UML Model – Actions 1.0 Component Specification

1. Design

The UML Model – Actions component declares the interfaces from the UML 1.5 framework, from the Actions package. It provides concrete implementations for each interface and provides powerful API to access the collection attributes.

1.1 Design Patterns

None

1.2 Industry Standards

UML 1.5

1.3 Required Algorithms

There are no complex algorithms in this design.

1.4 Component Class Overview

Action
This interface extends ModelElement interface. The ModelElement interface comes form the UML Model - Core component. An action is the fundamental unit of behavior specification. An action takes a set of input values and uses them to produce a set of output values, though either or both sets may be empty. If both inputs and outputs are missing, the action must have some kind of fixed, nonparameterized effect on the system state, or be performing some effect external to the system. Actions may access or modify accessible, mutable objects. A reference to an object to read or write is an input of the action. Composite actions may include data-transformation actions as well as object-access actions.

ActionAbstractImpl
This is a simple implementation of Action interface and extends ModelElementAbstractImpl from the Core Requirements component. As such, all methods in Action are supported. This class is declared as being abstract.

PrimitiveAction
This interface extends Action interface. A primitive action is one that does not contain any nested actions.

PrimitiveActionAbstractImpl
This is a simple implementation of PrimitiveAction interface and extends ActionAbstractImpl. As such, all methods in PrimitiveAction are supported. This class is declared as being abstract.

ExplicitInvocationAction
This interface extends PrimitiveAction interface. Abstract action that indicates sending a request object to a target object using an explicit argument list. Creates a request that is transmitted to the target object. The request is resolved into a

behavioral effect by the target object or its class based on the type of the request. Depending on the kind of action, the requestor may or may not wait for a reply.

ExplicitInvocationActionAbstractImpl
This is a simple implementation of ExplicitInvocationAction interface and extends PrimitiveActionAbstractImpl. As such, all methods in ExplicitInvocationAction are supported. This class is declared as being abstract.

CallOperationAction
This interface extends ExplicitInvocationAction interface. Assembles the call arguments into an operation call request that is transmitted to the target object, where it causes the selection of a method and the execution of its procedure. The argument values are available to the execution of the invoked procedure as predefined OutputPin values. (They are output pins because they represent values available within the procedure.) The action execution waits until the effect invoked by the request completes and returns to the caller. When the execution of a procedure is complete, its result values are returned to the calling execution. When a return message is received, execution of the action is complete and the return values are used as the result values of the call operation action execution.

CallOperationActionImpl
This is a simple concrete implementation of CallOperationAction interface and extends ExplicitInvocationActionAbstractImpl. As such, all methods in CallOperationAction are supported.

SendSignalAction
This interface extends ExplicitInvocationAction interface. Creates a request signal that is transmitted to the target object where it may cause the firing of a state machine transition and the execution of an attached procedure. The argument values are available to the execution of attached procedures. The requestor continues execution without waiting for the request to be delivered or handled. Any attempt by the state machine to issue a reply is ignored.

SendSignalActionImpl
This is a simple concrete implementation of SendSignalAction interface and extends ExplicitInvocationActionAbstractImpl. As such, all methods in SendSignalAction are supported.

CreateObjectAction
This interface extends PrimitiveAction interface. This action instantiates a concrete classifier. The new object is created, and the classifier of the object is set to the given classifier. The new object is returned as the value of the action. The action has no other effect. In particular, no constructors are executed, no initial expressions are evaluated, and no state machines transitions are triggered. The new object has no attributes values and participates in no links. The semantics is undefined for creating objects from abstract classifiers or from association classes.

CreateObjectActionImpl
This is a simple concrete implementation of CreateObjectAction interface and extends PrimitiveActionAbstractImpl. As such, all methods in CreateObjectAction are supported.

DestroyObjectAction
This interface extends PrimitiveAction interface. This action destroys an object. The classifiers of the object are removed as its classifiers, and the object is destroyed. The action has no other effect. In particular, no destructors are executed, no state machines transitions are triggered, and references to the objects are unchanged. Destroying an object that is already destroyed has no effect.

DestroyObjectActionImpl
This is a simple concrete implementation of DestroyObjectAction interface and extends PrimitiveActionAbstractImpl. As such, all methods in DestroyObjectAction are supported.

1.5 Component Exception Definitions

This component defines no custom exceptions.

The general approach to parameter handling is not to do it. The architectural decision was to allow the beans to hold any state, and delegate to the users of these beans to decide what is legal and when it is legal.

1.6 Thread Safety

This component is not thread-safe, and there is no requirement for it to be thread-safe. In fact, the PM discourages method synchronization. Thread safety will be provided by the application using these implementations.

The classes are made non-thread-safe by the presence of mutable members and collections. In order to provide thread-safety, if that is ever desired, all simple member accessors and collections would need to be synchronized.

2. Environment Requirements

2.1 Environment

JDK 1.5

2.2 TopCoder Software Components

- TC UML Common Behavior 1.0
    - TC UML component defining the Common Behavior.
- TC UML Core 1.0
    - TC UML component defining the Core Requirements.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.topcoder.uml.model.actions

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 demo will demonstrate the usage of these beans. It will show them being instantiated, then used via their interface. This will be the typical usage of such simple entities under any scenario. This demo will focus on showing how a simple attribute is managed, with the understanding that all other attributes are managed in exactly the same manner, and therefore not shown here.

```
// creates an instance of CallOperationActionImpl.
CallOperationAction callOperationAction = new CallOperationActionImpl();

// creates a new instance of Operation.
Operation operation = new OperationImpl();
// use setter to set the operation to this action.
callOperationAction.setOperation(operation);
// use getter to get the operation that will be invoked by this action.
// the retrievedOperation will be the same as the original operation.
Operation retrievedOperation = callOperationAction.getOperation();

// sets the asychronous flag to true, so the operation of this action will be called
 asynchronously.
callOperationAction.setAsynchronous(true);
// we can also get the info about the asychronous flag.
// the result will be : true.
boolean result = callOperationAction.isAsynchronous();

// creates a new instance of Procedure.
Procedure procedure = new ProcedureImpl();
// use setter to set the procedure to this action.
callOperationAction.setProcedure(procedure);
// use getter to get the procedure that belongs to this action.
```

```
// the retrievedProcedure will be the same as the original procedure.
Procedure retrievedProcedure = callOperationAction.getProcedure();
```

5. Future Enhancements

Providing a complete model, or moving to UML 2.