UML Tool Actions - Diagram Actions 1.0 Component Specification

1. Design

The Diagram Actions component provides the Actions related to the Diagrams declared in Diagram Interchange component. The actions are strategy implementations of the action interfaces in the Action Manager component. The provided actions are for creating a diagram, removing a diagram, changing the diagram title, adjusting the zoom level and changing the diagram viewport.

This design provides implementations of undoable actions: create diagram actions, remove diagram actions and change diagram title action. Create diagram actions include create class diagram, sequence diagram, use case diagram and activity diagram actions. Remove diagram actions include remove class diagram, sequence diagram, use case diagram and activity diagram action.
And a log function for undoable actions is provided by this design. It will log the execute/undo/redo actions in INFO level and log the exceptions in ERROR level.

This design also provides implementations of transient actions: adjust diagram zoom level action and scroll diagram view action.

1.1 Design Patterns

- Template Pattern is used by UndoableAbstractAction to provide general undoable functionalities.
- Strategy Pattern is used for UndoableAction and TransientAction to be implemented as different undoable and transient actions.

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 Constructs CreateDiagramAction

```
Call super("Create "+type+" diagram " + name)
Create a new diagram with new DiagramImpl()
Set the diagram type (the typeInfo property of the Diagram) to type
Set the owner property of the Diagram to the owner. If owner is null,
the root Model(get by UMLModelManager) will be the owner.
To set the owner of a Diagram should:
1. create a modelBridge by new UML1SemanticModelBridge()
2. set the element of the modelBridge by modelBridge.setElement(owner)
3. set the owner of the Diagram by diagram.setOwner(modelBridge )
Set the diagram name (the name attribute of the Diagram) to the name
Set the viewport, size and position of the diagrams to (0.0,0.0).
Set the zoom attribute of the diagrams to "1.0".
Get the ProjectConfigurationManager and the project language from the
UMLModelManager, apply initial formatting for the diagram with the
project language and ProjectConfigurationManager.
Save diagram to the like named variable
```

1.3.2 Constructs CreateActivityDiagramAction

```
Call super("Create activity diagram" + name)
Create an ActivityGraph with new ActivityGraphImpl()
If the owner is not null, set the context property of the ActivityGraph
to the owner,
Else set the context to a new use case element added directly to the
Model.
Create an empty CompositeState and set the "top" attribute of the
ActivityGraph to it.
Save the activityGraph to the variable
Create the helpAction by new
CreateActivityDiagramHelpAction(activityGraph, name) and save to the
variable
```

1.4 Component Class Overview

UndoableAbstractAction:

This class is an implementation of UndoableAction. It applies the template pattern to provide undoable functionalities for its subclass. And it will log the execute/undo/redo actions in INFO level and log the exceptions in ERROR level.

ActionState:

This class is an enumeration which representing the state of the undoable action.

CreateDiagramAction:

This class is a subclass of UndoableAbstractAction. This action will support creating a class diagram, a use case diagram, a sequence diagram or an activity diagram.

CreateClassDiagramAction:

This class is a subclass of CreateDiagramAction. The action will create a class diagram with no contained elements.

CreateUseCaseDiagramAction:

This class is a subclass of CreateDiagramAction. The action will create a use case diagram with no contained elements.

CreateSequenceDiagramAction:

This class is a subclass of CreateDiagramAction. The action will create a sequence diagram with no contained elements.

CreateActivityDiagramHelperAction:

This class is a subclass of CreateDiagramAction. This action will help CreateActivityDiagramAction to create an activity diagram.

CreateActivityDiagramAction:

This class is a subclass of UndoableAbstractAction. This action will support creating an activity graph diagram.

RemoveDiagramAction:

This class is a subclass of UndoableAbstractAction. This action will support removing a class diagram, a use case diagram, a sequence diagram or an activity diagram.

RemoveClassDiagramAction:

This class is a subclass of RemoveDiagramAction. The action will simply remove the class diagram.

RemoveUseCaseDiagramAction:
This class is a subclass of RemoveDiagramAction. The action will simply remove the use case diagram.

RemoveSequenceDiagramAction:
This class is a subclass of RemoveDiagramAction. The action will simply remove the sequence diagram.

RemoveActivityDiagramAction:
This class is a subclass of UndoableAbstractAction. This action will support removing an activity graph diagram.

ChangeDiagramTitleAction:
This class is a subclass of UndoableAbstractAction. This action will change the title of a diagram.

AdjustDiagramZoomLevelAction:
This class is an implementation of TransientAction. This action will change the zoom attribute of a diagram.

ScrollDiagramViewAction:
This class is an implementation of TransientAction. This action will change the viewport property of a diagram.

1.5 Component Exception Definitions

UndoableActionException:
This exception will be created and thrown by all the undoAction and redoAction methods in subclasses of UndoableAbstractAction when the redo/undo action can't complete successfully. This exception will be exposed to the UndoableAbstractAction's redo/undo methods.

