UML Tool Actions - Use Case Elements Actions Version 1.0 Component Specification

1. Design

The Use Case Elements Actions component provides the Actions related to the model elements specific to a use case 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 actor, subsystem and use case. The relationships are extended and include. The other elements and relations that might be added to a use case diagram are declared in the Class Elements Actions component

There are two types of actions in the component – *UndoableAction*(provide ability to run undo and redo) and *TransientAction*(provide simple execute method). Only CopyXXXActions are transient.

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

Component does not provide some special algorithm. Classes' methods contain a lot of code example. This part of specification contains algorithm for calculating namespace for relationships and explanation about deep copying.

1. 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 subsystems.

- 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: subsstem1.subsystem2 and subsystem3.subsystem4)
- If they all have base namespace – relationship belong to this namespace (example: subsstem1.subsystem2 and subsstem1.subsystem3; result namespace – subsstem1).

Note: in examples above you should understand that for example subsystem – 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 that is 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 ModelTransfer

This class is used for transporting model elements through a system clipboard. It contain 5 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 five public constructors for setting different types of data flavors without using instanceof operator.

Class is thread safety because it is immutable.

Class UsecaseUndoableAction

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 UsecaseUndoableAction. 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 UsecaseUndoableAction. 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 UsecaseUndoableAction. 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 UsecaseUndoableAction. 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.

Actor Actions
Class AddActorAction
This class extends from AddAction and used for adding Actor 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 PasteActorAction
This class extends from PasteAction and used for adding Actor 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 CopyActorAction
This class extends from CopyAction and used for copying Actor 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 RemoveActorAction
This class extends from RemoveAction and used for removing Actor 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 CutActorAction
This class extends from CutAction and used for cutting (copy + remove) Actor 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 ActorUtil
This class extends from UsecaseToolUtil. It overrides some method which is unique for current instance of ModelElement – Actor.

Class is thread safety because it is immutable.

UseCase Actions
Class AddUseCaseAction
This class extends from AddAction and used for adding UseCase 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 PasteUseCaseAction
This class extends from PasteAction and used for adding UseCase 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 CopyUseCaseAction
This class extends from CopyAction and used for copying UseCase 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 RemoveUseCaseAction
This class extends from RemoveAction and used for removing UseCase 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 CutUseCaseAction
This class extends from CutAction and used for cutting (copy + remove) UseCase 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 UseCaseUtil
This class extends from UsecaseToolUtil. It overrides some method which is unique for current instance of ModelElement – UseCase.

Class is thread safety because it is immutable.

Subsystem Actions
Class AddSubsystemAction
This class extends from AddAction and used for adding Subsystem 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 PasteSubsystemAction
This class extends from PasteAction and used for adding Subsystem 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 CopySubsystemAction
This class extends from CopyAction and used for copying Subsystem 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 RemoveSubsystemAction
This class extends from RemoveAction and used for removing Subsystem 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 CutSubsystemAction
This class extends from CutAction and used for cutting (copy + remove) Subsystem 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 SubsystemUtil
This class extends from UsecaseToolUtil. It overrides some method which is unique for current instance of ModelElement – Subsystem.

Class is thread safety because it is immutable.

Include Actions
Class AddIncludeAction
This class extends from AddAction and used for adding Include 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 PasteIncludeAction
This class extends from PasteAction and used for adding Include 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 CopyIncludeAction
This class extends from CopyAction and used for copying Include 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 RemoveIncludeAction
This class extends from RemoveAction and used for removing Include 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 CutIncludeAction
This class extends from CutAction and used for cutting (copy + remove) Include 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 IncludeUtil
This class extends from UsecaseToolUtil. It overrides some method which is unique for current instance of ModelElement – Include.

Class is thread safety because it is immutable.

Extend Actions
Class AddExtendAction
This class extends from AddAction and used for adding Extend 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 PasteExtendAction
This class extends from PasteAction and used for adding Extend 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 CopyExtendAction
This class extends from CopyAction and used for copying Extend 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 RemoveExtendAction
This class extends from RemoveAction and used for removing Extend 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 CutExtendAction
This class extends from CutAction and used for cutting (copy + remove) Extend 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 ExtendUtil
This class extends from UsecaseToolUtil. It overrides some method which is unique for current instance of ModelElement – Extend.

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 ModelTransfer 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

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 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 - Use Cases 1.0

This component represents Actor, UseCase, Include and Extend interfaces and also their default implementation

UML Model - Model Management 1.0

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

Base Exception 1.0

This component is basic of all exceptions in this component.

Logging Wrapper 1.3

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

UML Model - Core Relationships 1.0

Interface of this component (Relationship) is parent for Include and Extend. It is needed to find all attributes of model elements

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

2.2 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.actions.model.usecase

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

Logging Wrapper 1.3 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

None

4.3 Demo

This demo will be provided just for Actor 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 actor, then press undo, then press redo, then copy it and paste into new namespace. Now it has two actors – he delete actor from namespace and then cut and paste into namespace actor from model (actually default namespace). The next code shows how to realize such user actions:

```
/*Actor and Namespace instances will be received from
graphical part of UML Tool. For this demo image that we
already have Actors instance – actor, and one Namespace –
namespace; also we have default UMLModelManager - manager */

// let's create first add action without namespace

AddActorAction addActor = new AddActorAction(actor, manager);

//Note that it was just creation and actor don't added to
model

addActor.execute();

//After execution actor was add to model

//Now we can use undo and redo actions

addActor.undo();

//As a result actor was deleted from model

addActor.redo();

//As a result actor was added to model again

/*Now let copy actor to namespace, for this target we should
create copy action and execute it. */

CopyActorAction copyActor = new CopyActorAction(actor);

copyActor.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
actor from clipboard directly. It just get Transferable
instance and only than put actor to model (it wait for some
action from graphical part). 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(ModelTransfer.ACTOR_FLAVOR)) {

PasteActorAction pasteActor = new
PasteActorAction(transferable, namespace);

pasteActor.execute();}

else{
```

```
            //Provide some exception logic
        }


        /*for now we actually create second actor let it be – actor1.
        Component may just receive it for making some action. It
        doesn't have reference to it
        Then let provide deleting cutting and pasting actor as was
        mentioned in scenario. Note now we have two actors*/
        RemoveActorAction removeActor = new RemoveActorAction(actor1);
        removeActor.execute();
        //we just deletes actor1.
        CutActorAction cutActor = new CutActorAction(actor);
        cutActor.execute();
        pasteActor = new PasteActorAction(transferable,namespace);
        pasteActor.execute();
        // After this operation we has only actor and it is in given
namespace
```

5. Future Enhancements

None