XMI Writer UML Activity Graph Plugin 1.0 Component Specification

1. Design

The XMI Writer UML Activity Graph Plugin component provides the ability to write the Activity Graph structure to XMI format. The transformer will be invoked by the XMI Writer with the root Activity Graph element. It will write the response to the output print stream. For the general purpose, this component is designed to accept any TC UML model elements as long as it is properly configured.

The general idea is using reflection. First, there are too many kinds of model elements in the Activity Graph. This compels us to seek a generic way to do transformation. Second, the TC UML model is well defined, and it is well fitted into the OMG model. We can easily deduct the XMI element name from package name, and easily deduct attribute and child element name methods name.

Transforming process can be divided two parts. The first part is used to retrieve OMG XMI element description for specific TC model element. The second part is transform TC model element with the help of corresponding OMG XMI element description. Bridge between these two parts is XMIElementDescription, which contains XMI element name, a list of AttributeDescriptions, and a list of ChildDescriptions. AttributeDescription maps an XMI attribute name to a java.lang.Method, which can be used to retrieve the attribute value. ChildDescription maps an XMI child element name to a java.lang.Method, which can be used to retrieve the child value.

Description retrieving part is pluggable. Because we can retrieve description in various ways, which can be loading from configuration directly, or hard coding the description, or even parsing DTD. In current implementation, we use reflection to retrieve XMI element description. Element name can be deducted from interface name. Attributes' and child elements' names can be deducted from "getXXX()" and "isXXX()" method. Because the OMG DTD is so loose, we treat Enum, Boolean values as attributes, and String, model element and Collection values as child elements. Some "getXXX()" methods must be excluded with help from configuration. And some child elements are logically contained, while some are only referenced. We use a little configuration to distinguish them.

Element transforming part is hard coded in the transformer, because DTD is fixed. We should always transform element to same format. If one wants to use different transform mechanism. He/she should define another plugin. With the help of XMI description, it is somewhat easy to transform Activity Graph element to XMI. The general idea is transforming Enum, Boolean and String directly, and transforming model element recursively.

Above two parts are described very detailed in Section 1.3.

Thread safety is not an issue for this component. Please see 1.6 for explanation.

1.1 Design Patterns

Strategy Pattern:

XMIDescription in ActivityGraph2XMITransformer is used as a strategy. It is configurable and pluggable.

This components itself is also a strategy for XMI Writer component. It acts as a plugin for writer.

1.2 Industry Standards

UML 1.5

XMI 1.2

1.3 Required Algorithms

NOTE, in the following algorithm explanation, every step has an example, which is only a fragment from XMI file. I don't wan to paste the completed XMI file here. Developers should look through some XMI files several times before going on. Be familiar with XMI first!

1.3.1 Build XMIElementDescription via reflection

Because TC UML Activity Graph Model is well defined, it fits OMG DTD very well. We can use reflection to get XMI element name for model classes, and also retrieve attributes and nested element names.

The basic idea is as follows. Every getXXX() or isXXX() method can be expected to return an attribute or child node of XMI element defined by given TC UML element. We can deduct the attribute name or child name from the method name. Note, another key point here is we use methods but not field names, because field names may be changed un-expectedly.

The first question is how to distinguish attribute and child node? Fortunately, OMG DTD is very loose. Every attribute can be written as a child element, and also most of child elements can be written as attributes. The simplest way is to write every attribute as child. But this way would make the resulting XMI look weird. We just pick simple values like Enum and Boolean to be written as attribute.

The second question comes from logical contained child and referenced child. To solve this problem, we use some pre-defined configurations. A list of contained node names is provided for each element via configuration. All the other nodes are referenced by default.

Unfortunately, TC UML Model is too powerful. It holds references between classes, which are not defined in OMG DTD. The third problem is we must exclude some getXXX() methods. Again, we use pre-defined configurations to solve this problem. A list of excluded methods is provided for each element via configuration.

The last problem belongs to method names. Methods which return a collection is named with plural. We use a tricky method to deal with it. Every getXXAs() method has a corresponding addXXB() method. E.g. For getStimulus() method, there is a corresponding method named as addStimuli(). From "getStimulus", we can get "addStimu". And then try to find method name beginning with this word, which is "addStimuli". At last step, we just remove "add" from head. That's our expected result.

