Diagram UML Class Elements 1.0 Requirements Specification

1. Scope

1.1 Overview

The Diagram UML Class Elements component provides the graphical diagram elements representing the model elements specific to a class diagram.

The elements provided are the package, the class, the interface, the exception and the enumeration. The last four elements are very similar so they will be represented almost in the same manner. The elements extend from the base element from the Diagram Elements component, providing the visual aspect of each element.

1.2 Logic Requirements

1.2.1 PackageNode

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

The structure of the GraphNode that corresponds to a Package is given in the “UML Tool – Class Diagram Elements Compartments.rtf” file posted on the forums. The file shows the structure of GraphNodes (from Diagram Interchange component) that makes up a GraphNode of a Package.

1.2.1.1 The visibility attribute of the compartments

The stereotype and package compartments could be hidden. The ‘isVisible’ attribute of the compartment graph node is for this property. There should be methods to tell the package that the compartments’ visibility has changed.

1.2.1.2 Resize event

The package will be resized and an event will be triggered, to signal the size change (the reason is also passed, as a string). The package node should also have a method that computes the preferred size, according to the new visible/hidden compartment.

1.2.1.3 The stereotype and package compartments’ reaction to events

The stereotype and package compartments do not react to any events. They let the events pass to the NameCompartment, which reacts as described below.

1.2.1.4 React to double-click event, by showing the name input tool

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 package. The text of the Name compartment will not be shown while the edit control is up (though the package 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 package and graph node for which the name is set. The package will also provide a method to check the size that will be required for the package if the name would be set (this is required, as other representations of the same Package 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 package, and it will generate a resize event (the reason is also passed, as a string).

1.2.1.5 Drop Target

The graphical component will be implemented as a drag and drop DropTarget. It should provide a pluggable handler for the Transferable element that will transform it into the actual graphical element. The element will be added after that to the package's body compartment. The component should make it easy to the application to add the new graphical element to the BodyCompartment (note that the new graphical element is not added in the JComponent container hierarchy; it is a parallel element of the package added to the diagram view, but drawn in front of the package). There should also be a way for the application to query the size of the BodyCompartment and whether the new graphical element would fit at a certain location, in order to be able to perform the resize. The location of the mouse event is translated so it fits on top of the BodyCompartment.

1.2.1.6 Minimum and preferred size

The package should define a minimum size. It should also have a method that computes the preferred size, according to the name compartment and the existing contained graph nodes. It should also provide a method to compute the preferred size if a new element would be added (same thing as for the new name situation) or if the size of a certain contained element is changed.

1.2.1.7 Reaction to adding new elements to the diagram

In case the diagram viewer's flag for adding new elements from the toolbar is on, the package should react to mouse events differently. When the mouse is pressed, it should consume the event:

- in case the new element is a node:
    - mouse clicked - by accepting the new element (it provides a validator that simply returns true). It should translate the point to an appropriate position first, in case the event hasn't occurred on the body compartment. It should generate an event, with the position where the element should be added.
- in case the new element is an edge:
    - mouse pressed - by accepting the edge end, if the edge is a Dependency. Otherwise, it will consume the event by and calling a pluggable handler that will bring the state of the diagram viewer and element toolbar to a neutral state (setting the flag of the diagram viewer to false...).
    - mouse released - by accepting the edge end, if the edge is a Dependency.

Therefore, for all the situations above, there should be an event triggered, with the proper information. Other handlers will perform the actual operations.

1.2.1.8 Edge connector

There will be a connector for the edges defined for the package. It will behave differently than the connector for the rectangle shaped elements. At the top part, it will return the connector point by following the outer line of the name compartment and the top of the body compartment for the right side.

1.2.1.9 Popup

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

1.2.1.10 Attributes

The package should have several properties configurable through the graph node:

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

1.2.2 InterfaceNode

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

The structure of the GraphNode that corresponds to an Interface is given in the “UML Tool – Class Diagram Elements Compartments.rtf” file posted on the forums. The file shows the structure of GraphNodes (from Diagram Interchange component) that makes up a GraphNode of an Interface.

1.2.2.1 The visibility attribute of the compartments

The stereotype and package compartments 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 compartments’ visibility has changed.

1.2.2.2 Resize event

The interface will be resized and an event will be triggered, to signal the size change (the reason is also passed, as a string). The interface node should also have a method that computes the preferred size, according to the new visible/hidden compartment.

