Diagram UML Sequence Elements 1.1 Requirements Specification

1. Scope

1.1 Overview

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

Version 1.1 adds the lifeline drawing functionality on the Object and will fix the way the edges are drawn. The edges will be fixed so that each UML line will have only one corresponding edge in the diagram, not two.

An edge layout algorithm will be provided also.

1.2 Logic Requirements

1.2.1 Object Node

This class is a concrete Node. It takes its information from the Object 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 node that the compartment's visibility has changed. The 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 show "anonymous" if there is no name, or it could be hidden.

The stereotype compartment do not react to any events. It lets 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 all the selection corners, when it is selected. The corners will be shown around the shape.

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 vertical axis of the shape.

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

The component should receive events only in the drawn shape.

The component should show

The object can move only horizontally. It should have a configurable location on the "y" axis. When the header is resized (due to stereotype compartment being shown), this location could be changed.

1.2.2 Create Message Edge

This class is a concrete Edge. It takes its information from the Link with a Stimulus with a Procedure that has a CreateObjectAction as the Action class from the UML Model and from the GraphEdge associated with it.

The structure of the GraphEdge 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 edge that the compartment's visibility has changed.

The edge has a continuous line. The stereotype compartment contains the "create" stereotype (as KeywordMetaclass). It cannot be removed, it is shown first and is separated from the other stereotypes by a blank space.

There should be two end types defined:

- sender end: none
    - o ----
- receiver end: a filled 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 not do anything.

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.

The line cannot be dragged or split into segments. It is fixed, based on an external algorithm.

1.2.3 SynchronousMessage Edge

This class is a concrete Edge. It takes its information from the Link with a Stimulus with a Procedure that has a CallOperationAction (a synchronous action) as the Action class from the UML Model and from the GraphEdge associated with it.

The structure of the GraphEdge that corresponds to an Extend is given in section 3.2.1.

1.2.4 AsynchronousMessage Edge

This class is a concrete Edge. It takes its information from the Link with a Stimulus with a Procedure that has a CallOperationAction (an asynchronous action) as the Action class from the UML Model and from the GraphEdge associated with it.

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

The difference is in the edge ends:

- sender end: none
    - o ----
- receiver end: an empty arrow, with the color of the edge
    - o ---->

1.2.5 SendSignalMessage Edge

This class is a concrete Edge. It takes its information from the Link with a Stimulus with a Procedure that has a SendSignalAction as the Action class from the UML Model and from the GraphEdge associated with it.

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

The difference is in the edge ends:

- sender end: none
    - o ----
- receiver end: a half of an empty arrow, with the color of the edge
    - o ----\

These actions are similar to the ones above. Only they use a Stimulus with a Procedure that has a SendSignalAction as the Action.

1.2.6 Return Message Edge

This class is a concrete Edge. It takes its information from the Link with a Stimulus with a Procedure that has no Action class from the UML Model and from the GraphEdge associated with it.

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

The edge is similar to the AsynchronousMessage edge, but the line is dashed.

1.2.7 Lifeline segments

The Object will have lifeline segments drawn. These are the vertical rectangles drawn on Object's vertical lifeline. The colors of the lifelines (stroke and fill) will default to Object's colors. There must be an option to set different colors for a segment.

1.2.7.1 The colors of the segments will be set in the Diagram Interchange model using Properties for the Links that start the lifeline segments. The names of these properties will be configurable. The value of the colors must have this format: "#RRGGBB".
The Lifeline segments can be selected with the mouse, just like the Object can be selected. It does not need resize functionality, just a way to visually show that the segment was selected (like

a wider stroke line using the same stroke color). An event will be triggered to signal that the segment was selected

To cover this part of the functionality, the Lineline segments will not be set or loaded here. There is a layout algorithm required below, which will have the task to arrange the edges and set the Lineline segments to be drawn.

1.2.7.2 The width of the lifeline segments will be configurable in the API.

Note that the lifeline with the first edge will have a lifeline segments drawn from where the object header down to where the first arrow starts. The properties for this lifeline are saved in the Object itself, not in the Links.

1.2.7.3 All Objects will be drawn initially with a dotted line. When the first edge is drawn, the Object's lifeline will have a Lifeline segment drawn on it.

1.2.8 Edge layout algorithm

A pluggable edge layout algorithm must be defined, with a concrete implementation. This layout algorithm will have the task to arrange the edges, by setting their location and by setting the lifeline segments.

In the default implementation, the edges will be arranged like a chain. There will not be two edges drawn on the same level. The spacing between the edges will be configurable and it will be constant.

Note that all the arrows start a new lifeline segment in the destination object. And also note that the asynchronous message is the only one that does not break the lifeline segment in the source lifeline. For all the other messages, the lifeline segments end in the position where the arrow starts.

1.2.9 Fixes for Edges

1. The Edges will have only one line associated with them, not two. To cover a proper method lifecycle in the sequence diagrams, the user will draw a synchronous call, for instance, and one return arrow.

2. The edges must be drawn in such a way that they will not overlap the lifeline segments drawn.

3. The edges will react to dragging events by triggering an event. Adding the same functionality to the Lifeline segments will be considered major functionality.

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 sequence 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.sequenceelements

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

Lifeline segment width
Edge spacing

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

UML Tool - Sequence Diagram Elements Compartments.rtf

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None

3.2.2 TopCoder Software Component Dependencies:

- Action Manager 1.0
- UML Model Manager 1.0
- UML Project Configuration 1.0
- UML Model components
- 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.