Variables used in this algorithm:

type (INPUT) : Class -- TC UML model element type, for which we find nodes, and attributes.

excludedMethods : Map<String, Set<String>> -- A predifined field which maps a interface name to a set of excluded methods in the interface.

containedElements : Map<String, Set<String>> -- A predifined field which maps a interface name to a set of element names which is logically contained.

xmiElements : Map<Class, XMIElement> -- A cache which holds the already retrieved XMIElement.

KNOWN_PACKAGE: String -- A constant value like "com.topcoder.uml". We only support type from this package and its sub-packages.

Following is the algorithm's work flow.

1) Try to retrieve implemented interface of type if type is not a interface.

```
if (type is not an interface) {
  foreach (interface in type.getInterfaces()) {
    if (interface.getPackge().getName().startsWith(KNOWN_PACKAGE)) {
      type = interface; break; } }
}
if (type.getPackage().getName().startsWith(KNOWN_PACKAGE) == false
  return null;
}
```

2) If given type is cached in xmiElements, directly return the value is cache.

3) Element name should be "UML:" + type.getSimpleName(), "UML:Classifier".

And create XMIElementDescription with the name and type.

4) Iterate through every methods to get attributes and nested children.

```
foreach method in type.getMethods() {
  // ignore methods with parameter
  if (method parameters array length is not zero) {
    continue;
  }

  // ignore excluded methods
  if (method name is in excludedMethods) {
    continue;
  }

  // treat "isXXX" as an attribute
  if (method name starts with "is", like "isLeaf") {
    Create AttributeDescription(method, method.getName())
    Add the attribute to the created XMIElement is step 3.
```

```
  continue;
}

// ignore methods not start with "get"
if (method name doesn't start with "get") {
  continue;
}

Class returnType := method.getReturnType();

// treat enum value as an attribute
if (returnType is kind of Enum, like "getVisibility") {
  Remove "get" prefix, and make 1st letter lowercase, like "visibility"
  Add the AttributeDescription(name retrieved in above step, method)
  continue;
}

// treat string, collection and other types as child element
if (returnType is kind of String, like "getNamespace") {
  Remove "get" prefix, and make 1st letter lowercase, like "namesapce"
}
else if (returnType is kind of Collection, like "getStimulus") {
  Find method starts with "addStimu", like "addSimuli"
  If return type of "addXXX" is not in KNOWN_PACKAGE, continue;
  Remove "add" prefix, and make 1st letter lowercase, like "stimuli"
}
else if (returnType is in KNOWN_PACKAGE, like "getMultiplicity") {
  Remove "get" prefix, and make 1st letter lowercase, like "multiplicity"
}
else
  continue;

// add string, collection and other types as child element
Get the declaring class of method, like owner := method.getDeclaringClass
Get child name as "UML:" + owner.getSimpleName + [.name],
  like "UML:ModelElement.namespace"
```

```
    If (returnType is not a String) {
      Get the reference flag, like
containedElement.get("ModelElement").get("Namespace")
    }
    Add ChildDescription(child name, method, reference flag) to XMIElement.
  }
```

5) Put the type/XMIElement in xmiElements cache, and return the element.

1.3.2 Transform Model Element to XMI

Because of the XMIElementDescription element built in above section, it is somewhat easier to transform model element to XMI. The general idea of this algorithm is transforming Enum, Boolean and String directly, and transforming model element recursively.

The last thing to note before we go to the algorithm work flow is that we should deal with all kinds of values for both attributes and child elements, although in the above section attributes only contain Enum and Boolean, and child elements only contain String and model element. This is because the XMIDescription is pluggable, we can't expect what kind of value is for the attributes or children.

Variables and common routines (All defined in transformer class):

xmiDescription : XMIDescription -- An XMIDescription used to retrieve XMIElementDescription

retrieveId(Object) : String -- Retrieve id for the element

buildIDREFS(Collection) : String -- Build IDREFS.

(See http://www.w3.org/TR/REC-xml/#idref)

normalizeString(String) : String -- Replace escape characters in String.

(See http://www.w3.org/TR/REC-xml/#dt-escape)

normalizeEnum(Enum) : String -- Make the first character lowercase. (Required by DTD)

Following is the algorithm work flow.

