XMI Writer 1.0 Component Specification

1. Design

The XMI Writer component provides the ability to write XMI files. The writer will provide a framework for element transformers according to their type. The transformers will transform the element they receive, by writing the response to an output print stream.

The XMIWriter main class in this design provides the ability to write XMI data, it is able to write data to different kinds of data stores: a XMI file, a zip file, and an OutputStream. And 4 types of XMITransformer are supported by this class to transform the XMI header, Model, ActivityGraphs and Diagrams data respectively.

Improvements:

XMIWriter class is able to write XMI data a zip file, and it also supports the XMI root attributes and the XMI header. All of them are not required by this component.

1.1 Design Patterns

Strategy Pattern – The XMITransformer interface implements this pattern so that we can plug in different implementations.

1.2 Industry Standards

UML 1.5

1.3 Required Algorithms

None.

1.4 Component Class Overview

1.4.1 Package com.topcoder.xmi.writer

- **XMIWriter**: XMIWriter class is able to write the XMI data retrieved from the UML Model Manager component to a file or an output stream, it is also able to write the XMI data to a zip file. Currently this writer only supports 4 types of transformers, and they would write the XMI header, Model, ActivityGraphs and the Diagrams respectively. The Model, ActivityGraphs and Diagrams data are all retrieved from the UML Model Manager component. This writer also supports configurable XMI root attributes, and it won't write the Diagrams data if the withDiagramData flag is false.
- **XMITransformer**: XMITransformer interface defines the contract to transform an element Object and write the transformation result into the out PrintStream. It also contains methods to set and access the XMIWriter it is registered to.
- **TransformerType**: TransformerType enum class defines the transformer types supported in this design, currently 4 types are supported: Model, ActivityGraph, Diagram and Header, each type corresponds to an XMITransformer object in the XMIWriter to transform the corresponding data.
- **AbstractXMITransformer**: AbstractXMITransformer abstract class implements the XMITransformer interface, and it provides implementation to the setXMIWriter & getXMIWriter from the interface. All XMITransformer implementations should extend this abstract class for convenience.

1.4.2 Package com.topcoder.xmi.writer.transformers.xmiheader

- **XMIHeaderTransformer**: XMIHeaderTransformer class extends AbstractXMITransformer abstract class, and it is registered into the XMIWriter with

the TransformerType.Header, and it is able to write the XMI header data to the output stream.

1.5 Component Exception Definitions

1.5.1 Custom Exceptions

- **ElementAlreadyExistsException**: ElementAlreadyExistsException is thrown by the XMIWriter and XMITransformer implementations if there is already an xmi id associated with the element to add.
- **UnknownElementException**: UnknownElementException is thrown by the XMIWriter and XMITransformer implementations if there is no xmi id defined for the requested element object.
- **XMITransformException**: XMITransformException is thrown by the XMIWriter and XMITransformer implementations if any error occurs during the transformation. It is mainly to wrap the underlying exceptions other than the I/O related exceptions.
- **XMIWriterException**: XMIWriterException is the base-exception for all custom exceptions defined in this design.

1.5.2 System Exceptions

- **IllegalArgumentException**: Used wherever empty String argument is used while not acceptable. Normally an empty String is checked with trimmed result. It also thrown when null argument is passed in or some other cases when the argument is invalid.
- **IOException**: It is thrown when I/O error occurs in XMIWriter or XMITransformer implementations.

1.6 Thread Safety

This component is not thread-safe, each thread is expected to create its own XMIWriter object, and register into its own XMITransformer objects in order to write the XMI data correctly. As the XMIWriter object is quite lightweight, so create multiple such objects won't cause any performance issues. And there is no such requirement too.

The XMIWriter class is mutable, and the XMITransformer's abstract implementation is mutable too, so they are both not thread-safe, and the same would apply to the XMITransformer's implementations as they are expected to extend the abstract implementation for convenience – the XMIHeaderTransformer is such an example.

2. Environment Requirements

2.1 Environment

Java 1.5

2.2 TopCoder Software Components

- **UML Model Manager 1.0**: The UMLModelManager class used in this design is from this component.
- **UML Model - Core 1.0**: Entity classes from this component are used by this design.
- **UML Model - Activity Graphs 1.0**: Entity classes from this component are used by this design.
- **UML Model - Model Management 1.0**: Entity classes from this component are used by this design.
- **Diagram Interchange 1.0**: Entity classes from this component are used by this design.

- **Base Exception 1.0**: The custom exceptions from this design extend the BaseExcetpion from this component.

NOTE: Configuration Manager is not used as some arguments cannot be easily created from the configured properties (e.g. the UMLModelManager object), and it is also more flexible to set the arguments directly to constructor rather than load them from Configuration Manager.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.xmi.writer

com.topcoder.xmi.writer.transformers.xmiheader

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

4.2 Required steps to use the component

None.

