UML Tool Actions – Auxiliary Elements Actions 1.0 Component Specification

1. Design

The Auxiliary Elements Actions component provides the Actions related to the auxiliary elements specific to all diagrams. The actions are strategy implementations of the action interfaces in the Action Manager component. The provided actions are for adding / removing / copying / cutting / pasting the elements and relationships. The elements are comment, free text and polyline.

This design is quite straightforward, and it follows the requirement specification exactly to implement all required project actions as mentioned above.

The one thing that worth mention is the cut/copy logic used in this design, as we ought to cut/copy the objects across different running applications/JVMs, the custom Transferable implementation (the AuxiliaryElementSelection class) and several DataFlavor objects provided by this design will make the cut/copied objects be serialized to the clipboard, and we eventually get the deserialized objects from clipboard and pass them to the paste actions. Both serialization and deserialization mentioned above are done by the JDK automatically; this design only follows the necessary principals in order to make it work.

NOTE: When doing cut/copy actions, if user doesn't provide the clipboard argument, the system clipboard retrieved from Toolkit.getDefaultToolkit().getSystemClipboard() should be used instead.

1.1 Design Patterns

- **Strategy Pattern** – All concrete action classes in this design are strategy implementations of the action interfaces in the Action Manager component.
- **Utility Pattern –** The AuxiliaryElementCliboardUtility and AuxiliaryElementCloneUtility classes implement this pattern to provide utility methods to this design.
- **Template Method Pattern –** The AbstractCutGraphElementAction abstract class and its subclasses implement this pattern (Specifically, the copyToClipboard protected method).

1.2 Industry Standards

UML 1.5

1.3 Required Algorithms

For those are not quite familiar with the java Clipboard stuff, the following link might be helpful: http://www.javaworld.com/javaworld/jw-08-1999/jw-08-draganddrop.html.

1.4 Component Class Overview

1.4.1 Package com.topcoder.uml.actions.auxiliary

- **AbstractAuxiliaryUndoableAction**: This abstract class implements the UndoableAction interface and extends the AbstractUndoableEdit abstract class. It is used as the base class for all the undoable actions in this design, it also provides a logging method that could be used by its subclasses to log the exceptions raised in the redo/undo methods, and its name attribute is used as the action's presentation name.
- **AbstractPasteGraphElementAction**: This abstract class extends the AbstractAuxiliaryUndoableAction abstract class to paste the child graph element into

the parent graph node. It is used as the base-class for all paste graph element actions.

- **AbstractRemoveGraphElementAction**: This abstract class extends the AbstractAuxiliaryUndoableAction abstract class, and it is used as the base-class for all actions that are responsible for removing graph element from its parent graph node.
- **AbstractAddGraphElementAction**: This abstract class extends the AbstractAuxiliaryUndoableAction abstract class, and it is used as the base-class for all actions that are responsible for adding graph element into another graph node.
- **AbstractCutGraphElementAction**: This abstract class extends the AbstractAuxiliaryUndoableAction abstract class, and it is used as the base-class for all actions that are responsible for copying the child graph element to the clipboard and removing the child graph element from its parent graph node.
- **AuxiliaryElementSelection**: This class implements the Transferable interface, and it is used to transfer the auxiliary objects (which are serializable) to the clipboard.
- **AuxiliaryElementDataFlavor**: This class defines all the DataFlavor constants used in this design.
- **AuxiliaryElementCliboardUtility**: This utility class is used to copy specific auxiliary element to the clipboard.
- **AuxiliaryElementCloneUtility**: This is a utility class, and it provides method to clone the auxiliary model object for the cut/copy actions.