Input: 1) **element** -- the model element

2) **isReference** -- a flag indicating whether a reference should be written.

3) **writtenElements** -- the set of elements which have been logically written.(Not a reference)

To implement transform() method in transformer, just pass *[element, true, new HashSet()]* to this algorithm.

1) Retrieve element description from xmiDescription:

```
elementDesc = xmiDescription.getXMIElementDescription(element)
if (elementDesc == null) {
  According to ignoreUnknownElement flag,
  throw UnknownElementTypeException or return directly
}
```

2) Determine whether the element is logically written twice. This can also avoid cyclic dependency.

```
If (isReference == false ) {
  if (writtenElements.contains(element) throw DuplicateElementException
  else writtenElements.add(element);
}
```

3) If the isReference flag is true, simply write:

```
<elementDesc.getElementName() xmi.idref='retrieveId()'/>
```

Like:

```
<UML:Classifier xmi.idref='a435baasds45435' />
```

And directly return.

4) Start the element as:

```
"<" + elementDesc.getElementName(). Like "<UML:Classifier"
```

5) Write XMI id as an attribute:

```
xmi.id='retrieveId(element)'
```

6) Write attributes:

```
foreach (attribute : elementDesc.getAttributes()) {

  // retrieve the attribute value in model element
  Object value = attribute.getValue(element);
  if (value == null) {
    continue;
  }

  // covert the value to string
```

```
if (value instanceof String) {
  attrStr = normalizeString(value);
}
else if (value instanceof Enum) {
  attrStr = normalizeEnum(value);
}
else if (value instanceof Boolean) {
  attrStr = value.toString();
}
else if (value instanceof Collection) {
  attrStr = buildIDREFS(value);
}
else {
  attrStr = retrieveId(value);
}

// output the attribute with attribute.getName() and attrStr
// like isLeaf='true'
```

7) Close the start tag with ">"

8) Write child elements:

```
foreach (child : elementDesc.getChildren()) {

  // retrieve the child value in model element
  Object value = child.getValue(element);
  if (value == null) {
    continue;
  }

  // write the element, childName below is from child.getName()
  if (value instanceof String) {
    <childName>normalizeString(value)</childName>
  }
  else if (value instanceof Enum) {
    <childName xmi.value='normalizeEnum(value)' />
```

```
}
else if (value instanceof Boolean) {
  <childName xmi.value='value.toString()' />
}
else if (value instanceof Collection) {
  <childName>
    foreach (childElement in value) {
      Recursively call this algorithm with childElement.
      NOTE, child.isReference() flag should be passed.
    }
  </childName>
}
else {
  <childName>
    Recursively call this algorithm with value.
    NOTE, child.isReference() flag should be passed.
  </childName>
}
```

9) End the element as:

```
</elementDesc.getElementName()>, like </UML:Classifier>
```

1.4 Component Class Overview

ActivityGraph2XMITransformer

It is the core class of this component. It can be used to transform Model element to its XMI representation.

This class holds an XMIDescription instance, from which it can retrieve XMIElementDescription for specific model element. Because of the XMIElementDescription, the transforming process is rather straightforward. We just get the XMI element name, all the attributes and child elements from the description, and print it one by one. All the simple values are printed directly, like Boolean, Enum, String. For single model element value, and model element value contained in Collection, we directly print the id reference if it is contained as an attribute. And if it is contained as a child element, we just call the transform method recursively.

Besides that, “ignoreUnknownElementType” flag is defined in this class. It is used to indicate whether unknown element type should be ignored. If it is true, unknown element type will be ignored, otherwise, exception would occur.

This class is mutable, thus not thread safe.

XMIElementDescription

This class describes an XMI element. It contains two simple attributes, the element type in TC UML Model, and the element name in OMG DTD. It also contains a list of AttributeDescriptions, which contains information for the attributes this element, and a list of ChildDescriptions, which contains information for the child elements of this element.

Typically, the XMIElementDescription instances will be created by XMIDescription implementations, and be used by ActivityGraph2XMITransformer to help transform the UML Model to XMI.

This class is mutable, thus not thread-safe.

NodeDescription <<abstract>>

This abstract class describes an XMI node. As convention, node can be attribute or element. It contains two attributes, a String representing node name, and a java.lang.reflect.Method representing the method whose returned value will be used as node value. Besides that, it also defines a "getValue" method which uses reflection to retrieve node value.

