ZUML_2 TCUML_Converter_1.0 Component Specification

1. Design

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.

λ Design Overview

1. Because the XMI Reader 1.0 component doesn't create handler by Object Factory, the variables in the handler can't be configured when using the configuration file to add handler to the XMI Reader.

   In this component, the additional handlers will be configured by a utility class and that class use Object Factory to create the handlers.

2. The post reading tasks are handled in the utility class.

3. XMI Reader 1.0 has another weakness, only one XMIHandler can be added to one element, so when we want to ignore the <UML:Operation> in <UML:Method>, ***we must wrap the original handler for <UML:Operation> element. To make it general, the Chains of Responsibility Pattern is used to let the XMIConverter instance do some tasks before or after the original handler's processing.*** It is very useful functionality, by using the XMIConverter's chains, we can adjust the diagram size when the <UML:Diagram> element ends, we can remove the title or convert the title to text node in diagram and we also can replace the properties in the diagram and all its sub-elements when the <UML:Diagram> element ends. ***Also, the RS 1.2.5 can be achieved by adding a RenameConverter to rename the element before the <UML:Class> handler's processing, rather than adding a new XMIHandler.***

4. The ZUMLMethodHandler must be handled although it does nothing. Because if it is not added, <UML:Method> will be handled by the handler for <UML:Class>, this may cause exception.

5. When converting the title node in diagram to text node, we should convert the contained elements in title node to TextElement, but not convert the whole title node to TextElement, because a text node is a graph node with position, size and contained TextElement instance.

λ Typical usages of this component

The handlers or converters for RS 1.2.1, 1.2.5 & 1.2.6 must be configured in the file, the converters for RS 1.2.2, 1.2.3, 1.2.4 can be configured in the file, or the

application can just use the XMIConvertersUtil directly as doing post reading tasks.

λ **Enhancements**

1. In RS 1.2.3, "An alternative could be to create a Text node", this is implemented as an enhancement. See XMIConvertersUtil.convertDiagramTitleToText method.
2. In RS 1.2.4, this design adds a regular expression key matching transformer.
3. In RS 1.2.5, this design adds a converter to rename any element name to achieve it, so the rename converter can be also used to rename other elements in future.
4. Converters (DiagramSizeConverter, DiagramTitleConverter and GraphPropertiesConverter) are provided to do the post reading tasks in the processing of reading. This is useful because the original logic of Open Action in TC UML Tool has minimal changes; the only change is using the XMIConvertersUtil to configure the XMIReader instance. See demo section for details.
5. This design uses Chains of Responsibility Pattern to implement XMIConverter chains to do some tasks before or after the original handler's processing. This provides a useful enhancement to XMI Reader 1.0. And it can be used for future converters from ZUML to TCUML.

1.1 Design Patterns

Decorator Pattern, XMIConverterHandler class wraps the existing handler in XMIReader to provide the XMIConverter chains for it.

Chains of Responsibility Pattern, XMIConverter and XMIConverterChain implement this pattern, multiple XMIConverter instances can be added to one handler in XMIReader.

1.2 Industry Standards

XML

Java

1.3 Required Algorithms

1.3.1 Calculate the size of Diagram

Method: *XMIConvertersUtil.adjustGraphNodeSize*
Input: graphNode: GraphNode, emptyWidth: double, emptyHeight: double, margin: double
Purpose: calculates the size of the given graph node.

```
1.     Set maxWidth and maxHeight as 0
2.     For each contained graph node(Note: just GraphNode, not each DiagramElement)
2.1    Set the maxWidth to the max of maxWidth and (position.x + size.width) of the
graph node
2.2    Set the maxHeight to the max of maxHeight and (position.y + size.height) of the
graph node
3.     For each contained graph edge(Note: just GraphEdge, not each elemnt)
3.1    For each way point in the waypoints collection in the edge.
3.1.1  Set the maxWidth to the max of maxWidth and point.x of the graph node
3.1.2  Set the maxHeight to the max of maxHeight and point.y of the graph node

4.     If both maxHeight and maxHeight are 0, it's a empty diagram, set the size of
graph to the default empty size.
5.     Otherwise, set the size of graph node with width = maxWidth + margin and height
= maxHeight + margin
```

1.3.2 *Remove Diagram Title*

Method: *XMIConvertersUtil.removeDiagramTitle*
Input: diagram: Diagram
Purpose: Remove the title graph node in the diagram which generated by Poseidon.

