UML Tool Actions - Class Elements Actions 1.0 Component Specification

1. Design

The Class Elements Actions component provides the Actions related to the model elements specific to a class diagram. 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 package, interface, class, exception and enumeration. The relationships are association, aggregation, composition, realization/abstraction and dependency.

All classes of this components belongs to one package but it representation has 11 class diagrams. It was made for more clear showing classes and relationships.

Class provide sequence diagram for all Add and Copy actions of one model element – Class. The other actions very similar or more simple – Paste very more simple than add because it don't provide applying initial configuration, Remove –do not contain 5 objects and Cut are very similar to Copy. Logic of actions for the rest elements is the same.

Component use logger to notify about problem until running redo() and undo() methods. They should not throw some exception according requirements of this component.

1.1 Design Patterns

None

1.2 Industry Standards

UML 1.5

1.3 Required Algorithms

1 Calculating namespace of relationships

All of model elements should have some namespace. Only model as a root element and all elements which are directly in the model has no namespace (Actually their namespace is null).

Relationships do not get namespace as a parameter. They should calculate it. Each of relationship contains references for model elements which contain it. At least it should be two model elements but may be more.

Calculated relationship namespace should contain all model elements which are owners of this relationship. Namespace for classes are actually packages.

- If at least of model elements belongs directly to the model – relationship namespace belongs to the model(namespace = null)
- If all components belongs to different namespaces which has no own base namespace – relationship belongs to the model (example: com.topcocer.test and org.organization.test)
- If they all have base namespace – relationship belong to this namespace (example: com.topcoder.test1 and com.topcoder.test2; result namespace – com.topcoder).

Note: in examples above you should understand that for example com.topcoder – it just description of namespace. It can not be compared like string. For getting such structure you need use getNamespace() method and compare Namespace instances using equal() method. For each model element it will be list of namespace reference. If list is empty thatis mean element is directly in model.

2. Component should provide correct copying of elements. There are two different way to copy elements – by creating new object and by copying reference. The simplest way to choose correct way of copying is used relationships between different elements. Note that if it will be list or collection you should just copy each element of this.

1. Simple attributes and Strings should be copied by values. Example (from ModelElement) – name;

```
copyModelElement.setName(originalModelElement.getName());
```

2. Aggregation. It should be used copying only by reference. Example (from ModelElement) – templateArguments;

```
for (TemplateArgument temp:
originalModelElement.getTemplateArguments()){

     copyModelElement.addTemplateArgument(temp);

 }
```

3. Bidirectional aggregation. The same algorithm as for aggregation, but you need also update element which is used in aggregation by adding to it attribute (which is used for aggregation) this instance of ModelElement. Example (from ModelElement) – stereotypes;

```
for (Stereotype temp:
originalModelElement.getTemplateStereotypes()){

     copyModelElement.addTemplateArgument(temp);

     temp.addExtendedElements(copyModelElement);

 }
```

4. Composition. It should be used creating new object and copying its attributes. (from ModelElement) – taggedValues; Note that it also present aggregation here

```
      for (TaggedValue temp:
originalModelElement.getTaggedValues()){

            TaggedValue newTag = new TaggedValue();

            // copying attrinbutes from temp to newTag as was
described in this rules

            copyModelElement.addTemplateArgument(newTag);

            newTag.setModelElement(copyModelElement);

       }
```

Note that when you use deep copying for composition, you should use these four rules.

Using this rules help developer easy and correct copy each concrete element. The list of element which should be copied is described for each element separately.

1.4 Component Class Overview

Class ClassElementsTransfer
This class is used for transporting model elements through a system clipboard. It contain 9 custom DataFlavor instances – each element has own DataFlavor. Class also implement interface ClipboardOwner for providing ability to be owner for elements in clipboard. It has eight public constructors for setting different types of data flavors without using instanceof operator.

Class is thread safety because it is immutable.

Base Actions
This abstract class extends from *AbstractUndoableEdit* and implements *UndoableAction* interface. This class is base for all rest action classes in component. It contains who attributes and their protected getter. Attributes represent all possible instances of ModelElement for this component and their utility classes.

