Diagram Elements Add Actions 1.0 Component Specification

1. Design

Diagram Elements Add Actions component provides the Actions related to the concrete diagrams elements – add and update. The actions are strategy implementation of the action interface in the Action Manager component.

The component provides a set of concrete implementation of Action interface for each distinct action (independent on which object it is applied). Each action is undoable and redoable.

In addition, the component defines a set of compartment handler of each type of GraphElement object defined in the specification. This component does not perform compartment building at all. In addition, the component will only perform minimal validation to check whether the given GraphElement is properly configured. The component might throw exception during update/add actions if the related compartment is missing/malformed.

1.1 Design Patterns

Strategy Pattern: strategy pattern is used in this component to provide independent and pluggable implementation of compartment extractor. The implementation of compartment extractor is specific to each type of object. In addition, the implementation of action interface also employs strategy pattern. One variation of strategy pattern which utilizes interface as well as abstract implementation is used.

Factory Method: factory method is used for the client to create appropriate compartment extractor object given its type. One variation of factory pattern is used which is the parameterized constructor.

Command Pattern: command pattern is used to represent each action in the component. Thus each object can encapsulate the actions required as well as undo and redo functionality.

1.2 Industry Standards

Diagram Interchange 2.0

UML Model 1.5

1.3 Required Algorithms

1.3.1 Compartment Extraction algorithm.

This algorithm is provided as an example on how the compartment is structured. And thus, how it should be extracted. Developers are free to come up with their own algorithm to perform compartment extraction.

In this example, we are working with a GraphNode that corresponds to an Interface, as follow:

```
Uml1SemanticModelBridge.element = <UML:Interface>
            SimpleSemanticModelElement.typeInfo = "NameCompartment"
                SimpleSemanticModelElement.typeInfo = "StereotypeCompartment"
                SimpleSemanticModelElement.typeInfo = "Name"
                SimpleSemanticModelElement.typeInfo = "NamespaceCompartment”
            SimpleSemanticModelElement.typeInfo = "CompartmentSeparator"
            SimpleSemanticModelElement.typeInfo = "AttributeCompartment”
                SimpleSemanticModelElement.typeInfo = "DelimitedSection"
                        Uml1SemanticModelBridge.element = <UML:Attribute>
```

The meaning of such hierarchy can be described as follow:

There is a GraphNode N in which it has semanticModel attribute set to an instance of Uml1SemanticModelBridge in which the element attribute is set to an instance of UML:Interface.

The GraphNode N will have the contained attribute containing 3 GraphNode elements corresponds to “NameCompartment”,”CompartmentSeparator” and “AttributeCompartment”. Each of those GraphNode will have similar structure.

To give a clear picture, information below corresponds to the hierarchy presented above showing the attributes of each object and their types.

```
GraphElement N;
        N.semanticModel = P
                Uml1SemanticModelBridge P;
                        P.element = Q;
                                UML:Interface Q;
        N.contained = {A,B,C}   -> ordered list
                GraphElement A;
                        A.semanticModel = X;
                                SimpleSemanticModelElement X;
                                        X.typeInfo = “NameCompartment”
                        A.contained = {D,E,F}
                                GraphElement D
                                        D.semanticModel = D2
                                                SimpleSemanticModelElement D2
                                                        D2.typeInfo=“StereoTypeCompartment”
                                GraphElement E
                                        E.semanticModel = E2
                                                SimpleSemanticModelElement E2
                        E2.typeInfo=“Name”
            GraphElement F
                F.semanticModel = F2
                    SimpleSemanticModelElement F2
                        F2.typeInfo=“NamespaceCompartment”
    GraphElement B
        B.semanticModel = B2
            SimpleSemanticModelElement B2
                B2.typeInfo="CompartmentSeparator"
    GraphElement C
        C.semanticModel = C2
            SimpleSemanticModelElement C2
                C2.typeInfo=”AttributeCompartment”
        C.contained = {C3}
        GraphElement C3
            C3.semanticModel = C4
                SimpleSemanticModel C4
                    C4.typeInfo=”DelimitedSection”
            C3.contained = {C5}
                GraphElement C5
                    C5.semanticModel = C6
                        Uml1SemanticModelBridge C6
                            C6.element=UML:Attribute
```