4.3 Demo

```
Assume the modelManager (UMLModelManager object), modelTransformer
(XMITransformer object), activityGraphTransformer (XMITransformer object),
diagramTransformer (XMITransformer object) are provided by application.
```

4.3.1 Create XMIWriter object to write the xmi data

```
// create a Map to hold all transformers
Map<TransformerType, XMITransformer> transformers =
       new HashMap<TransformerType, XMITransformer>();
transformers.put(TransformerType.Model, modelTransformer);
transformers.put(TransformerType.ActivityGraph, activityGraphTransformer);
transformers.put(TransformerType.Diagram, diagramTransformer);
transformers.put(TransformerType.Header,
       new XMIHeaderTransformer("TCUML", "1.0", "1.2"));

// create an XMIWriter with default xmi root attributes
XMIWriter writer = new XMIWriter(modelManager, transformers);

// transform to a file, with diagram data
writer.transform(new File("test.xmi"), true);

// the result file would be like:
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmlns:UML = 'org.omg.xmi.namespace.UML'
```

```
      xmlns:UML2 = 'org.omg.xmi.namespace.UML2'>
  <XMI.header>
   <XMI.documentation>
     <XMI.exporter>TCUML</XMI.exporter>
     <XMI.exporterVersion>1.0</XMI.exporterVersion>
     <XMI.metaModelVersion>1.2</XMI.metaModelVersion>
   </XMI.documentation>
 </XMI.header>
 <XMI.content>
    … xml data for model, activityGraphs, and diagrams
 </XMI.content>
</XMI>

// create an XMIWriter with custom xmi root attributes
Map<String, String> xmiRootAttrs = new HashMap<String, String>();
xmiRootAttrs.put("xmlns:UML", "org.omg.xmi.namespace.UML");
xmiRootAttrs.put("xmlns:UML2", "org.omg.xmi.namespace.UML2");
xmiRootAttrs.put("xmi.version", "1.2");
writer = new XMIWriter(modelManager, transformers, xmiRootAttributes);

// transform to a file, without diagram data
writer.transform(new File("test.xmi"), false);

// the result would be like:
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version='1.2' xmlns:UML = 'org.omg.xmi.namespace.UML'
      xmlns:UML2 = 'org.omg.xmi.namespace.UML2'>
  <XMI.header>
   <XMI.documentation>
     <XMI.exporter>TCUML</XMI.exporter>
     <XMI.exporterVersion>1.0</XMI.exporterVersion>
     <XMI.metaModelVersion>1.2</XMI.metaModelVersion>
   </XMI.documentation>
 </XMI.header>
 <XMI.content>
    … xml data for model, activityGraphs (without diagrams data)
 </XMI.content>
</XMI>
```

4.3.2 *Transform to different sources*

```
// the transformers variable are the same as the one in 4.3.1
// create an XMIWriter with default xmi root attributes
XMIWriter writer = new XMIWriter(modelManager, transformers);

// transform to a file
writer.transform(new File("test.xmi"), true);

// transform to a zip file
writer.transformToZipFile(new File("test.zuml"), true);

// transform to an OutputStream object
OutputStream output = new FileOutputStream("test.xmi");
writer.transform(output, true);
```

4.3.3 *Manage the xmi root attributes*

```
// the transformers variable are the same as the one in 4.3.1
```

```
// create an XMIWriter with default xmi root attributes
XMIWriter writer = new XMIWriter(modelManager, transformers);

// clear all xmi root attributes
writer.clearXMIRootAttributes();

// add xmi root attribute
writer.putXMIRootAttribute(“xmi.version”, “1.2”);
writer.putXMIRootAttribute(“timestamp”, “Mon Jan 30 16:36:02 CST 2006”);

// remove xmi root attribute
writer.removeXMIRootAttribute(“timestamp”);

// get xmi root attribute
String attrValue = writer.getXMIRootAttribute(“xmi.version”);

// get all xmi root attribute keys
String[] keys = writer.getXMIRootAttributeKeys();

// transform the xmi data
writer.transform(new File(“test.xmi”), true);

// the result would be like:
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version='1.2'>
  <XMI.header>
   <XMI.documentation>
     <XMI.exporter>TCUML</XMI.exporter>
     <XMI.exporterVersion>1.0</XMI.exporterVersion>
     <XMI.metaModelVersion>1.2</XMI.metaModelVersion>
   </XMI.documentation>
 </XMI.header>
 <XMI.content>
    … xml data for model, activityGraphs (without diagrams data)
 </XMI.content>
</XMI>
```

4.3.4 A dummy XMITransformer to call putElementId & getElementId

```
public class DummyXMITransformer extends AbstractXMITransformer {
     public DummyXMITransformer() {}

     public void transform(Object element, PrintStream out) {
          XMIWriter writer = getXMIWriter();

          // put element id and get element id
          writer.putElementId(element, “123566”);
          String xmiId = writer.getElementId(element);

          …
     }
}
```

5 Future Enhancements

Add more XMITransformer implementations.