Class contains reference to some model element which is mutable and also it extends from mutable super class. That is why it is not thread safety.

Class AddAction

This abstract class extends from ClassUndoableAction. It implement all logic of add action for all elements in component. It contains three methods – execute(), redo(), undo(). Also this class is responsible for applying initial formatting of elements.

Class is not thread safety because it extends from mutable class.

Class PasteAction
This abstract class extends from ClassUndoableAction. It implements all logic of paste action for all elements in component. It contains three methods – execute(), redo(), undo().

Class is not thread safety because it extends from mutable class.

Class CopyAction
This abstract class provides all logic for Copy actions for all elements and contains all attributes which are common for them. It realize TransientAction interface. Provide copying modelElement to clipboard

Class contains reference to some model element which is mutable and that is why it is mutable and is not thread safety too.

Class RemoveAction
This abstract class extends from ClassUndoableAction. It implements all logic of remove action for all elements in component. It contains three methods – execute(), redo(), undo().

Class is not thread safety because it extends from mutable class.

Class CutAction
This abstract class extends from ClassUndoableAction. It implements all logic of cut action for all elements in component. It contains three methods – execute(), redo(), undo().

Class is not thread safety because it extends from mutable class.

Class UsecaseToolUtil
This abstract class provides similar simple operations which are the same for all actions classes. Its children provides unique logic for each element

Class is thread safety because it is immutable.

Class Actions
Class AddClassAction
This class extends from AddAction and used for adding Class instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteClassAction
This class extends from PasteAction and used for adding Class instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyClassAction
This class extends from CopyAction and used for copying Class instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveClassAction
This c1ass extends from ClassAction and provide functionality to remove Class instance from model or namespace. Also it has ability to use redo and undo of removing

Class is thread safety because it is immutable.

Class CutClassAction
This class extends from CutAction and used for cutting (copy + remove) Class instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class ClassUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Class.

Class is thread safety because it is immutable.

Exception Actions
Class AddExceptionAction
This class extends from AddAction and used for adding Class instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class. Also it provide checking if given Class instance is exception

Class is not thread safety because it extends from mutable class.

Class PasteExceptionAction
This class extends from PasteAction and used for adding Class instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyExceptionAction
This class extends from CopyAction and used for copying Class instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class. Also it provide checking if given Class instance is exception

Class is not thread safety because it extends from mutable class.

Class RemoveExceptionAction
This class extends from RemoveAction and used for removing Class instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class. Also it provide checking if given Class instance is exception

Class is not thread safety because it extends from mutable class.

Class CutExceptionAction
This class extends from CutAction and used for cutting (copy + remove) Class instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class. Also it provide checking if given Class instance is exception

Class is not thread safety because it extends from mutable class.

Class ExceptionUtil
This class extends from ClassUtil. It just contain one own method – to check if Class instance has “exception” stereotype

Class is thread safety because it is immutable.

Interface Actions
Class AddInterfaceAction
This class extends from AddAction and used for adding Interface instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteInterfaceAction
This class extends from PasteAction and used for adding Interface instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyInterfaceAction

This class extends from CopyAction and used for copying Interface instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveInterfaceAction
This class extends from RemoveAction and used for removing Interface instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutInterfaceAction
This class extends from CutAction and used for cutting (copy + remove) Interface instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class InterfaceUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Interface.

Class is thread safety because it is immutable.

Enumeration Actions
Class AddEnumerationAction
This class extends from AddAction and used for adding Enumeration instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteEnumerationAction
This class extends from PasteAction and used for adding Enumeration instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyEnumerationAction
This class extends from CopyAction and used for copying Enumeration instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveEnumerationAction

This class extends from RemoveAction and used for removing Enumeration instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutEnumerationAction
This class extends from CutAction and used for cutting (copy + remove) Enumeration instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class EnumerationUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Enumeration.

Class is thread safety because it is immutable.