Thus to extract the name compartment, can be done in the following:

```
    A=N.getElements().get(0)
    X=A.getSemanticModel()
    Check if X.getTypeInfo()==”NameCompartment”
    If Not, throw CompartmentInvalidException
    E=A.getElements().get(1)
    E2 = A.getSemanticModel()
    Check if E2.getTypeInfo()==”Name”
    If Not, throw CompartmentInvalidException
    Return E
```

To update the name compartment can be done in the following:

```
A=N.getElements().get(0)
X=A.getSemanticModel()
Check if X.getTypeInfo()==”NameCompartment”
If Not, throw CompartmentInvalidException
E=A.getElements().get(1)
E2 = A.getSemanticModel()
Check if E2.getTypeInfo()==”Name “
A.setElement(1,newName)
```

1.3.2 Mapping of object types to compartment

Table below lists the mapping between object type and the appropriate compartment extractor implementation.

Object Type	Compartment Extractor
Package	DefaultNodeCompartmentExtractor
Interface	ClassifierCompartmentExtractor
Class	ClassifierCompartmentExtractor
Exception	ClassifierCompartmentExtractor
Enumeration	ClassifierCompartmentExtractor
Association	AssociationCompartmentExtractor
Generalization	GeneralizationCompartmentExtractor
Abstraction	GeneralizationCompartmentExtractor
Dependency	GeneralizationCompartmentExtractor
Subsystem	DefaultNodeCompartmentExtractor
Actor	DefaultNodeCompartmentExtractor
UseCase	DefaultNodeCompartmentExtractor
Include	DefaultEdgeCompartmentExtractor
Extend	DefaultEdgeCompartmentExtractor
Object	ObjectCompartmentExtractor
SynchronousMessage	MessageCompartmentExtractor
AsynchronousMessage	MessageCompartmentExtractor
CreateMessage	MessageCompartmentExtractor
SendSignalMessage	MessageCompartmentExtractor
DestroyMessage	MessageCompartmentExtractor
ReturnMessage	MessageCompartmentExtractor

InitialNode	ActivityNodeCompartmentExtractor
ForkNode	ActivityNodeCompartmentExtractor
JoinNode	ActivityNodeCompartmentExtractor
DecisionNode	ActivityNodeCompartmentExtractor
MergeNode	ActivityNodeCompartmentExtractor
FlowFinalNode	ActivityNodeCompartmentExtractor
FinalNode	ActivityNodeCompartmentExtractor
ObjectFlowNode	ActivityStateCompartmentExtractor
ActionState	ActivityStateCompartmentExtractor
SendSignalAction	ActivityStateCompartmentExtractor
AcceptEventAction	ActivityStateCompartmentExtractor
Transition	DefaultEdgeCompartmentExtractor

This mapping is used in CompartmentExtractorFactory to create the appropriate compartment extractor implementation given the object type and element.

1.4 Component Class Overview

Compartment classes

The design of this component is such that some object types that are similar will be handled by one single class. Here, similar is viewed from the point of view of the possible actions and how to handle them.

CompartmentExtractor
An interface to represent methods need to be supported by a CompartmentExtractor CompartmentExtractor is responsible to extract and update several types of compartments from the given DiagramElement object.

AbstractCompartmentExtractor
An abstract implementation for CompartmentExtractor interface. All methods in this class simply throw CompartmentNotSupportedException.This class is provided for convenience when creating implementation of CompartmentExtractor as usually only some operations are supported.

AssociationCompartmentExtractor
Implementation of CompartmentExtractor dealing with association.

ActivityNodeCompartmentExtractor
Implementation of CompartmentExtractor dealing with activity node

DefaultEdgeCompartmentExtractor
Implementation of CompartmentExtractor dealing with graph edge

GeneralizationCompartmentExtractor
Implementation of CompartmentExtractor dealing with generalization, abstraction and

dependency.

