ZUML 2 TCUML Coverter - Activity Diagrams 1.1 Component Specification

IMPORTANT: Please note that all the updated changes in version 1.1 are painted in blue color, all the added changes in version are painted in red color.

1. Design

The ZUML 2 TCUML Converter - Activity Diagrams component provides means to help convert the zuml format from Poseidon to the tcuml format from TC UML Tool. This component provides the Activity Diagram conversion tasks on a zuml file.

This component will be used in the TC UML Tool to load a zuml file and transform it into its internal model. The action where this component will be used will be a modified Open file action, which will apply different transformations to the model while reading it, or after the reading process.

The Diagram Interchange representation of the activity diagrams is also different between Poseidon 1.5 and TC UML Tool, so version 1.1 must also convert the diagram interchange graph node structure.

Design overview

This design creates the programmatic classes needed to represent Activity Diagrams as read from a zuml input file. It will be used in the TC UML Tool application in conjunction with XMI Reader 1.0 component that identifies the UML 2.0 elements and builds the internal model using the classes designed in this component. XMI Reader UML Activity Graph Plugin Component 1.0 is configured and used to read zuml activity elements after configuring the ActivityGraphXMIHandler class. This class is a DefaultXMIHandler implementation and depending on the configuration options it can read and build a programmatic model out of an xmi based file structure.

Besides the data structures, the design also offers a utility class with static methods that will allow convenient conversions between the two model to be done by the UML Tool application. It will extract the ActivityGraphs from the Model and return them as a List in Version 1.1.

To convert the diagram interchange graph node structures, we should find out the different places between Poseidon 1.5 and TC UML Tool, and then convert them. Please refer to Algorithms part for more details.

Enhancements

On top of the requirements, the design also offers a few enhancements among which the following:

- full UML 2.0 activity diagram support which allows the usage of the component for a much broader context than it was initially required, as well as the ability to reuse some of the defined elements
- comment uml element conversion is supported by the component. Comments are a very useful basic extension mechanism of UML and can be added to almost any type of uml model element. Having the comments lost during the conversion process limits the model power of expression.
- a very flexible conversion framework that can be used as the base for creating many types of converters from many xmi based files.

1.1 Design Patterns

Façade design pattern: ActivityDiagramConversionFacade is a façade for the conversion system. The application can use the static method to convert the model in one simple method call.

Strategy design pattern: ZUML2TCUMLConvertible implementations are concrete strategies for the calling application.

1.2 Industry Standards

XMI
UML 2.0 and 1.4

1.3 Required Algorithms

The two types of models, zuml and tcuml use different classes to represent similar concepts and also save these concepts using different xmi compliant structures. The following sections try to provide an overview of what is required to be done when performing the conversion. Some elements from Poseidon don't have direct equivalents in the Uml Tool. For such concepts only the model representation is built and the conversion method returns null.
Developers are encouraged to test more possibilities.

In discussing the properties of the activity diagram elements, the elements are arranged in three groups that present similar properties and functionalities. Elements are referred by their zuml xmi tag and if not obvious by their graphical representation name in brackets. The three categories are:

1. Action nodes including: CallAction, Pin(Object_Node), SendSignalAction and AcceptEventAction.
2. Activity nodes including: InitialNode, ActivityfinalNode, FlowFinalNode, ForkNode, JoinNode, DecissionNode and MergeNode.
3. Activity edges including: ActivtyEdge and ExceptionHandler.

1. Action nodes

Too see how an action node would look like, but not in the most simplified version, we are taking a pretty complex Generic Action and analyzing its structure. For the other action types, while their UML meaning is different, they have similar structure.

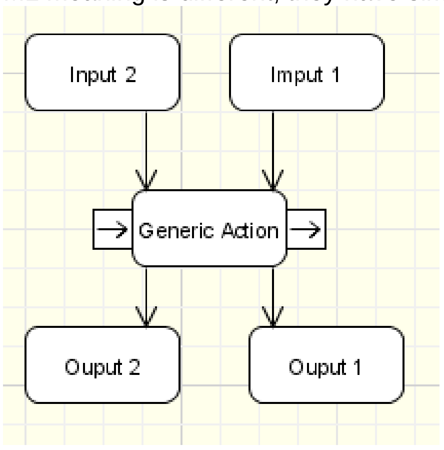

Fig 1. Relation saturated generic action node

And the xmi representation with key structures highlighted, is:

```
<UML2:CallAction xmi.id = 'Im50b29012m115ca1dad3bmm799d' name = 'Generic
Action'
                  visibility = 'public' isSpecification = 'false'>
                  <UML2:ActivityNode.incomingEdge>
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm7937'/>        //Input1's edge reference id
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm7911'/>        //Input2's edge reference id
                  </UML2:ActivityNode.incomingEdge>
                  <UML2:ActivityNode.outgoingEdge>
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm7964'/>        //Output1's edge reference id
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm78e9'/>        //Output2's edge reference id
</UML2:CallAction>
```