1.4.2 Package com.topcoder.uml.actions.auxiliary.comment.diagram

- **AddCommentGraphNodeAction**: This class extends the AbstractAddGraphElementAction abstract class, and it is responsible for adding the comment graph node into the diagram graph node.
- **AddCommentRelationshipGraphEdgeAction**: This class extends the AbstractAddGraphElementAction abstract class, and it is responsible for adding the comment relationship GraphEdge into the diagram graph node.
- **RemoveCommentGraphNodeAction**: This class extends the AbstractRemoveGraphElementAction abstract class, and it is responsible for removing the comment graph node from its parent diagram graph node.
- **RemoveCommentRelationshipGraphEdgeAction**: This class extends the AbstractRemoveGraphElementAction abstract class, and it is responsible for removing the comment relationship graph edge from its parent diagram graph node.
- **PasteCommentGraphNodeAction**: This class extends the AbstractPasteGraphElementAction abstract class, and it is responsible for pasting the comment graph node into the diagram graph node.
- **PasteCommentRelationshipGraphEdgeAction**: This class extends the AbstractPasteGraphElementAction abstract class, and it is responsible for pasting the comment relationship GraphEdge into the diagram graph node.
- **CutCommentGraphNodeAction**: This class extends the AbstractCutGraphElementAction abstract class, and it is responsible for removing the comment graph node from its contained diagram graph node, and copying it to the clipboard.
- **CutCommentRelationshipGraphEdgeAction**: This class extends the AbstractCutGraphElementAction abstract class, and it is responsible for removing the comment graph edge from its contained diagram graph node, and copying it to the clipboard.

- **CopyCommentGraphNodeAction**: This class implements the TransientAction interface, and it is responsible for copying the comment graph node to the clipboard.
- **CopyCommentRelationshipGraphEdgeAction**: This class implements the TransientAction interface, and it is responsible for copying the comment graph edge to the clipboard.

1.4.3 Package com.topcoder.uml.actions.auxiliary.comment.model

- **AddCommentAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for adding the comment into the Model or the provided Namespace.
- **AddNoteRelationshipAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for adding the comment into the provided ModelElement's comments attribute.
- **RemoveCommentAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for removing the comment from its contained Namespace.
- **RemoveNoteRelationshipAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for removing the comment from the comments list of all its annotated ModelElement objects.
- **PasteCommentAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for pasting the comment into the Model or the provided Namespace.
- **PasteNoteRelationshipAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for pasting the comment into the into the provided ModelElement's comments attribute.
- **CutCommentAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for removing the comment from its contained Namespace, and copying the comment to the clipboard.
- **CutNoteRelationshipAction**: This class extends the AbstractAuxiliaryUndoableAction abstract class, and it is responsible for removing the comment from the comments list of its annotated ModelElement objects and copying the comment object to the clipboard.
- **CopyCommentAction**: This class implements the TransientAction interface, and it is responsible for copying the comment object to the clipboard.
- **CopyNoteRelationshipAction**: This class implements the TransientAction interface, and it is responsible for copying the note relationship to the clipboard.
- **ChangeCommentTextAction**: This class extends AbstractAuxiliaryUndoableAction abstract class; it is used to change the comment's body text to a new one.

1.4.4 Package com.topcoder.uml.actions.auxiliary.diagram

- **AddFreeTextAction**: This class extends the AbstractAddGraphElementAction abstract class, and it is responsible for adding the free text graph node into the diagram graph node.
- **AddPolylineAction**: This class extends the AbstractAddGraphElementAction abstract class, and it is responsible for adding the polyline graph edge into the diagram graph node.
- **RemoveFreeTextAction**: This class extends the AbstractRemoveGraphElementAction abstract class, and it is responsible for removing the free text graph node from its contained diagram graph node.

- **RemovePolylineAction**: This class extends the AbstractRemoveGraphElementAction abstract class, and it is responsible for removing the polyline graph edge from its contained diagram graph node.
- **PasteFreeTextAction**: This class extends the AbstractPasteGraphElementAction abstract class, and it is responsible for pasting the free text graph node into the diagram graph node.
- **PastePolylineAction**: This class extends the AbstractPasteGraphElementAction abstract class, and it is responsible for pasting the polyline graph edge into the diagram graph node.
- **CutFreeTextAction**: This class extends the AbstractCutGraphElementAction abstract class, and it is responsible for removing the free text graph node from its contained diagram graph node and copying it to the clipboard.
- **CutPolylineAction**: This class extends the AbstractCutGraphElementAction abstract class, and it is responsible for removing the polyline graph edge from its contained diagram graph node and copying it to the clipboard.
- **CopyFreeTextAction**: This class implements the TransientAction interface, and it is responsible for copying the free text graph node to the clipboard.
- **CopyPolylineAction**: This class implements the TransientAction interface, and it is responsible for copying the polyline graph edge to the clipboard.