MessageCompartmentExtractor
Implementation of CompartmentExtractor dealing with message

ObjectCompartmentExtractor
Implementation of CompartmentExtractor dealing with Object

ActivityStateCompartmentExtractor
Implementation of CompartmentExtractor dealing with activity states

DefaultNodeCompartmentExtractor
Implementation of CompartmentExtractor dealing with graph node

ClassifierCompartmentExtractor
Implementation of CompartmentExtractor dealing with class, interface, and exception

ObjectType
Enumeration containing all object types available

CompartmentExtractorFactory
A class providing a factory method to create different type of concrete implementation of CompartmentExtractor. This is provided for convenience for clients as the client only needs to specify the object type and the associated compartment extractor will be created and returned.

Action classes
Each a distinct action is implemented in one class and independent of each other. Each action can work on any object. This is done by employing CompartmentExtractorInterface.

AddRemoveAction
Represent all update actions which add a new object to the parent GraphElement or remove an existing object.

UpdateAction
Represent the superclass of all update action in this component which change the existing value to another value (not addition or removal).

AddRemoveOperationAction
Represent Add/Remove Operation Action.This action will add/remove an operation graphElement to/from a parent node wrapped inside the given compartment extractor object.

AddRemoveAttributeAction
Represent Add/Remove Attribute Action.This action will add/remove an attribute graphElement to/from a parent node wrapped in a CompartmentExtractor object.

UpdateFirstAssociationEndAction
Represent Update Second Association End Action. This action updates the second association end of the given Graph Element wrapped inside a compartment extractor object.

UpdateSecondAssociationEndAction
Represent Update Second Association End Action. This action updates the second association end of the given Graph Element wrapped inside a compartment extractor

object.

UpdateStereotypeAction
Represent Update stereotype Action. This action updates the stereo type of the parent GraphElement wrapped inside a compartment extractor object.

UpdateNamespaceAction
Represent Update namespace Action. This action updates the namespace of the parent GraphElement wrapped in a compartment extractor object to the given name.

UpdateNameAction
Represent Update name Action. This action updates the name of the parent GraphElement wrapped in a compartment extractor object to the given name.

UpdateVisibilityAction
Represent Update Visibility Action. This action updates the visibility of the parent GraphElement to the given visibility.

UpdateSizeAction
Represent UpdateSize Action. This action updates the size of the parent GraphElement to the given Dimension.

Main classes
DiagramElementUndoableAction
The superclass of all actions in this component. This class extends UndoableAction and provide default impementation of the interface. However, not all methods are relevant, those methods who are not supported by this class will throw UnsupportedOperationException if called. This class stores the parent GraphElement to be the parent container or to be updated. In addition, it provides basic undo/redo mechanism.

DiagramElementAddAction
This class handles all add actions of this component. Client of this class will instantiate this class by giving both the parent and the child GraphElement as well as ProjectConfigurationManager. After that, execute method can be called to execute the Add action.

DiagramElementUpdateAction
Represent the superclass of all update actions in this component. This class employs a CompartmentExtractor class to extract particular GraphElement from the element to be updated.

1.5 Component Exception Definitions

CompartmentMalformedException:
Represent an Exception to represent a situation where the element being processed is malformed, it does not follow the proper compartment hierarchy.

This exception might be thrown in implementation of CompartmentExtractor interface when add/update compartment with incorrect semantic Model.

CompartmentNotFoundException

Represent an Exception corresponds to a situation where the compartment that is supposed to be there, is not there.

This exception might be thrown in implementation of CompartmentExtractor interface when the intended compartment is not available in the element.

CompartmentNotSupportedException
Represent an Exception to represent a situation where the intended compartment is not supported or not supposed to be available.

This exception might be thrown in implementation of CompartmentExtractor interface where the intended compartment is not supported by the element.

1.6 Thread Safety

Actions: Not thread safe, reason: execute/redo/undo might interfere each another. Thus, actions are not thread safe. This should not be a problem as normally, each action object is only used by one thread. Each thread will have its own instance of action object.

Compartment: some operations modifies the corresponding element object. Thus the thread safety will depend on whether Diagram Interchange implementation is thread safe or not.