1. Get the first graph node with the "semanticModel" having typeInfo=" NameCompartment" in the contained DiagramElement list.

2. Remove the found graph node

1.3.3 *Convert Diagram Title to Text Node*

Method: *XMIConvertersUtil.convertDiagramTitleToText*
Input: diagram: Diagram, properties: Map<String, String>
Purpose: Convert the title node in diagram to a text node. The position and size is not changed.

1. Get the first graph node with the "semanticModel" having typeInfo="NameCompartment" in the contained DiagramElement list. Set it to variable titleNode.
2. Change the typeInfo to "FreeText" in the semanticModel element in the titleNode.
3. Clear the contained diagram elements in titleNode : titleNode.clearContaineds()
4. For each property entry in the properties map, add the property to titleNode
5. Create a TextElement instance text with diagram name prefix and diagram name as its text.
   5.1 diagram name prefix:
   Get typeInfo of diagram semanticModel first:
   ((SimpleSemanticModelElement)diagram.getSemanticModel())
   .getTypeInfo();
   Using the mapping:
   "ClassDiagram" -> "cd:"
   "UseCaseDiagram" -> "ud:"
   "StateDiagram" -> "sm:"
   "ActivityDiagram" -> "ad:"
   "CollaborationDiagram" -> "cld:"
   "SequenceDiagram" -> "sd:"
   "ComponentDiagram" -> "cd:"
   "DeploymentDiagram" -> "dd:"
   5.2 diagram name: diagram.getName()
6. Add the created TextElement instance to the contained of titleNode: titleNode.addContained(text);

1.3.4 *Replace the keys of properties of DiagramElement instance*

Method: *XMIConvertersUtil.replacePropertyKeys*
Input: diagramElement: DiagramElement,
toReplacePropertyKeys: Map<String, String>
Purpose: Replace the keys of properties in the diagramElement and all the contained elements. This method will be executed recursively to replace all the properties keys in the contained elements.

1. For each property in the diagramElement, if the key is in the toReplacePropertyKeys map, set the new key to the property:
   Property.setKey(toReplacePropertyKeys.get(property.getKey()))
2. For each contained diagram element in this element, recursively replace the keys of properties in the contained diagram:
   replacePropertyKeys(containedElement, toReplacePropertyKeys);

1.3.5 *Transform the values of properties of DiagramElement instance*

Method: *XMIConvertersUtil.transformPropertyValues*
Input: diagramElement: DiagramElement,

```
            transformers: List<PropertyValueTransformer>
        Purpose: Transform the values of matched properties in the diagramElement
        and all the contained elements. This method will be executed recursively
        to transform all the matched properties' values in contained elements.

1.      For each property in the diagramElement
1.1     For each transformer in transformers list
                If the property is matched by the transformer:
                transformer.match(property.getKey()) is true
                   Transform the value of this property:

              property.setValue(transformer.transform(property.getValue()));
2.      For each contained diagram element in this element, recursively replace the
keys of properties in the contained diagram:
              transformPropertyValues(containedElement, transformers);
```

1.4 Component Class Overview

¬ **XMIConvertersUtil:** This utility class is used to: 1. Configure the XMIReader to add additional XMIHandler instances to convert Poseidon ZUML file and add XMIConverter instances for existing XMIHandlers to do some conversions before or after the handler processing. 2. Adjust the Diagram size of ZUML because it is always (0, 0) and the TC UML Tool can't display the diagram. 3. Remove Diagram title node because TC UML Tool can't display it. 4. Convert Diagram title node to text node so that TC UML Tool can display it. 5. Replace the keys of properties in diagram element. Because the properties keys are different between ZUML and TCUML 6. Transform the values of properties in diagram element by given transformers.