1.5 Component Exception Definitions

1.5.1 Custom Exceptions

- **AuxiliaryElementCloneException**: This class extends the ActionExecutionException class, and this exception is used if the clone operation of auxiliary objects fails.
- **ActionExecutionException**: This exception is from the Action Manager component, all actions in this design will throw this exception in the execute methods.

1.5.2 System Exceptions

- **IllegalArgumentException**: Used wherever empty String argument is used while not acceptable. Normally an empty String is checked with trimmed result. It also thrown when null argument is passed in or some other cases when the argument is invalid.
- **CannotUndoException**: Thrown from undo method if the action cannot be undone.
- **CannotRedoException**: Thrown from redo method if the action cannot be redone.

1.6 Thread Safety

This component is not thread-safe, and there is no need to make it thread-safe, as each thread is expected to create its own action object to perform the task rather than share the same action object in different threads. And there is no such requirement too.

Though all actions are immutable, the internal state of their instance variable could be changed, so they are all not thread-safe.

2. Environment Requirements

2.1 Environment

Java 1.5

2.2 TopCoder Software Components

- **Action Manager 1.0**: The UndoableAction and TransientAction interfaces implemented by this design are from this component.

- **UML Model Manager 1.0**: This component is not used directly by this design. But it is referenced indirectly by UML Project Configuration component.
- **UML Project Configuration 1.0**: The ProjectConfigurationManager class used in this design is from this component.
- **Logging Wrapper 1.2**: It is used to log exception in undoable actions' redo and undo methods.
- **Diagram Interchange Version 1.0**: GraphNode, GraphEdge and GraphElement classes used by this design are from this component.
- **UML Model - Core Auxiliary Elements 1.0**: Comment interface used by this design is from this component.
- **UML Model - Core Requirements 1.0**: Namespace and ModelElement interfaces used by this design are from this component.
- **UML Model - Model Management 1.0**: Model interface used by this design is from this component.

NOTE: Configuration Manager is not used as all the configurable properties are passed into actions' constructors directly as arguments, which are more convenient as some arguments cannot be easily created from the configured properties (e.g. the ProjectConfigurationManager object, which is supposed to be shared in multiple actions), and it is also more flexible to set these arguments directly to constructor than load them from Configuration Manager.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.actions.auxiliary

com.topcoder.uml.actions.auxiliary.diagram

com.topcoder.uml.actions.auxiliary.comment.model

com.topcoder.uml.actions.auxiliary.comment.diagram

3.2 Configuration Parameters

None.

3.3 Dependencies Configuration

None.

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

None.

4.3 Demo

Assume the modelManager (UMLModelManager object), comment (Comment object), note (Comment object), commentGraphNode (GraphNode object for CommentGraphNode actions), commentGraphEdge (GraphEdge object for CommentRelationshipGraphEdge actions), freeTextGraphNode (GraphNode object for FreeText actions), polylineGraphEdge (GraphEdge object for Polyline actions), namespace (Namespace object to add comment), diagram (GraphNode object to add graph-elements), clipboard (Clipboard object) are provided by the application.

4.3.1 Execute Comment related actions

```
// create an add action
AddCommentAction action1 = new AddCommentAction(comment, modelManager);

// execute the action to add the comment to the Model.
action1.execute();

// execute action to add the comment to the given namespace
action1 = new AddCommentAction(comment, namespace, modelManager);
action1.execute();

// undo the action
action1.undo();

// redo the action
action1.redo();


// create a remove action
RemoveCommentAction action2 = new RemoveCommentAction(comment);

// execute the action to remove the comment from its namespace
action2.execute();

// undo the action
action2.undo();

// redo the action
action2.redo();

// create a copy action
CopyCommentAction action3 = new CopyCommentAction(comment, clipboard);

// copy to the given clipboard
action3.execute();

// copy to system clipboard
action3 = new CopyCommentAction(comment, null);
action3.execute();

// create a cut action
CutCommentAction action4 = new CutCommentAction(comment, clipboard);

// remove from its namespace and copy to the given clipboard
action4.execute();

// undo the action
action4.undo();

// redo the action
action4.redo();
```

```
// create a paste action
PasteCommentAction action5 =
      new PasteCommentAction(comment, modelManager);

// paste to the Model
action5.execute();

// paste to the given namespace
action5 = new PasteCommentAction(comment, namespace, modelManager);
action5.execute();

// undo the action
action5.undo();

// redo the action
action5.redo();
```