2. Activity nodes

Too see how an activity node would look like, but not in the most simplified version, we are taking a pretty complex Generic Action and analyzing its structure. For the other action types, while their UML meaning is different, they have similar structure.

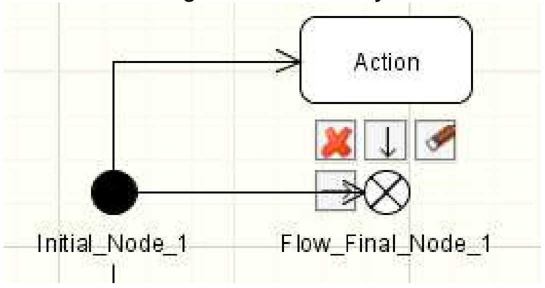

Fig 1. Relation saturated generic activity node

```
<UML2:InitialNode xmi.id = 'Im50b29012m115ca1dad3bmm7e54' name =
'Initial_Node_1'
                  visibility = 'public' isSpecification = 'true'>
                  <UML2:ActivityNode.outgoingEdge>
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm7d15'/>
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm781b'/>
                    <UML2:ActivityEdge xmi.idref =
'Im50b29012m115ca1dad3bmm77fa'/>
                  </UML2:ActivityNode.outgoingEdge>
</UML2:InitialNode>

Note: The other type of nodes will also have
<UML2:ActivityNode.incomingEdge> element. The InitialNode instances will
have the incomingEdge list empty and the FinalNode will have the
outgoingEdge list empty.
```

The ForkNode and JoinNode need special consideration. Poseidon allows these types of nodes to be represented either horizontally or vertically. The conversion is done inversing the width and height properties in the xmi file like in the example bellow:

```
<UML:GraphNode.size>
                <XMI.field>66.6499</XMI.field>
                <XMI.field>15.0</XMI.field>
</UML:GraphNode.size>
```

The convert methods of these two nodes solve this by setting the width to the larger of the two and the height to smaller one. The diagram will look bad immediately after conversion, but as soon as the user clicks on the objects to rearrange them, the layout algorithms will fix the problem.

The JoinNode also has an additional element, called an OpaqueExpression which permits the editing of the join criteria. The structure of node is the following:

```
<UML2:JoinNode.joinSpec>
<UML2:OpaqueExpression xmi.id = 'Im50b29012m115ca1dad3bmm7e2c' name = ''
                      visibility = 'public' isSpecification = 'true' body
= 'and' language = 'java'/>
</UML2:JoinNode.joinSpec>
```

3. Activity edges

Too see how an activity edges would look like, but not in the most simplified version, we are taking a pretty complex Generic Action and analyzing its structure. For the other action types, while their UML meaning is different, they have similar structure.

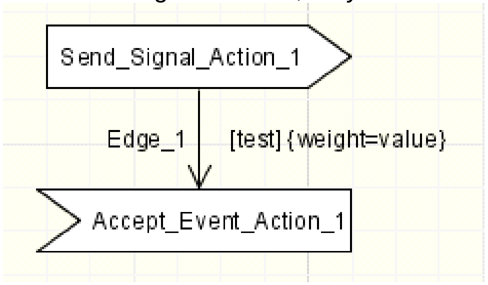

Fig 1. Relation saturated generic edge

```
<UML2:ActivityEdge xmi.id = 'Im50b29012m115ca1dad3bmm77b0' name =
'Edge_1'
                  visibility = 'public' isSpecification = 'false'>
                  <UML2:ActivityEdge.guard>
                    <UML2:OpaqueExpression xmi.id =
'Im50b29012m115ca1dad3bmm7746' name = ''
                      visibility = 'public' isSpecification = 'false'
body = 'test' language = ''/>
                  </UML2:ActivityEdge.guard>
                    <UML2:ActivityEdge.target>
                    <UML2:AcceptEventAction xmi.idref =
'Im50b29012m115ca1dad3bmm7d58'/>
                  </UML2:ActivityEdge.target>
                  <UML2:ActivityEdge.source>
                    <UML2:SendSignalAction xmi.idref =
'Im50b29012m115ca1dad3bmm7d64'/>
                  </UML2:ActivityEdge.source>
</UML2:ActivityEdge>
```