¬ **PropertyValueTransformer:** This interface is used to transform the value of matched property. It defines two methods - match and transform.

¬ **RegexToLowerPropertyValueTransformer:** The default property value transformer implementation in this component. This class uses regular expression to match the key of property, and transform the value of matched property to Lower case (use toLowerCase in String).

¬ **XMIConverter:** This is the responsibility interface of Chains of Responsibility Pattern. Post-processing or pre-processing tasks can be done in this interface. It can also ignore the processing of the original XMIHandler. In a word, it provides the full control of the original handler's processing.

¬ **XMIConverterChain:** The chain interface in the Chains of Responsibility Pattern. This interface is used by XMIConverter to control the processing of responsibility chains. The implementation of XMIConverter can do some tasks before and after the next chain processing.

¬ **AbstractXMIConverter:** The abstract XMIConverter implementation is used to provide the convenience to sub-classes. The sub-classes can just override the method they want to implement. All the processing methods in this class just delegate the calling to next chain.

¬ **DefaultXMIConverterChain:** This is the default implementation of XMIConverterChain interface. It stores the next chain and corresponding XMIConverter instance, if neither next nor converter is null, the chain's processing method will call the converter to do the processing, else it will call the handler directly to do the processing.

¬ **DiagramSizeConverter:** This converter is used to adjust the size of Diagram when the <UML:Diagram> element ended, it will use the utility method adjustDiagramSize in XMIConvertersUtil class to adjust diagram size. When the diagram doesn't contain any diagram element,

the size should be configured empty size; otherwise, the size will be calculated by the sub elements and the configured margin.

¬ **DiagramTitleConverter:** This converter is used to convert the title node of Diagram when the <UML:Diagram> element ended, it will use the utility methods in XMIConvertersUtil class to convert the title node. If the convertToTextElement is false, the title node will be just removed, otherwise, the title node will be converted to text node with the title text displayed in Poseidon unchanged and with the configured properties.

¬ **GraphPropertiesConverter:** This converter is used to convert the properties of Diagram and contained elements when the <UML:Diagram> element ended, it will use the utility methods in XMIConvertersUtil class to convert the properties.

¬ **RenameConverter:** This converter is used to rename the elements in startElement and endElement method so that the original handler for the elements can work. The elements need to rename can be configured. This converter will rename the elements before calling the next chain processing.

¬ **IgnoreOperationInMethodConverter:** This converter is used to ignore the <UML:Operation> element in <UML:Method> element. It will check the active handlers in XMIReader, if the previous handler is ZUMLMethodHandler instance; the processing will return immediately, the next chain will not be called.

¬ **ZUMLHeaderHandler:** This handler is used to handle the <XMI.header> element and extract the version in <XMI.metaModelVersion> element to compare it to the configured acceptable versions, if the version is not contained by acceptable versions, UnsupportedVersionException will be thrown.

¬ **ZUMLMethodHandler:** This handler is used to handler <UML:Method> element in ZUML file. This is a empty handler that do nothing for that element, it is added to the XMIReader so that the <UML:Method> will not be handled(handled by this handler, and this handler do nothing.), otherwise, the <UML:Method> element will be handled by <UML:Class> handler, that may cause exception.

1.5 Component Exception Definitions

¬ **XMIConverterConfigurationException[custom]:** This exception extends BaseException and is thrown if errors occurred when configuring the XMIReader in XMIConvertersUtil or configuring any other classes in this component.

¬ **UnsupportedVersionException[custom]:** This exception extends SAXException and is thrown when the version of parsed ZUML file is not acceptable.

¬ **SAXException:** This exception is thrown if errors occur in the callbacks when parsing the file.

¬ **IllegalArgumentException:** This exception is thrown if any argument is invalid in this component. (For example, null, empty argument sometime is not accepted.)