4.3.2 *Execute NoteRelationship related actions*

```
// create an add action to add the note to the given modelElement
AddNoteRelationshipAction action1 = new AddNoteRelationshipAction(note,
namespace, modelManager);
action1.execute();

// undo the action
action1.undo();

// redo the action
action1.redo();

// create a remove action
RemoveNoteRelationshipAction action2 =
      new RemoveNoteRelationshipAction(note);

// execute the action to remove the note from its annotated elements
action2.execute();

// undo the action
action2.undo();

// redo the action
action2.redo();

// create a copy action
CopyNoteRelationshipAction action3 = new CopyNoteRelationshipAction(note,
clipboard);

// copy to the given clipboard
action3.execute();

// copy to system clipboard
action3 = new CopyNoteRelationshipAction(note, null);
action3.execute();

// create a cut action
```

```
CutNoteRelationshipAction action4 = new CutNoteRelationshipAction(note,
clipboard);

// remove from its annotated elements and copy to the given clipboard
action4.execute();

// undo the action
action4.undo();

// redo the action
action4.redo();

// create a paste action to add the note to the given modelElement
PasteNoteRelationshipAction action5 = new
PasteNoteRelationshipAction(note, namespace, modelManager);
action5.execute();

// undo the action
action5.undo();

// redo the action
action5.redo();
```

4.3.3 *Execute CommentGraphNode related actions*

```
// create an add action to add the node to the given diagram
AddCommentGraphNodeAction action1 = new
AddCommentGraphNodeAction(commentGraphNode, diagram, modelManager);
action1.execute();

// undo the action
action1.undo();

// redo the action
action1.redo();

// create a remove action
RemoveCommentGraphNodeAction action2 =
      new RemoveCommentGraphNodeAction(commentGraphNode);

// execute the action to remove the node from the diagram
action2.execute();

// undo the action
action2.undo();

// redo the action
action2.redo();

// create a copy action
CopyCommentGraphNodeAction action3 =
      new CopyCommentGraphNodeAction(commentGraphNode, clipboard);
```

```
// copy to the given clipboard
action3.execute();

// copy to system clipboard
action3 = new CopyCommentGraphNodeAction(commentGraphNode, null);
action3.execute();

// create a cut action
CutCommentGraphNodeAction action4 =
         new CutCommentGraphNodeAction(commentGraphNode, clipboard);

// remove from the diagram and copy to the given clipboard
action4.execute();

// undo the action
action4.undo();

// redo the action
action4.redo();

// create a paste action to add the node to the given diagram
PasteCommentGraphNodeAction action5 =
       new PasteCommentGraphNodeAction(commentGraphNode, diagram);
action5.execute();

// undo the action
action5.undo();

// redo the action
action5.redo();
```

4.3.4 *Execute CommentRelationshipGraphEdge related actions*

```
// create an add action to add the edge to the given diagram
AddCommentRelationshipGraphEdgeAction action1 =
       new AddCommentRelationshipGraphEdgeAction(commentGraphEdge,
                    diagram, modelManager);
action1.execute();

// undo the action
action1.undo();

// redo the action
action1.redo();


// create a remove action
RemoveCommentRelationshipGraphEdgeAction action2 =
  new RemoveCommentRelationshipGraphEdgeAction(commentGraphEdge);

// execute the action to remove the edge from its diagram
```

```
action2.execute();

// undo the action
action2.undo();

// redo the action
action2.redo();

// create a copy action
CopyCommentRelationshipGraphEdgeAction action3 =
  new CopyCommentRelationshipGraphEdgeAction(commentGraphEdge, clipboard);

// copy to the given clipboard
action3.execute();

// copy to system clipboard
action3 =
   new CopyCommentRelationshipGraphEdgeAction(commentGraphEdge, null);
action3.execute();

// create a cut action
CutCommentRelationshipGraphEdgeAction action4 =
  new CutCommentRelationshipGraphEdgeAction(commentGraphEdge, clipboard);

// remove from its diagram and copy to the given clipboard
action4.execute();

// undo the action
action4.undo();

// redo the action
action4.redo();

// create a paste action to add the edge to the given diagram
PasteCommentRelationshipGraphEdgeAction action5 =
  new PasteCommentRelationshipGraphEdgeAction(commentGraphEdge, diagram);
action5.execute();

// undo the action
action5.undo();

// redo the action
action5.redo();
```