This class is immutable, thus thread-safe.

ChildDescription

This class describes an XMI nested child element. It extends from NodeDescription class, and adds a reference flag. So it contains two attributes, name and methods as which in NodeDescription. And the additional reference flag is used to indicate whether this child should be stored as a reference or not.

This class is immutable, thus thread-safe.

AttributeDescription

This class describes an XMI attribute. It extends from NodeDescription class without any additional methods. So it contains two attributes, name and methods as which in NodeDescription.

This class is immutable, thus thread-safe.

XMIDescription <<interface>>

This interface defines a contract to retrieve XMIElementDescription for some element type in TC UML Model. The implementation can load all the descriptions from configuration file, try to parse OMG DTD, or even hardcode the descriptions. Currently, an implementation which uses reflection with a few configurations is provided.

Implementation of interface is not required thread-safe.

ReflectedXMIDescription [xmidescription sub-package]

An implementation of XMIDescription interface. It uses reflection to retrieve XMI descriptions for given TC UML element type. This implementation assumes TC UML Model is well fitted into OMG DTD.

It uses the "getXXX" methods to determine the element attributes and children name. The methods which return Boolean or Enum will be mapped to an

attribute. The methods which return String, Collection or other objects will be mapped to a child element. Please see CS for details.

Because some additional methods are defined in TC UML Model, which are not defined in OMG DTD, we would exclude such methods via configuration. And some children are stored by references, the configuration is also used to indicate whether some child is logically contained or not. Please see CS and sample configuration file for details.

The XMIElementDescription is retrieved lazily, and also cached.

This class is not thread-safe, since it is mutable.

1.5 Component Exception Definitions

DescriptionRetrievalException [extends XMITransformException]:
This exception is used to indicate any error while retrieving XMI description.

This exception can be thrown from XMIDescription.getXMIElementDescription(Class) method. All implementation of the interface may throw this exception.

NodeValueRetrievalException [extends XMITransformException]:
This exception is used to indicate any error while retrieving node value.

This exception can be thrown from NodeDescription.getValue(Object) method. It will be used to wrap method invoking exception.

UnknownElementTypeException [extends XMITransformException]:
This exception is used to indicate some model element type is unknown by the transformer.

This exception can be thrown from ActivityGraph2XMITransformer.transform (Object, java.io.PrintStream) method as kind of XMITransformException.

DuplicateElementException [extends XMITransformException]:
This exception is used to indicate some model element is logically contained more than once.

This exception can be thrown from ActivityGraph2XMITransformer.transform (Object, java.io.PrintStream) method as kind of XMITransformException.

ConfigurationException [extends BaseException]:
This exception is used to indicate any error in configuration. The error can be internal error in ConfigurationManager component, or invalid configuration value.

This exception can be thrown from ReflectedXMIDescription's constructor and ActivityGraph2XMITransformer's constructor.

XMITransformException [form XMI Writer component]:
This exception is used to indicate any error in transforming process. Actually, all the above exceptions except ConfigurationException extend this exception.

This exception can be thrown from ActivityGraph2XMITransformer.transform (Object, java.io.PrintStream).

1.6 Thread Safety

This component is not thread-safe. Each thread is expected to create its own ActivityGraph2XMITransformer object. As the ActivityGraph2XMITransformer object is quite lightweight, so create multiple such objects won't cause any performance issues. Another reason is transforming is usually a time-consuming process. Synchronization is not very acceptable. And there is no such requirement too, because writing process is

usually done in a single thread.

Only NodeDescription, AttributeDescription, and ChildDescription are immutalbe. They are simple classes acting like value holding entities. All the other classes are mutable in this component, thus they are not thread-safe.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- **Config Manager 2.1.5**
  It is used in ReflectedXMIDescription's constructor to exclude some methods and to determine which children are logically contained. And it is also used in ActivityGraph2XMITransformer's constructor to load configuration to create XMIDescription instance.

- **XMI Writer 1.0**
  This design is a plugin to XMI Writer component, and ActivityGraph2XMITransformer extends from AbstractXMITransformer in the component.

- **Base Exception 1.0**
  BaseException in this component is extended by ConfigurationException.

