond (with a simple arrow) used in edges: ><>.

It can draw an empty (white) diamond (with a simple arrow) at specified location and angle, and it also can determine whether given point is in the edge ending diamond and arrow triangle. It also allows selection, name/multiplicity and edge ending popup menu, because it is a child of the ActiveEdgeEnding.

Composition:

This class represents the filled diamond used in edges: <*>.

It can draw a filled (with the color of parent edge) diamond at specified location and angle, and it also can determine whether given point is in the edge ending diamond. It also allows selection, name/multiplicity and edge ending popup menu, because it is a child of the ActiveEdgeEnding.

CompositionBiDirectional:

This class represents the filled diamond (with a simple arrow) used in edges: ><*>.

It can draw a filled (with the color of parent edge) diamond (with a simple arrow) at specified location and angle, and it also can determine whether given point is in the edge ending diamond and arrow triangle. It also allows selection, name/multiplicity and edge ending popup menu, because it is a child of the ActiveEdgeEnding.

1.5.2 com.topcoder.gui.diagramviewer.uml.classedges.event namespace

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().

TextChangeListener:

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.

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.

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 dependency node, and the text field representing name compartment will be associated with this trigger. As a result, when the dependency node is clicked, the name compartment will be editable.

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 edge in this component automatically.

PopupMenuEndingTrigger:

This is an event listener which listens to mouse right clicked event for active edge endings. If the event occurs, popup menu for edge ending would be shown. This event listener will be registered to every active edge ending (and their TextFields) in this component automatically.

EdgeSelectionTrigger:

This class can trigger selection when some edge is clicked. A BaseEdge instance should be given to this class to understand which edge should be selected.

For example, this trigger can be registered to a dependency edge, and the text field representing name compartment will be associated with this trigger. As a result, when the dependency edge is clicked, the edge and its compartments will be selected.

EdgeEndingSelectionTrigger:

This class can trigger selection when some edge ending is clicked. An ActiveEdgeEnding instance should be given to this class to understand which edge ending should be selected.

For example, this trigger can be registered to a composition edge ending, and the text field representing name compartment will be associated with this trigger. As a result, when the composition edge ending is clicked, the edge ending and its compartments will be selected.

1.6 Component Exception Definitions

The component defines a custom exception and reuses system-defined exceptions too.

IllegalGraphElementException:

This exception is used to indicate some GraphEdge is illegal in specific situation. For example, if a Generalization GraphEdge is given to the AssociationEdge constructor. It could be thrown when retrieving graph information from 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.

1.7 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: Java 1.5
- Compile target: Java 1.5

2.2 TopCoder Software Components

- Diagram Edges 1.0 – the basic functionality is used.
- Diagram Viewer 1.0 – allows editing name of edges. It holds the edges defined in the class diagram.
- Diagram Interchange 1.0 – defines top-level parent classes (GraphNode, GraphEdge and so on). It provides the standard UML diagram information, including size, position and etc.
- UML Model Core 1.0 – realization of DiagramElement. This component defines all the UML Model elements, like Association, Generalization, and etc.
- Base Exception 1.0 – contains the basic functionality for the custom exception.

Not used components from Requirement Specification – Configuration Manager 2.1.5.

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

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.classedges – contains all the class diagram edges.

com.topcoder.gui.diagramviewer.uml.classedges.event – 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 an instance of the edge
AssociationEdge assocEdge = new AssociationEdge(some graph edge);

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

  public void textChange(TextChangeEvent e) {
    // retrieve the TextField, and package node.
    TextField textField = e.getSource();
    BaseEdge ascEdge = textField.getParent();

    // get package model element.
    GraphNode graphNode = ascEdge.getGraphNode();
    Association modelElement = graphNode.getModelElement();
    // actually change the name
    modelElement.setName(e.getNewText());

    // get preferred graphNode size
    Dimension newSize = graphNode.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.
    ascEdge.notifyGraphEdgeChange();
  }
}

// register the text change listener to the name compartment.
// the name can be changed
assocEdge.getNameCompartment().addTextChangeListener(new NameChangeListener());

// Retrieve compartments
TextField nameCompartment = assocEdge.getNameCompartment();
TextField stereotypeCompartment = assocEdge.getStereotypeCompartment();

// Set popup menu
JPopupMenu edgePopup = new JPopupMenu…
assocEdge.setPopup(edgePopup);


// Work with endings
ActiveEdgeEnding edgeEnding = assocEdge.getLeftEnding();
TextField endingName = edgeEnding.getName();
TextField endingMultiplicity = edgeEnding.getMultiplicity();
edgeEnding.multiplicityVisibleWhenSelected(false);
edgeEnding.nameVisibleWhenSelected(true);
edgeEnding.setHideNameText("");

// Hiding compartments
edgeEnding.nameVisible(false);
edgeEnding.multiplicityVisible(false);
... change assocEdge name and stereotype visiblity through Diagram Interchange,
and:
assocEdge.notifyNameVisibilityChanged();
assocEdge.notifyStereotypeVisibilityChanged();
```

5. Future Enhancements

- To implement another rendering scheme for edges (for example, with shadows or 3D-like).