Package Actions
Class AddPackageAction
This class extends from AddAction and used for adding Package instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PastePackageAction
This class extends from PasteAction and used for adding Package instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyPackageAction
This class extends from CopyAction and used for copying Package instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemovePackageAction
This class extends from RemoveAction and used for removing Package instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutPackageAction
This class extends from CutAction and used for cutting (copy + remove) Package instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PackageUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Package

Class is thread safety because it is immutable.

Abstraction Actions
Class AddAbstractionAction
This class extends from AddAction and used for adding Abstraction instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteAbstractionAction
This class extends from PasteAction and used for adding Abstraction instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyAbstractionAction
This class extends from CopyAction and used for copying Abstraction instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveAbstractionAction
This class extends from RemoveAction and used for removing Abstraction instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutAbstractionAction
This class extends from CutAction and used for cutting (copy + remove) Abstraction instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class AbstractionUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Abstraction.

Class is thread safety because it is immutable.

Association Actions
Class AddAssociationAction
This class extends from AddAction and used for adding Association instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteAssociationAction
This class extends from PasteAction and used for adding Association instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyAssociationAction
This class extends from CopyAction and used for copying Association instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveAssociationAction
This class extends from RemoveAction and used for removing Association instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutAssociationAction
This class extends from CutAction and used for cutting (copy + remove) Association instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class AssociationUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Association.

Class is thread safety because it is immutable.

Dependency Actions

Class AddDependencyAction

This class extends from AddAction and used for adding Dependency instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteDependencyAction

This class extends from PasteAction and used for adding Dependency instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyDependencyAction

This class extends from CopyAction and used for copying Dependency instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveDependencyAction

This class extends from RemoveAction and used for removing Dependency instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutDependencyAction

This class extends from CutAction and used for cutting (copy + remove) Dependency instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class DependencyUtil

This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Dependency.

Class is thread safety because it is immutable.

Generalization Actions

Class AddGeneralizationAction

This class extends from AddAction and used for adding Generalization instance to model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class PasteGeneralizationAction
This class extends from PasteAction and used for adding Generalization instance to model or namespace from clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CopyGeneralizationAction
This class extends from CopyAction and used for copying Generalization instance to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class RemoveGeneralizationAction
This class extends from RemoveAction and used for removing Generalization instance from model or namespace. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class CutGeneralizationAction
This class extends from CutAction and used for cutting (copy + remove) Generalization instance from model or namespace to clipboard. Its methods provide only proper configuration and presentation of action. All logic represent into its super class.

Class is not thread safety because it extends from mutable class.

Class GeneralizationUtil
This class extends from ClassToolUtil. It overrides some method which is unique for current instance of ModelElement – Generalization.

Class is thread safety because it is immutable.

1.5 Component Exception Definitions

Component may throw IllegalArgumentEception when some method was given null parameter (check documentation of each concrete methods) and ActionExecutionException when appears problem during running execute() method of actions.
Also ClassElementsTransfer may throw own exception see documentation of this class.

Class InvalidDataContentException
This exception may be thrown in two situations – when data in transferable object is incorrect or when we use incorrect utility class for current element Also it may be thrown by each constructor of exception actions, if given Class instance has no “exception” stereotype

Class is thread safety because it is immutable.

1.6 Thread Safety

Component is not thread safety because many of its classes are not thread safety. Actually this is not needed because UML Tool provides external thread safety model.

2. Environment Requirements

2.1 Environment

Java 1.5 for test and compile component

2.2 TopCoder Software Components

Action Manager 1.0

It is the base component for creating actions. It contains interfaces which should be implemented.

UML Model Manager 1.0

This component gives ability to get Model and project language.

UML Project Configuration 1.0

This component provide applying of initial formatting for model elements

UML Model - Core Classifiers 1.0

This component represents Class, Enumeration and Interface interfaces and also their default implementation

UML Model - Core Relationships 1.0

This component represents Association and Generalization interfaces and also their default implementation

UML Model - Core Dependencies 1.0

This component represents Abstraction and Dependency interfaces and also their default implementation

UML Model - Core Requirements 1.0

This component represents Namespace interface and its default implementation. Also this component is needed to find all attributes of model elements

UML Model - Model Management 1.0

This component represents Model and Package interfaces and also their default implementation

UML Model - Core Extension Mechanisms 1.0

This component is used for giving stereotype of class. It is needed for checking if class is really exception

Base Exception 1.0

This component is basic of all custom exceptions in this component.