¬ **PatternSyntaxException:** This exception is thrown by RegexToLowerPropertyValueTransformer if the regular expression passed in is invalid.

1.6 Thread Safety

This component is not thread-safe. And it is not required in this component.
Some classes such as XMIConverterHandler and ZUMLHeaderHandler are mutable. In fact, the XMIHandler instance is always mutable as it contains mutable lastRefObject and lastProperty variables.
We don't need to care about the thread-safety because all the handlers are called by single thread, because the SAX parsing is in single thread mode.

Note: The thread-safety description in XMIHandler in XMI Reader component is wrong, because the lastRefObject and lastProperty is mutable, requiring the implementations to be thread-safe is unnecessary. The mutable variables will be changed by other thread in multi-threads environment although the setters are synchronized. As a result, the code will doesn't work because the two variables are used to communicate with other handlers.

2. Environment Requirements

2.1 Environment

λ Development language: Java1.5

λ Compile target: Java1.5

2.2 TopCoder Software Components

¬ **Base Exception 1.0**: The XMIConverterConfigurationException extends BaseException

¬ **Configuration Manager 2.1.5**: It is used to load configurations for this component from file.

¬ **XMI Reader 1.0**: This component is an extension of XMI Reader component, as this component is used to convert the ZUML format to TCUML format

¬ **Object Factory 2.0.1:** It is used to create XMIHandler and XMIConverter instances in this component.

¬ **Diagram Interchange 1.0.1**: The diagram relative classes are defined in the component.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.topcoder.umltool.xmiconverters.poseidon5

3.2 Configuration Parameters

Note: All the strings read from configuration file should be trimmed first.

*3.2.1 Configuration parameters for **ConverterConfigManager** class*

Name
Description
Required/Type/
Default Value
ObjectFactoryNamespace

The namespace of ObjectFactory used in this component.
Required/String
AdditionalHandlers
The additional XMIHandler instances to XMIReader
Optional/
Configuration properties
AdditionalHandlers
.<PropertyName>
The subproperties' names are the TYPE of XMIHandler instances to XMIReader
Required/String
AdditionalHandlers
.<PropertyValue>
The subproperties' values are the object name in ObjectFactory of the XMIHandler instances.
The format is "key:identifier" or "key"
Required/String
Converters
The XMIConverters added to existing handlers in XMIReader, The earlier declaration means the more near to the handler.
See the sample configuration for details.
Optional/
Configuration properties
Converters
.<PropertyName>
The subproperties' names are the TYPE of existing XMIHandler instances in XMIReader
Required/String
Converters
.<PropertyValue>
The subproperties' values are the object name in ObjectFactory of the XMIConverter instances.
The format is "key:identifier" or "key"
Required/String

Sample Configuration:

```
<CMConfig>
    <Config name="com.topcoder.umltool.xmiconverters.poseidon5">
        <Property name="ObjectFactoryNamespace">
            <Value>object_factory_namespace</Value>
        </Property>
        <Property name="AdditionalHandlers">
            <Property name="XMI.header">
                <Value>handlers:xmiheader</Value>
            </Property>
            <Property name="UML:Method">
                <Value>handlers:umlmethod</Value>
            </Property>
```

```
            </Property>
            <Property name="Converters">
                <!--  The following three converters are all applied to UML:Diagram handler.
                      The converter chains is
                      "converters:replaceproperties" -> "converters:diagramsize"
                      -> "converters:diagramtitle" -> handler
                 -->
                <Property name="UML:Diagram">
                    <Value>converters:diagramtitle</Value>
                </Property>
                <Property name="UML:Diagram">
                    <Value>converters:diagramsize</Value>
                </Property>
                <Property name="UML:Diagram">
                    <Value>converters:replaceproperties</Value>
                </Property>

                <Property name="UML:Attribute">
                    <Value>converters:renamedatatype</Value>
                </Property>
                <Property name="UML:Parameter">
                    <Value>converters:renamedatatype</Value>
                </Property>
                <Property name="UML:Operation">
                    <Value>converters:ignoreoperationinmethod</Value>
                </Property>
            </Property>
        </Config>
    </CMConfig>
```