The two types of edges are the normal one presented above and the exception handler. Exception handlers will be treated as normal activity edges. The only difference is an additional inner block with has that defines the handler.

```
<UML2:ExceptionHandler.handlerBody>
```

```
                        <UML2:CallAction xmi.idref =
'Im50b29012m115ca1dad3bmm7c2f'/>
                      </UML2:ExceptionHandler.handlerBody>
```

Using the same XMI Reader class for “UML2:ExceptionHandler” will do the job here.

4. Unstructured element nesting in the zuml format

In zuml we can have the following options:

```
UML:Collaboration
   UML2:BehavioredClassifier.ownedBehavior
      UML2:Interaction // (Sequence Diagram)
      // or
      UML2:Activity // (Activity Diagram)
// or
UML:UseCase
   UML2:BehavioredClassifier.ownedBehavior
      UML2:Interaction // (Sequence Diagram)
      // or
      UML2:Activity // (Sequence Diagram)
```

This means we cannot make a decision when reading the xmi file since XMI Reader is looking forward only. There is no problem when the outside element is a Collaboration instance, but when it’s any other instance a new Collaboration will have to be created in the model directly. Other Collaboration instances shouldn’t be reused because this would mean to let two otherwise distinct diagrams be mixed in a single representation.
This task will be done using a com.topcoder.umltool.xmiconverters.poseidon5.XMIConverter instance to adapt any type of BehavioredClassifier nesting so that we can have the above mentioned cases parsed as if they were well structured.

5. Extract the ActivityGraphs from the Model
In version 1.0, it already has a Model conversion function. So we just need to extract the ActivityGraphs from the converted Model and return them as List in Version 1.1. The following steps can extract the ActivityGraphs.

- In version 1.0, the transformed result of ZUMLActivityDiagramImpl is StateMachine instance, need to change it to ActivityGraphImpl instance in version 1.1.
- Add an ArrayList instance for ActivityDiagramConversionFacade to keep the ActivityGraphs.
- While converting the Model, initialize the ArrayList instance, and add all converted ActivityGraph instances to List, the ActivityGraph instance is got from ZUMLActivityDiagram#convertToTCUML() method, then return the List.

6. Convert the diagram interchange graph node structures

UML:Diagram has an UML:GraphElement.contained property, which contains the GraphNodes and GraphEdges for the representations of activities and transitions. These GraphNodes and GraphEdges contain, at their turn GraphNodes for the name compartment, stereotype compartment, ...
A GraphNode represents an activity graph element, and it's contained graph nodes represent the name, stereotype...

To convert the graph node structures, it needs to find out two xmi format's difference and convert them. Compare the graph node structures between Poseidon 1.5 and TC UML Tool, it has following difference.

a) Attributes' different
It has an "isVisible" attribute in Poseidon 1.5, but it has "visible" attribute in TC UML Tool.
But it can be handled in DiagramInterchangeXMIHandler when parsing the xmi file, so it's not a converted issue here.

b) Title graph node
It has a title graph node in Poseidon 1.5, but it doesn't have equivalent instance in TC UML Tool.
It already removed in "ZUML to TCUML Converter" component, so it's not a converted issue here.

c) Different Properties
The Properties are different between Poseidon 1.5 and TC UML Tool.
It was fixed in "ZUML to TCUML Converter" component.

d) Different node structure
It has different structure in "position", "size", "viewpoint", "waypoints" node between Poseidon 1.5 and TC UML Tool.

Take "size" node for example, it has following structure in Poseidon 1.5.

```
  <UML:GraphNode.size>
     <XMI.field>0.0</XMI.field>
     <XMI.field>0.0</XMI.field>
   </UML:GraphNode.size>
```

But in TC UML Tool, it has different structure.

```
  <UML:GraphNode.size>
     <UML:Dimension height="459.0" width="686.0"/>
   </UML:GraphNode.size>
```

Fortunately, this issue was fixed by DiagramInterchangeXMIHandler when parsing the xmi file, so it's not a converted issue here.

e) Different node reference
It has different reference in some "SemanticModel" and "owner" nodes between Poseidon 1.5 and TC UML Tool.