Logging Wrapper 1.2

This component uses by redo() and undo() methods of actions for logging errors which may occur in execution process. Also it is used by ClassElementsTransfer class.

Typesafe Enum 1.0

This component is needed by component Logging Wrapper. For more detail see its component specification

Configuration Manager 2.1.5

This component is needed by component Logging Wrapper. For more detail see its component specification

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.actions.model.classifiers

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

Logging Wrapper 1.2 is needed to be configured. For more detail see its component specification

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

There are no some special requirements for running this component

4.3 Demo

This demo will be provided just for Class actions. Working with all other model elements are practically the same. This component can not have own scenario of running it used by the other component which define its behavior. It just simple example when user at first add component class, then press undo, then press redo, then copy it and paste into new namespace. Now it has two classes – he delete actor from namespace and then cut and paste into namespace class from model (actually default namespace). The next code shows how to realize such user actions:

```
// let’ s create first add action without namespace
// Note that it was just creation and actor don’ t added to model
AddClassAction addClass = new AddClassAction(classElement, manager);

// After execution classElement was add to model
addClass.execute();

// Now we can use undo and redo actions
// As a result classElement was deleted from model
addClass.undo();

// As a result classElement was added to model again
addClass.redo();

// Now let copy actor to namespace, for this target we should create
// copy action and execute it.
CopyClassAction copyClass = new CopyClassAction(classElement, clipboard);
copyClass.execute();

/*
 * Now actor is in clipboard and we need to paste it in needed
 * namespace. Paste action has such requirement that it can’t get class
 * from clipboard directly. It just get Transferable instance and only
 * than put class to model(it wait for some action from graphical part).
 * For now image that user click paste and we get transferable from
 * clipboard. For example, user may click copy but than don’t click
 * paste. He can copy some next model element that is why impossible
 * directly connect copy and paste action. Application may use
 * ClassElementsTransfer to check if clipboard contain right element For
 * now image that user click paste and we get transferable from
 * clipboard Component may define which paste action should be created
 * by using methods of transferable
 */
Transferable trans = clipboard.getContents(null);
if (trans.isDataFlavorSupported(ClassElementsTransfer.CLASS_FLAVOR)) {
   PasteClassAction pasteClass = new PasteClassAction(trans, namespace);
   pasteClass.execute();
} else {
   // Provide some exception logic
}

// As a result classElement was deleted from model
addClass.undo();
```

```
// As a result classElement was added to model again
addClass.redo();

/*
 * For now we actually create second classElement let it be:
 * classElement1. Component may just receive it for making some action.
 * It doesn’t have reference to it Then let provide deleting cutting and
 * pasting classElement as was mentioned in scenario. Note now we have
 * two classes
 */
RemoveClassAction removeClass = new RemoveClassAction(classElement1);

// we just deletes classElement1.
removeClass.execute();

CutClassAction cutClass = new CutClassAction(classElement);

// the classElement was cut from the namespace.
cutClass.execute();
// For the addClass lost the owner of the clipboard, a warn log will be
// given: Clipboard System contains Class

// For classElement was already cut from the namespace, a warn log will
// be given: Given element doesn't exist in the namespace.
addClass.undo();

// As a result classElement was added to model again
addClass.redo();

PasteClassAction pasteClass = new PasteClassAction(trans, namespace);

// the classElement was added to the namespace again.
pasteClass.execute();
// For the addClass lost the owner of the clipboard, a warn log will be
// given: Clipboard System contains Class

// As a result classElement was deleted from model
addClass.undo();

// As a result classElement was added to model again
addClass.redo();

// after this operation we has only classElement and it is in given
 // namespace
```

5. Future Enhancements

None