Action Manager 1.0 Requirements Specification

1. Scope

1.1 Overview

The Action Manager component provides a general framework for executing actions. It also provides the undo/redo actions framework.

1.2 Logic Requirements

The component provides the one place to be accessed when executing actions inside an application. All the GUI significant changes that affect the model of the application must be executed through this component. It will keep track of the undo / redo actions, also resetting the undo / redo manager if a non undoable action is executed, or leaving the undo / redo manager's state as is if a transient action is executed.

1.2.1 ActionManager

The manager will be responsible for executing simple actions, transient actions, undoable actions and undo / redo actions.

The manager will maintain the undo / redo state (the undoable actions) in a javax.swing.undo.UndoManager.

It provides methods to execute actions, to redo / undo actions that exist in the undo / redo manager and methods to query about the actions to be undone / redone.

1.2.2 Action interface

The action has an execute method. Actions that are not transient or undoable will be executed and the undo/redo manager will be reset.

The execute() method will throw an ActionExecutionException, in case there is a problem.

Examples of Action: load a new project, close the existing project.

1.2.3 TransientAction interface

This interface is a markup interface for the actions that should not affect the state of the undo/redo manager. It extends the Action interface.

Examples of TransientAction: print a diagram, copy an element to the clipboard, save the project.

1.2.4 UndoableAction

This interface is an action that also inherits from the javax.swing.undo.UndoableEdit interface. These actions will be executed and will be added to the undo manager.

The undo() and redo() methods will not throw any exception, according to their signatures, but the concrete classes should provide error handling either through logging, or by storing the last exception, or...

Examples of UndoableActions: add a class to the diagram move an element on the diagram, remove an element.

1.2.5 Adapters

1.2.5.1 CompoundUndoableAction

This class adapts the javax.swing.undo.CompoundEdit to be used as an UndoableAction.

1.2.5.2 TransientUndoableAction

This class adapts a TransientAction, to be used as a non-significant UndoableAction. Its purpose is to allow the TransientAction to be used inside a CompoundUndoableAction. The adapter's canUndo() method will return true, and the undo() method will do nothing (it will not throw exceptions).

Example of usage: the cut undoable action is composed by a transient copy action and an undoable remove action. In order to be able to create a single undoable action for the cut action, and avoid creating a new class, the action will be a compound undoable action, containing the non-significant undoable copy action and the undoable remove action.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool as the central place for executing actions triggered by the GUI.

1.5 Future Component Direction

None at this moment.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The design must follow the interfaces found in the class diagram with the component interfaces. The designer is encouraged to add to the existing interfaces, 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.util.actionmanager

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The undoable actions limit - the maximum number of edits the UndoManager will hold.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

- javax.swing.undo

3.2.2 TopCoder Software Component Dependencies:

- Configuration Manager 2.1.5

**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.