Take "owner" for example, it has following structure in Poseidon 1.5.

```
  <UML:Diagram.owner>
     <UML:Uml1SemanticModelBridge xmi.id="Im7ace8e2fm11628776085mm7f27"
presentation="">
        <UML:Uml1SemanticModelBridge.element>
           <UML2:Activity xmi.idref="Im7ace8e2fm11628776085mm7f2a"/>
        </UML:Uml1SemanticModelBridge.element>
     </UML:Uml1SemanticModelBridge>
   </UML:Diagram.owner>
```

But in TC UML Tool, it has different structure.

```
  <UML:Diagram.owner>
     <UML:Uml1SemanticModelBridge xmi.id="TC_UML_ad25cd8c-103a-4e6a-ad27-
7d7b5b5efb3d">
        <UML:Uml1SemanticModelBridge.element>
           <UML:ActivityGraph xmi.idref="TC_UML_6e1d6518-76a2-4969-834b-
ec511ed9e317"/>
        </UML:Uml1SemanticModelBridge.element>
     </UML:Uml1SemanticModelBridge>
   </UML:Diagram.owner>
```

We can find that one refers to <UML2:Activity> and the other refers to <UML:ActivityGraph>. It's the right thing that we should convert.

It already provides the UML model elements from UML 2.0 in version 1.0. So it only needs to find out all GraphNode and GraphEdge under Diagram node and convert them in version 1.1.

The following sample code shows how to find and convert them.
```
function convertDiagrams(List<Diagram> diagrams) {
        for all diagrams {
                call convertContainedElements() to convert containeds,
                convert owner,
                convert SemanticModel.
        }
}

function convertContainedElements(List<DiagramElement> diagramElements) {
        for all diagramElements {
                convert SemanticModel,
                call convertContainedElements() recursively to convert its'
containeds.
        }
}
```

Note: There are two xmi files which can help to understand the difference in the /docs folder.

TCUML TaggDefinition bug
There is a problem with saving two TaggedValue dependant nodes, SendSignalAction and AcceptEventAction nodes. This design does not attempt to fix this bug in its first version. At the date of submission there was no PM answer in the forums. After further instructions, this will be fixed in the next stage of development.

Note: There are two rtf files in the /docs folder of this submission, presenting a simplified version of the activity diagram elements hierarchy.

1.4 Component Class Overview

ActivityDiagramConversionFacade:
This class is a facade that hides the conversion system from the application. It is used to convert uml2 zuml files to uml 1.4 tc internal model.
The application will use it with the convertModel() static method. It convert all Activity Diagrams contained in the respective model.
In version 1.1, the return result of convertModel() is changed to List<ActivityGraph>, and convertDiagrams() is added to convert the Diagram.
This class holds no state but calling the static mehtod from different threads with the same model argument will lead to unpredictable results, so it is not thread safe.

ZUML2TCUMLConvertible (interface):
An interface that defines the generic convertToTCUML() method. Every concrete class that represents an uml 2.0 model element and implements this interface defines
the its custom conversion algorithm. It extends the ModelElement which permits the use of the elements defined in this design in tcuml internal model.
Implementations of this interface are not required to be thread safe.

ZUMLActivityDiagram (interface):
Generic ZUMLActivityDiagram interface. Defines a uml 2.0 ActivityDiagram from a conceptual poin of view with the least demanding interface contract.

In this context, a ZUMLActivityDiagram is collection of nodes and edges defined in an activity diagram.
This interface defines only method to retrieve the edges and nodes, this being the minimal contract required by a layered conversion framework for this element.
It extends ZUML2TCUMLConvertible interface to allow implementations to be converted in the same way.
Implementations of this interface are not required to be thread safe.

ZUMLActivityEdge (interface):
Generic ZUMLActivityEdge interface. An ActivityEdge is composed of a source, a target and a guard that describes the edge.
This interface defines only method to retrieve the three consituent elements, this being the minimal contract required by a layered conversion framework for this element.
It extends ZUML2TCUMLConvertible interface to allow implementations to be converted in the same way.
Implementations of this interface are not required to be thread safe.