*3.2.2 Configuration parameters for **DiagramTitleConverter** class*

Name	Description	Required/Type/ Default Value
ConvertDiagramTitleToTextElement	Whether to convert the diagram title to text node, true to convert	Optional/Boolean /false
DefaultTextElementProperties	The properties of converted text node	Optional/ Configuration properties
DefaultTextElementProperties .<PropertyName>	The sub properties' names are the keys of properties of text node	Required/String
DefaultTextElementProperties .<PropertyValue>	The sub properties' values are the values of properties of text node	Required/String

Sample Configuration:

```
<CMConfig>
    <Config name=”com.topcoder.umltool.xmiconverters.poseidon5”>
        <Property name=”ConvertDiagramTitleToTextElement”>
            <Value>true </Value>
        </Property>
        <Property name=”DefaultTextElementProperties”>
            <Property name=” FILL_COLOR”>
                <Value>CCFFCC</Value>
            </Property>
            <Property name=”FONT_FAMILY”>
                <Value>Arial</Value>
            </Property>
        </Property>
    </Config>
</CMConfig>
```

*3.2.3 Configuration parameters for **GraphPropertiesConverter** class*

Name	Description	Required/Type/ Default Value
ObjectFactoryNamespace	The namespace of ObjectFactory used in this component.	Required/String
ToReplacePropertyKeys	The key-to-key pairs which need to be replaced.	Optional/ Configuration properties
ToReplacePropertyKeys .<PropertyName>	The sub properties’ names are the keys of properties in the diagram to be replaced in ZUML	Required/String
ToReplacePropertyKeys .<PropertyValue>	The sub properties’ values are the new keys of properties in the diagram	Required/String
PropertyValueTransformers	The property value transformers. A list of transformer object name in Object Factory, the object name’s format is “key:identifier” or “key”	Optional/String[]

Sample Configuration:

```
<CMConfig>
    <Config name=”com.topcoder.umltool.xmiconverters.poseidon5”>
        <Property name=”ObjectFactoryNamespace”>
            <Value>object_factory_namespace</Value>
```

```
            </Property>
             <Property name=” PropertyValueTransformers”>
                  <Value>transformer:all</Value>
                  <Value>transformer:color</Value>
             </Property>
             <Property name=” ToReplacePropertyKeys”>
                  <Property name=”fill”>
                       <Value>FILL_COLOR</Value>
                  </Property>
                  <Property name=” font-color”>
                       <Value>FONT_COLOR</Value>
                  </Property>
             </Property>
        </Config>
    </CMConfig>
```

*3.2.4 Configuration parameters for **RenameConverter** class*

Name

Description

Required/Type/

Default Value

ToRenameNames

The name-to-name pairs. The key is the name in ZUML file, the value is the name in TCUML file.

Optional/

Configuration properties

ToRenameNames

.<PropertyName>

The sub properties’ names are the names in ZUML file

Required/String

ToRenameNames

.<PropertyValue>

The sub properties’ values are the names in TCUML file

Required/String

Sample Configuration:

```
<CMConfig>
    <Config name=”com.topcoder.umltool.xmiconverters.poseidon5”>
        <Property name=”ToRenameNames”>
            <Property name=”UML2:TypedElement.type”>
                <Value>UML:StructuralFeature.type </Value>
            </Property>
            <Property name=”UML:Class”>
                <Value>UML:Classifier </Value>
            </Property>
            <Property name=”UML:DateType”>
                <Value>UML:Classifier </Value>
            </Property>
        </Property>
    </Config>
</CMConfig>
```