1.2.2.3 The stereotype and package compartments’ reaction to events

The stereotype and package compartments do not react to any events. They let the events pass to the NameCompartment, which reacts as described below.

1.2.2.4 React to double-click event, by showing the name input tool

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 interface (the attributes and methods do not conform to this). The rest of the ‘name editing’ functionality is similar to that of the Package.

1.2.2.5 Drop Target

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 interface and the event should be handled by the package or diagram behind it (as the intention of the user is to add the element to the package, or diagram).

1.2.2.6 Attributes and operations compartments

The component has a compartment for the attributes and one for operations.

a) When there is no feature declared, the compartments will have a default minimum size. These compartments can be shown or hidden, according to the “isVisible” attribute of the compartment’s graph node. The individual attributes and operations could be shown or hidden also, but this is not a requirement. The stereotype compartment of each attribute or operation can be shown or hidden. There should be methods to show/hide the compartments.

b) The attributes inside the AttributeCompartment will be ordered according to the order of the features inside the Classifier’s list of features. Same thing for the operations.

c) In case there is no name set for the attribute, or no type, or no return value, the component will display nothing, or display a configurable value. Same thing goes if the type has no name.

d) The attributes will be selectable. The component will add the attribute to the list of selectable elements in the diagram viewer (or will set it, if Ctrl is not pressed). Same thing goes for the operations. An event is also triggered (a GUI event).

e) Each attribute and operation will react to a mouse double-click event, by showing the edit control of the DiagramViewer in order to edit the attribute or operation. The ‘editing’ functionality is similar to that of the name compartment. The text passed for an attribute to the text editing control contains the visibility, the name, the type and the initial value. The text passed for an operation to the text editing control contains the visibility, the name, the parameters and the return type.

1.2.2.7 Minimum and preferred size

The interface should define a minimum size. It should also have a method that computes the preferred size, according to the name compartment and the existing attributes and operations. It should also provide a method to compute the preferred size if a new feature would be added (same thing as for the new name situation) or if the size of a certain owned feature would be changed.

1.2.2.8 Reaction to adding new elements to the diagram

In case the diagram viewer’s flag for adding new elements from the toolbar is on, the interface should react to mouse events differently:

- in case the new element is a node:
    - mouse clicked - by not consuming the event. The event should pass to the element behind it.
- in case the new element is an edge:
    - mouse pressed - by accepting the edge end, if the edge is not an Abstraction. Otherwise, it will consume the event by and calling a pluggable handler that will bring the state of the diagram viewer and element toolbar to a neutral state (setting the flag of the diagram viewer to false...).

- mouse released - by accepting the edge end (if the edge is a Generalization and if the other end's classifier is not an Interface, it should be rejected).

Therefore, for all the situations above, there should be an event triggered, with the proper information. Other handlers will perform the actual operations.

1.2.2.9 Edge connector

There will be a connector for the edges defined for the package. It will behave differently than the connector for the rectangle shaped elements. At the top part, it will return the connector point by following the outer line of the name compartment and the top of the body compartment for the right side.

1.2.2.10 Popups

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, a popup for all the attributes, but there should be a way to determine which attribute is the one for which the popup is shown, and the same thing for the operations. There must be a way to reuse the same popup for all the attributes, in order to keep the memory low.

1.2.3 ClassNode

These actions are similar to the ones above. They use Class, instead of Interface.

The name could be shown in italics if the instance is abstract.

1.2.4 ExceptionNode

These actions are similar to the ones above. They use Class with an <<exception>> stereotype, instead of a simple Class.

1.2.5 EnumerationNode

These actions are similar to the ones above. They use Enumeration, instead of Class.

It has an extra EnumerationLiteralCompartment above the AttributesCompartment. The EnumerationLiterals have only the name and the stereotype compartment.

1.2.6 Showing / hiding compartments

The elements above will be able to show or hide the compartments or elements according to the DiagramElement.isVisible attribute.

There should be a way to set through the API the visible flag of the contained compartments and elements.

There should be a way for the edges to configure for what values of the edge texts these texts will not be shown and to configure which texts will be shown when the edge is selected.

1.2.7 Receive events location

The node should receive events only in the drawn shape.

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 UML class elements in the diagrams.

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.classelements

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 - 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.