ZUMLOpaqueExpression (interface):
Generic ZUMLOpaqueExpression interface. An OpaqueExpression is composed of a body that represents the expression and a language that defines the language in which the expression is expressed.
This interface defines only method to retrieve the body and language, this being the minimal contract required by a layered conversion framework for this element.
It extends ZUML2TCUMLConvertible interface to allow implementations to be converted in the same way.
Implementations of this interface are not required to be thread safe.

ZUMLActivityNode (interface):
Generic ZUMLActivitynode interface. An ActivityNode is composed of a list of incomming edges and a list of outgoing ones.
This interface defines only the methods to retrieve the incoming and outgoing edges, this being the minimal contract required by a layered conversion framework for this element.
It extends ZUML2TCUMLConvertible interface to allow implementations to be converted in the same way.
Implementations of this interface are not required to be thread safe.

AbstractZUMLActivityNode (abstract class):
This abstract class implements the ZUMLActivityNode interface and extends the AbstractModelElement class.
It defines the common methods of any type of ZUMLNode to lessen the imlementation contract. It also defines methods to manage the incoming and outgoing edges.
This class is mutable and not thread safe.

AbstractZUMLOpaqueExpression (abstract class):
This abstract class implements the ZUMLOpaqueExpression interface and extends the AbstractModelElement class.
It defines the common methods of any type of ZUMLOpaqueExpression to lessen the implementation contract. It also defines methods to set and get the body and language of an OpaqueExpression model element.
This class is mutable and not thread safe.

AbstractZUMLActivityEdge (abstract class):
This abstract class implements the ZUMLActivityEdge interface and extends the AbstractModelElement class.
It defines the common methods of any type of ActivityEdge to lessen the implementation

contract. It also defines methods to set and get the source, target and guard of an ActivityEdge.
This class is mutable and not thread safe.

AbstractZUMLActivityDiagram (abstract class):
This abstract class implements the ZUMLActivityDiagram interface and extends the ActivityGraphImpl class.
It defines the common methods of any type of ZUMLActivityDiagram to lessen the implementation contract.
It also defines methods to manage the nodes and the edgets of an Activity Diagram instance.
This class is mutable and not thread safe.

ZUMLActivityDiagramImpl:
This class represent an UML2:Activity element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model. It should return an ActivityGraph instance in version 1.1.
This class is mutable and not thread safe.

ZUMLActivityEdgeImpl:
This class represent an UML2:ActivityEdge element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLExceptionHandlerImpl:
This class represent an UML2:ExceptionHandler element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLOpaqueExpressionImpl:
This class represent an UML2:OpaqueExpression element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLJoinNodeImpl:
This class represent an UML2:JoinNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLPinNodeImpl:
This class represent an UML2:PinNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLSendSignalActionImpl:
This class represent an UML2:SendSignalAction element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLFlowFinalNodeImpl:
This class represent an UML2:FlowFinalNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLSendSignalActionImpl:
This class represent an UML2:SendSignalAction element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLInitialNodeImpl:
This class represent an UML2:InitialNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLActivityFinalNodeImpl:
This class represent an UML2:ActivityFinalNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLCallActionImpl:
This class represent an UML2:CallAction element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLAcceptEventActionImpl:
This class represent an UML2:AcceptEventAction element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLDecisionNodeImpl:
This class represent an UML2:DecisionNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLMergeNodeImpl:
This class represent an UML2:MergeNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

ZUMLForkNodeImpl:
This class represent an UML2:ForkNode element. Will be used by the XMI Reader and its plugins to create a object representation of the read zuml data.
It implements the convertToTCUML() method returning the converted tcuml object that can be used in the internal uml tool model.
This class is mutable and not thread safe.

1.5 Component Exception Definitions

None.

1.6 Thread Safety

This component is no required to be thread safe and it isn't. The data structures are mutable so thread unsafe and the utility is stateless but processes thread unsafe structures which will get corrupted if used in a concurrent environment.
In version 1.1, it added a method just like convertModel() and the thread safety is the same with verion 1.0.
If required, thread safety can be achieved by synchronizing the methods that mutate data structure classes.

