ZUML 2 TCUML Converter 1.0 Requirements Specification

1. Scope

1.1 Overview

The ZUML 2 TCUML Converter component provides means to convert the zuml format from Poseidon to the tcuml format from TC UML Tool. This component provides the initial conversion tasks on a zuml file. There will be future enhancements to this component (or different components) that will take care of the differences between the two formats found in the concrete types of diagrams.

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

1.2 Logic Requirements

“zuml” represents the file format used by Poseidon.

“tcuml” represents the file format used by TC UML Tool.

This component will provide classes that will help convert a zuml file into the TC UML Tool’s internal model. The user will eventually use the TC UML Tool to save the model into tcuml format.

1.2.1 Zuml Header parsing

Empty projects saved in files show no difference in the XMI, except the zuml having populated the “XMI.header” node. This node is ignored by the XMI Reader, but this component will provide a handler for it and its subnodes. The handler will react to “XMI.metaModelVersion” and it will be configured with the acceptable versions. An exception will be thrown if the version from xmi is not acceptable.

This is a handler to be plugged in the XMI Reader component.

1.2.2 Diagram size

The size of the Diagram needs to be updated, as if it is kept as (0,0) as it is saved in zuml, the actual diagram will not be visible in the TC UML Tool. It will default to some value if the diagram has no children or only the title GraphNode (see it’s description below), or it will be computed using the size and position of the direct contained graph nodes and the waypoints of the edges (adding a configured margin).

This is a post reading task.

1.2.3 Diagram title GraphNode

The zuml’s diagrams have a node for diagram’s title, which must be ignored.

Diagram contains a GraphNode with SimpleSemanticModelElement with typeInfo="NameCompartment".

The NameCompartment GraphNode contains

- a GraphNode with SimpleSemanticModelElement with typeInfo="DiagramNamePrefix"
- a GraphNode with SimpleSemanticModelElement with typeInfo=" Name".

This title GraphNode must be removed.

An alternative could be to create a Text node, but removing it must be provided as an option.

This is a post reading task.

1.2.4 Replace properties

There will be a post-reading task that will iterate all the graph nodes and replace certain properties with other properties. The task will be configured with the key-to-key pairs. If a pair of keys is ("fill", "FILL_COLOR"), it will replace the property ("fill", "#0000000") with ("FILL_COLOR", "#0000000"). This is needed because the properties of the graph nodes from the .zuml have the same meaning as those from .tcuml, only different keys or different format for the values.

It will also have an option to apply a transformation to the values. The only required transformation is to change the value string to lower letters. The transformation will be mapped to ALL properties, or to certain properties (specified with the keys).

This is a post reading task.

1.2.5 Data type reference

The attributes of a classifier have a type property, which references some Classifier. The .zuml indicates this using a node named "UML2:TypedElement.type" and the classifier is specified using the concrete type (i.e. Class, …). The .tcuml indicates this information using a "UML:StructuralFeature.type" node and the classifier is specified using a node named "UML:Classifier". There will be a handler that will react to the "UML2:TypedElement.type" node in the same manner that the handler in the reader plugin reacts to the "UML:StructuralFeature.type" node, setting the property to be "UML:StructuralFeature.type", instead of named "UML2:TypedElement.type".

The same thing must be done to convert the "UML:StructuralFeature.type" node from the Parameters into "UML:Parameter.type".

This is a handler to be plugged in the XMI Reader component.

1.2.6 Remove the Method node

The .zuml contains an extra node with the name "UML:Method", besides the "UML:Operation" node. This must be ignored by the Reader. This means that there will be a handler that will react to the "UML:Method" node, "UML:Method.body", "UML:ProcedureExpression", "UML:Method.specification" nodes and to ignore the contained node "UML:Operation" which has the "idref" property only.

This is a handler to be plugged in the XMI Reader component.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool import .zuml files saved by Poseidon.

1.5 Future Component Direction

None at this moment.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

None.

2.1.3 Environment Requirements

- Development language: Java1.5
- Compile target: Java1.5

2.1.4 Package Structure

com.topcoder.umltool.xmiconverters.poseidon5

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The default diagram size.
- The transformations.
- The acceptable versions for the zuml header.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

XML

3.2.2 TopCoder Software Component Dependencies:

- Configuration Manager 2.1.5

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.