4.3.5 *Execute FreeTextGraphNode related actions*

```
// create an add action to add the node to the given diagram
AddFreeTextAction action1 =
    new AddFreeTextAction(freeTextGraphNode, diagram, modelManager);
action1.execute();

// undo the action
action1.undo();
```

```
// redo the action
action1.redo();


// create a remove action
RemoveFreeTextAction action2 =
   new RemoveFreeTextAction(freeTextGraphNode);

// execute the action to remove the node from its diagram
action2.execute();

// undo the action
action2.undo();

// redo the action
action2.redo();

// create a copy action
CopyFreeTextAction action3 = new CopyFreeTextAction(freeTextGraphNode,
clipboard);

// copy to the given clipboard
action3.execute();

// copy to system clipboard
action3 = new CopyFreeTextAction(freeTextGraphNode, null);
action3.execute();

// create a cut action
CutFreeTextAction action4 = new CutFreeTextAction(freeTextGraphNode,
clipboard);

// remove from its diagram and copy to the given clipboard
action4.execute();

// undo the action
action4.undo();

// redo the action
action4.redo();

// create a paste action to add the node to the given diagram
PasteFreeTextAction action5 =
    new PasteFreeTextAction(freeTextGraphNode, diagram);
action5.execute();

// undo the action
action5.undo();

// redo the action
action5.redo();
```

4.3.6 *Execute PolylineGraphNode related actions*

```
// create an add action to add the edge to the given diagram
AddPolylineAction action1 =
    new AddPolylineAction(polylineGraphEdge, diagram, modelManager);
action1.execute();


// undo the action
action1.undo();
```

```
// redo the action
action1.redo();


// create a remove action
RemovePolylineAction action2 =
       new RemovePolylineAction(polylineGraphEdge);

// execute the action to remove the edge from its diagram
action2.execute();

// undo the action
action2.undo();

// redo the action
action2.redo();

// create a copy action
CopyPolylineAction action3 =
       new CopyPolylineAction(polylineGraphEdge, clipboard);

// copy to the given clipboard
action3.execute();

// copy to system clipboard
action3 = new CopyPolylineAction(polylineGraphEdge, null);
action3.execute();

// create a cut action
CutPolylineAction action4 = new CutPolylineAction(polylineGraphEdge,
clipboard);

// remove from its diagram and copy to the given clipboard
action4.execute();

// undo the action
action4.undo();

// redo the action
action4.redo();

// create a paste action to add the edge to the given diagram
PastePolylineAction action5 =
       new PastePolylineAction(polylineGraphEdge, diagram);
action5.execute();

// undo the action
action5.undo();

// redo the action
action5.redo();
```

4.3.7 *Execute ChangeCommentTextAction*

```
// create action to change comment's text
ChangeCommentTextAction action =
       new ChangeCommentTextAction(comment, "new-text");
action.execute();

// undo to change the text back
action.undo();

// redo to make the change
action.redo();
```

4.3.8 *Select proper paste action by the application*

After the cut/copy action, when we click the paste button on the tool bar, the application should be able to select proper paste action to paste the object correctly.

```
Transferable contents = clipboard.getContents(null);
if (contents.isDataFlavorSupported(AuxiliaryElementDataFlavor.COMMENT)) {
    // get the pasted object
    Object data = contents.getTransferData(
                                AuxiliaryElementDataFlavor.COMMENT);
    PasteCommentAction action1 = new PasteCommentAction(
                                        data, modelManager);

    // execute the action to add data into the Model
    action1.execute();
} else if (contents.isDataFlavorSupported(
          AuxiliaryElementDataFlavor.NOTE_RELATIONSHIP)) {
     Object data = contents.getTransferData(
             AuxiliaryElementDataFlavor.NOTE_RELATIONSHIP);
     PasteNoteRelationshipAction action2 =
             new PasteNoteRelationshipAction(
                            data, modelManager);
     action2.execute();

} else if … -> do similarly to the other DataFlavor constants
                 defined in AuxiliaryElementDataFlavor class
 }
```

5 Future Enhancements

None.