UML Model Activity Graphs 1.0 Component Specification

1. Design

1.1 Overview

The UML Model Activity Graphs component provides interfaces and implementations to support the activity graph functionality of the TopCoder UML tool. There are only four interfaces (three of which are empty) and therefore only four implementing classes.

1.2 Design Patterns

None

1.3 Industry Standards

UML 1.5

1.4 Required Algorithms

None

1.5 Component Class Overview

interface ActivityDiagram extends StateMachine

According to the UML specification (www.omg.org), an Activity Graph is a special case of a state machine that is used to model processes involving one or more classifiers. Implementations of this interface need not be implemented in a thread-safe manner. It is required of the calling application to ensure thread safety.

class ActivityGraphImpl extends StateMachineImpl implements ActivityGraph

Implementation of the ActivityGraph interface. No additional functionality over the StateMachineImpl is provided. This class is thread-safe itself, since it has no instance data, but base classes may not be thread-safe.

interface ActionState extends SimpleState

According to the UML specification (www.omg.org), an Action State is a state that represents the execution of an atomic action, typically the invocation of an operation. Implementations of this interface need not be implemented in a thread-safe manner. It is required of the calling application to ensure thread safety.

class ActionStateImpl extends SimpleStateImpl implements ActionState

Implementation of the ActionState interface. No additional functionality over the SimpleStateImpl is provided. This class is thread-safe itself, since it has no instance data, but base classes may not be thread-safe.

interface CallState extends ActionState

According to the UML specification (www.omg.org), a Call State is an action state that invokes an operation on a classifier. Implementations of this interface need not be implemented in a thread-safe manner. It is required of the calling application to ensure thread safety.

class CallStateImpl extends ActionStateImpl implements CallState

Implementation of the CallState interface. No additional functionality over the ActionStateImpl is provided. This class is thread-safe itself, since it has no instance data, but base classes may not be thread-safe.

interface ObjectFlowState extends SimpleState

According to the UML specification (www.omg.org), an ObjectFlowState is a state in an activity graph that represents the passing of an object from the output of actions in one state to the input of actions in another state. Implementations of this interface need not be implemented in a thread-safe manner. It is required of the calling application to ensure thread safety.

class ObjectFlowStateImpl extends SimpleStateImpl implements ObjectFlowState

Implementation of the ObjectFlowState interface. This class is not thread-safe and is not required to be. It is required of the calling application to ensure thread safety.

1.6 Component Exception Definitions

None

1.7 Thread Safety

The classes in this component are not thread-safe, since thread-safety is not required. It is required of the calling application to ensure thread safety.

To make the individual classes in this component thread-safe, only the ObjectFlowStateImpl would have to be modified, since all the other classes are immutable (based on the attributes that were added to their base classes in this component.) But making the four methods in ObjectFlowStateImpl synchronized is not enough for thread-safety, the base components must be made thread-safe for this purpose.

2. Environment Requirements

2.1 Environment

- At minimum, Java 1.5 is required for compilation and executing test cases.

2.2 TopCoder Software Components

- UML Model State Machines 1.0. The Activity Graph component extends some of the interfaces and classes in the State Machine component.
- UML Model Core 1.0. The Object Flow State uses the Classifier interface from this component.

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.model.activitygraphs

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.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

- Extract the component distribution.
- Import the required classes and interfaces
- Follow the demo

4.3 Demo

4.3.1 Manage an activity graph

```
// create an activity graph
ActivityGraph graph = new ActivityGraphImpl();

// set top
ActionState top = new ActionStateImpl();
graph.setTop(top);

// get top
top = (ActionState) graph.getTop();
```

4.3.2 Manage an ObjectFlowState

```
// create an object flow state
ObjectFlowState state1 = new ObjectFlowStateImpl();

// create object flow state with given type
Classifier type = new ClassImpl();
ObjectFlowState state2 = new ObjectFlowStateImpl(type);

// get attributes of an object flow state
type = state1.getType();
if (state1.isSynch()) {
    // perform some operations here
} else {
    // perform some other operations here
}

// set attributes of an object flow state
state2.setSynch(true);
state2.setType(new ClassImpl());
```

4.3.3 Manage an ActionState

```
// create two action states
```

```
ActionState state1 = new ActionStateImpl();
ActionState state2 = new ActionStateImpl();

// the states can be added to a graph as the two parts of a Transition
Transition trans = new TransitionImpl();
trans.setSource(state1);
trans.setTarget(state2);
```

4.3.4 Manage a CallState

```
// create two call states
CallState state1 = new CallStateImpl();
CallState state2 = new CallStateImpl();

// the states can be added to a graph as the two parts of a Transition
Transition trans = new TransitionImpl();
trans.setSource(state1);
trans.setTarget(state2);
```

5. Future Enhancements

- Provide the complete UML 1.5 model
- Support UML 2.0