- **Object Factory 2.0.1**
  It is used to create XMIDescription instance in ActivityGraph2XMITransformer's constructor.

- **UML Model - Activity Graphs 1.0**
  UML Model - Common Behavior 1.0
  UML Model - Actions 1.0
  UML Model - Core Dependencies 1.0
  UML Model - State Machines 1.0
  UML Model - Core Extension Mechanisms 1.0
  UML Model - Collaborations 1.0
  UML Model - Core Auxiliary Elements 1.0
  UML Model - Data Types 1.0
  UML Model - Core Classifiers 1.0
  UML Model - Core 1.0
  UML Model - Core Relationships 1.0
  UML Model - Use Cases 1.0
  A series of UML Model components can be used by this component. And usually, all of them are used, because the model graph includes almost all the components.

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.xmi.writer.transformers.model

com.topcoder.uml.xmi.writer.transformers.model.xmidescription

3.2 Configuration Parameters

3.2. 1 Configuration for ActivityGraph2XMITransformer

Parameter	Description	Values
ignoreUnknownElementType	Represents the flag indicating whether unknown element type should be ignored. It is Required.	Boolean value
objectFactory	Represents a ConfigManager namespace used to initialized ObjectFactory instance. It is REQUIRED.	com.topcoder. objectfactory
xmiDescription	Represents the XMIDescription object name in ObjectFactory. It is REQUIRED.	xmiDescriptionObject

3.2.2 Configuration for ReflectedXMIDescription

Parameter	Description	Values
[ModelElementName]. excludedMethods	Represents a list of excluded methods for the specific model element. It is OPTIONAL.	getElementImports, getReferenceTags
[ModelElementName]. contianedNodes	Represents a list of logically contained node for the specific model element. It is OPTIONAL.	templateParameter, taggedValue

Comments on configuration of ReflectedXMIDescription:

1) All “getXXX()” methods can be divided into three categories: excluded methods, contained node methods, and referenced node methods. If we configure two categories of them, the other category comes out automatically. Actually, the most straightforward way to configure this class is defining “referencedNodes” and “containedNodes”. But after comparing XSD and TC UML Model, we can find that most of the methods define referenced nodes. We want to minimize the configuration task, so we configure “excludeMethods” and “containedNodes”, which need much less configuration.
2) In this section, we will give a sample to find excluded methods. For example, there is a method called “getElementImports()” in ModelElement class, and there is no attribute or child element defined as “elementImport” for ModelElement in DTD. As a result,

"getElementImports" should be excluded. As a reminder, most of the classes don't have excluded methods.

3) In this section, we will give a way to find contained nodes. If some name is defined in some element's child elements list, but it is not defined in the attributes list, it should be a contained node. Let's take ModelElement as an example again. "taggedValue" is defined in the child elements list, but it is not defined in attributes list of ModelElement in DTD. So it should be a contained node. "Stereotype" is defined in both child list and attribute list, so it should only be a referenced node. You can also note that type of "stereotype" attribute is IDREFS.

Please see sample.xml in docs directory for sample configuration.

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

Follow 3.2 to configure this component.

4.3 Demo

4.3.1 Create the transformer instance.

```
// use configuration to initialize

XMITransformer transformer = new ActivityGraph2XMITransformer(<<namespace in
ConfigManager>>);

// use XMIDescription instance to initialize

XMITransformer transformer = new
ActivityGraph2XMITransformer(<<XMIDescription instance>>, true);
```

4.3.2 Use the API directly

```
// transform an UML model element

transformer.transform(<<element>>, System.out);
```

4.3.3 Use this component as a plugin of XMI Writer component.

```
Map<TransformerType,XMITransformer> transformers = new
Map<TransformerType,XMITransformer>();

transformers.put(TransformerType.Model, transformer);
```

```
// ... add other transformers ...

XMIWriter writer = new XMIWriter(<<manager>>, transformers);

// plugin would be called automatically
writer.transform(<<outputStream>>, true);
```

4.3.4 Change the Ignore-Unknown-Element-Type Option

```
transformer.setIgnoreUnknownElementType(true);
// unknown element type will be silently ignored.

transformer.setIgnoreUnknownElementType(true);
// unknown element type will introduce some error.
```

5. Future Enhancements

Add more implementations of XMIDescription interface to transform other kinds of elements..