1.6 Thread Safety

This component is not thread-safe.

This component is considered to be used in single thread environment. The undoable actions are not thread safe by containing mutable state information. If an application using this component does intend to change the arguments in separate threads, then it should ensure the redo, undo and execute methods synchronized externally.

The transient actions are thread safe by being immutable.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- Action Manager 1.0
- Diagram Interchange 1.0
- UML Model - Core 1.0

- UML Model - Activity Graphs 1.0
- UML Model Manager 1.0
- UML Project Configuration 1.0
- Logging Wrapper 1.2

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.actions.diagram

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

Please review the documentation for the Logging Wrapper components for specifics on its configuration needs.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration <C:\filez\work\tc\tcs_corp\templates\component\>.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Nothing special required

4.3 Demo

4.3.1 Simple Demo: Create Create/Remove Diagram Actions

```
// Assume all the Diagrams is valid and added to the UMLModelManager
instance
// If create/remove action is triggered by the user of UML Model Tool, the
action will be created to be handled
// Create a ChangeEntityNameAction, which will create and add a new class diagram
named "New Class Diagram"
UndoableAction action = new CreateClassDiagramAction(null, "New Class
Diagram");

// Create a CreateUseCaseDiagramAction, which will create and add a new use
case diagram named
// "New Use Case Diagram"
action = new CreateUseCaseDiagramAction(null, "New Use Case Diagram");

// Create a CreateSequenceDiagramAction, which will create and add a new sequence
diagram named
// "New Sequence Diagram"
```

```
action = new CreateSequenceDiagramAction(null, "New Sequence Diagram");

// Create a CreateActivityDiagramAction, which will create and add a new activity
diagram named
// "New Activity Diagram"
action = new CreateActivityDiagramAction(null, "New Activity Diagram");

// Create a RemoveClassDiagramAction, which will remove the class diagram
classDiagram
Diagram classDiagram = new Diagram();
action = new RemoveClassDiagramAction(classDiagram);

// Create a RemoveUseCaseDiagramAction, which will remove the use case diagram
useCaseDiagram
Diagram useCaseDiagram = new Diagram();
action = new RemoveUseCaseDiagramAction(useCaseDiagram);

// Create a RemoveSequenceDiagramAction, which will remove the sequence diagram
sequenceDiagram
Diagram sequenceDiagram = new Diagram();
action = new RemoveSequenceDiagramAction(sequenceDiagram);

Diagram activityDiagram = new Diagram();
activityDiagram.setName("Name");
Uml1SemanticModelBridge uml1SemanticModelBridge = new
Uml1SemanticModelBridge();
uml1SemanticModelBridge.setElement(new ActivityGraphImpl());
activityDiagram.setOwner(uml1SemanticModelBridge);
// Create a RemoveActivityDiagramAction, which will remove the activity diagram
activityDiagram
action = new RemoveActivityDiagramAction(activityDiagram);
action.die();
```

4.3.2 *Simple Demo: Handle Action*

```
UndoableAction action = new CreateClassDiagramAction(null, "New Class
Diagram");

// If execute an action is triggered by the user of UML Model Tool, it will
be executed after created.
// Execute the action
action.execute();

// If undo an action is triggered by the user of UML Model Tool, it will be
undone
action.undo();

// If redo an action is triggered by the user of UML Model Tool, it will be
redone
action.redo();

// If redo an action when it is already redone, it is invalid
action.redo();

// The log is taken CreateClassDiagramAction as example
// The log should be:
// Create Class diagram New Class Diagram
// Undo Create Class diagram New Class Diagram
// Redo Create Class diagram New Class Diagram
// Can't Redo Create Class diagram New Class Diagram. Current state is UNDOABLE.
```

4.3.3 Simple Demo of Transient Actions

```
Diagram diagram = new Diagram();
// Create an adjust diagram zoom level action which will change zoom level
of the diagram to 200%
AdjustDiagramZoomLevelAction adjustDiagramZoomLevelAction = new
AdjustDiagramZoomLevelAction(diagram, 2.0);
// Execute the action
adjustDiagramZoomLevelAction.execute();
// check the zoom level
assertEquals("The zoom level of the diagram is incorrect.", 0,
Double.compare(2.0, diagram.getZoom()));

// Create a new view port (10, 20)
Point viewport = new Point();
viewport.setX(10);
viewport.setY(20);
// Create a scroll diagram view action which will change view port of the diagram
to (10,20)
ScrollDiagramViewAction scrollDiagramViewAction = new
ScrollDiagramViewAction(diagram, viewport);
// Execute the action
scrollDiagramViewAction.execute();
// check the view port
assertSame("The view port of the diagram is incorrect.", viewport,
diagram.getViewport());
```

5. Future Enhancements

None