2. Environment Requirements

2.1 Environment

Windows 2000
Windows Server 2003

2.2 TopCoder Software Components

XMI Reader 1.0 - used for reading the zuml file with a number of configured handlers

XMI Reader UML Activity Graph Plugin 1.0 – used to convert the zuml file to model elements defined in this design

Diagram Interchange 1.0 - used to set and get associations element associations from the tcuml model

UML Model State Machines 1.0 – used for StateMachine interfaces and classes used to convert the zuml to the tcuml file format

UML Model Core 1.0 – used for ModelElement and AbstractModelElement elements needed in the design.

UML Model Data Types 1.0 – used for ModelElement and AbstractModelElement elements needed in the design.

UML Model Management 1.0 – used for the Model class that is converted by ActivityDiagramConversionFacade.

ZUML 2 TCUML Converter 1.0 – used for XMIConverterUtil. For details please see 1.3.4.

Note: Configuration Manager 2.1.5 is not used directly in this component because there is nothing that requires configuration. It is used indirectly to configure **XMI Reader 1.0** and **XMI Reader UML Activity Graph Plugin** dependencies.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Names

com.topcoder.umltool.xmiconverters.poseidon5
com.topcoder.umltool.xmiconverters.poseidon5.activity
com.topcoder.umltool.xmiconverters.poseidon5.activity.impl

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

XMI Reader and XMI Reader UML Activity Graph Plugin Component 1.0 has to be configured with handlers that read UML 2.0 elements from the zuml file. This can be done by configuring the following parts:

The following name/values mappings have to be configured in the plugin component in the xml_to_element_mapping property:

Name	Value
UML2:Activity	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLActivityDiagramImpl
UML2:InitialNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLInitialNodeImpl
UML2:ActivityFinalNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLActivityFinalNodeImpl
UML2:FlowFinalNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLFlowFinalNodeImpl
UML2:ForkNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLForkNodeImpl
UML2:JoinNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLJoinNodeImpl
UML2:DecisionNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLDecisionNodeImpl
UML2:MergeNode	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLMergeNodeImpl
UML2:Pin	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLObjectNodeImpl
UML2:ExceptionHandler	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLExceptionHandler
UML2:ActivityEdge	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLActivityEdgeImpl
UML2:CallAction	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLCallActionImpl
UML2:SendSignalAction	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLSendSignalActionImpl
UML2:AcceptEventAction	com.topcoder.umltool.xmiconverters.poseidon5.activity.impl.ZUMLAcceptEventActionImpl

XMI Reader has the optional “handlers” parameter that contains name/value pairs mapping element names to XMIHandler class names. The names are the fully-qualified element names and the values are the fully-qualified class name for the XMIHandler instance that will handle.

For each element in the above table, add an entry having the contents:

```
name = UML2:Element
```

```
value = com.topcoder.xmi.reader.handlers.uml.
        activityGraph.ActivityGraphXMIHandler
```

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Execute ‘nant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

The demo assumes all the needed configurations have been done as specified in section 3.3.

4.3.1 *Read zuml file*

```
//Create or retrieve an XMIReader
XMIReader reader =  new  XMIReader();

//parse the zuml and create an internal graphical representation of the file data
reader.parseZipFile("test_files" + File.separator + "input.zuml");

//Depending on the input file this will result in an internal Model creation that contains the defined
//ZUML instances.
```

4.3.2 *Convert the model*

```
//A model containing uml 2.0 zuml  elements can be converted using the factory's static methods
List<ActivityGraph> graphs = ActivityDiagramConversionFacade.convertModel(model);

//If the model does not contain any activity diagrams the conversion will do nothing and the
//converted model will be the same as the one passed to convert
//The return graphs List should contain converted ActivityGraphs.
//Then can set the ActivityGraphs List to umlModelManager.
```

4.3.3 *Apply post reading transformations to the model*

```
//The conversion framework is flexible and any adjustments can be done before doing the
//conversion operation

ZUMLPinNodeImpl pinNode = new ZUMLPinNodeImpl();

activityDiagram.add(pinNode);

activityDiagram.remove(pinNode);

//any operation can be done on the uml 2.0 zuml elements and the model can be built or adjusted
//programmatically
```

4.3.4 Convert the Diagram

```
//Initial the UMLModelManager manager.

ActivityDiagramConversionFacade.convertDiagrams(manager.getDiagrams());
```

//The Activity Diagrams in manager are converted to TC UML Tool graph node structure.

5. Future Enhancements

Supporting conversions from more file types and uml elements.