In conclusion, this component is not thread safe. In multi thread environment, each thread must create a new Action and Compartment objects. In addition, if Diagram Interchange implementation is not thread safe, then a thread safe wrapper is needed.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5
- Compile target: Java 1.5

2.2 TopCoder Software Components

- Action Manager 1.0: defines the classes and interfaces for management of actions
- UML Model Manager 1.0: defines the implementation of UML Model classes.
- UML Project Configuration 1.0: provide implementation of ProjectConfigurationManager used in this component
- Diagram Interchange 1.0: defines the implementation of GraphElement, DiagramElement and all Diagram Interchange classes

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.actions.diagram.elements

com.topcoder.uml.actions.diagram.elements.actions

com.topcoder.uml.actions.diagram.elements.compartments

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

The usage of this component is somewhat obvious. The client of this component will instantiate concrete implementation of the action he wants to perform. Before that, depending on the context, it might be necessary to create a concrete implementation of CompartmentExtractor interface. The action will be configured by the element or compartment extractor and the new element to be updated/removed/added. Such configuration is done in the constructor. Finally, the action will be executed.

There are 4 scenarios presented here.

Scenario 1.1: Update the size of element

```
// Update the size of element to the new size

newSize = new Dimension(50,123)

UndoableAction action = new UpdateSizeAction(element,newSize);

action.execute();

….

// Undo the last action

action.undo();

….

// Redo the last action

action.redo()
```

Scenario 2: Add a new element

GraphNode child // child can be a GraphElement corresponds to child to be added. For example, to add a new **Package**, child is a GraphNode corresponds to Package. To add a new **Action State** GraphNode, child will be a GraphNode corresponds to Action State GraphNode. Assume that child corresponds to the appropriate element has been created outside the component.

Assume that manager:ProjectConfigurationManager has been created outside the component.

UndoableAction action = new DiagramElementAddAction(element,child,manager) // element is the parent Graph Element.

action.execute();

action.undo();

action.redo()

action.undo();

Scenario 3.1: Update the name of an Final Node GraphNode element

GraphNode finalNodeElement // assume that graph node corresponds to final node has been created outside the component

GraphElement name // assume that graphElement corresponds to name has been created

CompartmentExtractorFactory factory = new CompartmentExtractorFactory();

CompartmentExtractor extractor = factory.createCompartmentExtractor(ObjectType.Interface, finalNodeElement);

UndoableAction action = new UpdateNameAction(extractor,name);
action.execute();

Scenario 3.2: Update the name of an Synchronize Message GraphEdge element

GraphEdge syncmsg // assume that graph edge corresponds to Synchronize Message has been created outside the component

GraphElement name // assume that graphElement corresponds to name has been created

CompartmentExtractorFactory factory = new CompartmentExtractorFactory();

CompartmentExtractor extractor = factory.createCompartmentExtractor(ObjectType.SynchronizeMessage,syncmsg); // now use ObjectType.SynchronizeMessage to create appropriate compartment extractor

UndoableAction action = new UpdateNameAction(extractor,name);

Action.execute()

Scenario 3.3: Update the namespace of a Use Case GraphNode element

GraphNode usecase // assume that graphnode corresponds to Use Case has been created outside the component

GraphElement namespace // assume that graphElement corresponds to namespace has been created

CompartmentExtractorFactory factory = new CompartmentExtractorFactory();

CompartmentExtractor extractor = factory.createCompartmentExtractor(ObjectType.UseCase,usecase); // now use ObjectType.UseCase to create appropriate compartment extractor

UndoableAction action = new UpdateNamespaceAction(extractor,namespace);
action.execute();

Scenario 4: Add a new attribute to an Exception element

GraphNode attribute // assume that graphnode corresponds to attribute has been created outside the component

CompartmentExtractorFactory factory = new CompartmentExtractorFactory();

CompartmentExtractor extractor = factory.createCompartmentExtractor(ObjectType.Exception,element);

UndoableAction action = AddRemoveAttribute.createAddAttribute(extractor,name);

Action.execute();

5. Future Enhancements

More actions can be supported, for example update the enumeration literal compartment of an Enumeration, etc.

A default compartment builder could be incorporated for add operations. For example, given UML:Operation, a compartment corresponds to Operation can be built with all attributes set to default.