3.3 Dependencies Configuration

3.3.1 You need to configure the Object Factory correctly.

Sample Configuration:

```
<CMConfig>
    <Config name=”object_factory_namespace”>
        <Property name=”handlers:xmiheader”>
            <Property name="type">
                <Value>com.topcoder.umltool.xmiconverters.poseidon5
                    .handlers.ZUMLHeaderHandler</Value>
            </Property>
            <Property name="params">
                <Property name="param1">
                    <Property name="name">
                        <Value>acceptableVersions</Value>
                    </Property>
                </Property>
            </Property>
        </Property>
        <Property name=”acceptableVersions”>
            <Property name="arrayType">
                <Value>String</Value>
            </Property>
            <Property name="dimension">
                <Value>1</Value>
            </Proeprty>
            <Property name="values">
                <Value>{“1.4.5”, “1.4.6”}</Value>
            </Property>
        </Property>
        <Property name=”handlers:umlmethod”>
            <Property name="type">
                <Value>com.topcoder.umltool.xmiconverters.poseidon5
                    .handlers.ZUMLMethodHandler</Value>
            </Property>
        </Property>
        <!—converters -->
        <Property name=”handlers:umlmethod”>
            <Property name="type">
                <Value>com.topcoder.umltool.xmiconverters.poseidon5
                    .converters.DiagramTitleConverter</Value>
            </Property>
        </Property>
        ….
    </Config>
</CMConfig>
```

3.3.2 You need to configure XMI Reader 1.0 correctly

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

- Configure XMI Reader 1.0
- Configure the Object Factory used in this component

- Configure the classes in this component.

4.3 Demo

4.3.1 Set up this component

```
See 4.2 to configure this component and dependent components.
```

4.3.2 Easy Usage, do the post reading tasks in the converters

Assume the DiagramSizeConverter, DiagramTitleConverter and GraphPropertiesConverter are configured in namespace "com.topcoder.umltool.xmiconverters.poseidon5"

```
// create XMI Reader
XMIReader reader = new XMIReader();
// configure XMI reader with the additional handlers and converters via XMIConvertersUtil, with
default namespace
// "com.topcoder.umltool.xmiconverters.poseidon5"
XMIConvertersUtil.config(reader);

// configure XMI reader with given namespace
XMIConvertersUtil.config(reader, "custom_namespace");

// parse the ZUML file
reader.parseZipFile("sample.ZUML");
```

4.3.3 Not use the converters to do the post reading tasks

```
// follow the demo 4.3.2, the only difference is that DiagramSizeConverter, DiagramTitleConverter and
// GraphPropertiesConverter are not configured.

// then use XMIConvertersUtil to do the post reading tasks.
// get the diagrams from model manager.
Diagram[] diagrams = modelManager.getDiagrams();
// for each diagram in the diagrams,
// remove title of diagram
XMIConvertersUtil.removeDiagramTitle(diagram);

// or convert the title of diagram with specific properties
XMIConvertersUtil.convertDiagramTitleToText (diagram, properties);
// if the ("FILL_COLOR", "000000") pair is in the properties.
// the converted text node has the property ("FILL_COLOR", "000000").
// the converted text node has the text same as displayed in Poseidon, the position and size are also un-
changed.

// adjust the size of diagram
XMIConvertersUtil.adjustGraphNodeSize(diagram);

// transform the matched properties' values
List<PropertyValueTransformer> transformers = new ArrayList<PropertyValueTransformer>();
transformers.add(new RegexToLowerPropertyValueTransformer("font-.*"));
transformers.add(new RegexToLowerPropertyValueTransformer("fill"));
XMIConvertersUtil.transformPropertyValues(diagram, transformers);
// this will transform the values of "font-xxx" and "fill" properties to lower case.

// replace the property keys
XMIConvertersUtil.replacePropertyKeys(diagram, toReplacePropertyKeys);
// if the ("fill", "FILL_COLOR") pair is in the toReplacePropertyKeys
// the "fill" properties in any element in diagram will be replaced by "FILL_COLOR".
```

5. Future Enhancements

Add more XMIHandler and XMIConverter instances to convert ZUML to TCUML in future.