UML Model – Core Extension Mechanisms 1.0 Component Specification

1. Design

The UML Model - Core Extension Mechanisms component declares the interfaces from the UML 1.5 framework, from the Core - Extension Mechanisms package. It provides concrete implementations for each interface and provides powerful API to access the collection attributes.

The Extension Mechanisms package in the UML Model is used to customize and extend the UML Model elements with new semantics by using stereotypes, tag definitions, and tagged values. The three interfaces Stereotype, TagDefinition, and TaggedValue provided by this component are for this purpose, and implementations are provided for them respectively in this design, please consult the class overview section below for more details.

1.1 Design Patterns

None.

1.2 Industry Standards

UML 1.5

1.3 Required Algorithms

None.

1.4 Component Class Overview

1.4.1 Package com.topcoder.uml.model.core.extensionmechanisms

- **Stereotype**: Stereotype interface extends the GeneralizableElement interface, it is mainly used to differentiate the model elements so that they could have different meanings or usages even if they have identical structure, the stereotype could specify additional tag definitions to the model elements such as the attributes, associations, operations etc. And a stereotype could also be a subclass of another stereotype, in this case, it will inherit all the constraints and tagged values from its super type, but the parent-child stereotype must have the same kind of base class.
- **TagDefinition**: TagDefinition interface extends the ModelElement interface, and it contains a set of the tagged value that can be attached to the model element. From the UML spec, it is used to define the meta attributes of the stereotype to which it is attached to, and in other words, the stereotype is the owner of the tag definition.
- **TaggedValue**: TaggedValue interface extends ModelElement interface, and it is used to attach information to model element in conformance with its tag definition. The information could contain code generation options, model management information, or user-specified semantics. The tagged value must be associated with a tag definition, and its reference values should follow the constraint set by the tagType of the tag definition it belongs to.
- **StereotypeImpl**: StereotypeImpl class implements Stereotype interface and extends the GeneralizableElementAbstractImpl abstract class.
- **TagDefinitionImpl**: TagDefinitionImpl class implements TagDefinition interface and extends the ModelElementAbstractImpl abstract class.
- **TaggedValueImpl**: TaggedValueImpl class implements the TaggedValue interface

and extends the ModelElementAbstractImpl abstract class.

1.5 Component Exception Definitions

1.5.1 Custom Exceptions

No custom exceptions are necessary for this component.

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.

1.6 Thread Safety

This component is not thread-safe, and there is no need to make it thread-safe, as for a component comprised of all entity classes, it is very unlikely to share entity objects in different threads, and it is normally left to the application to ensure the thread-safety if it is needed, rather than put the burden on the entity classes.

All classes in this component are mutable, so they are all not thread-safe.

2. Environment Requirements

2.1 Environment

Java 1.5

2.2 TopCoder Software Components

- **UML Model - Core Requirements Version 1.0**: This component provides the base interface and abstract class for this design.
- **UML Model - Data Types Version 1.0**: The Multiplicity interface used by this design is from this component.

NOTE: The other UML Model components are used indirectly by this design through the two directly dependent components above.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.model.core.extensionmechanisms

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

4.3.1 Create Stereotype and manage its attributes

```
 // create object
Stereotype stereotype = new StereotypeImpl();

// set and get baseClass attribute
stereotype.setBaseClass("Method");
String baseClass = stereotype.getBaseClass();

// manage defined tags
TagDefinition tag1 = new TagDefinitionImpl();
TagDefinition tag2 = new TagDefinitionImpl();

// add defined tag
stereotype.addDefinedTag(tag1);
stereotype.addDefinedTag(tag2);

// get all defined tags
Collection<TagDefinition> tagDefs = stereotype.getDefinedTags();

// check the existence of tag1, the result should be true
boolean existent = stereotype.containsDefinedTag(tag1);

// count the number of defined tags, the result should be 2
int ret = stereotype.countDefinedTags();

// remove defined tag
stereotype.removeDefinedTag(tag1);

// clear all defined tags
stereotype.clearDefinedTags();

// manage extended elements, MethodImpl class is from the UML
 // Model Core Requirement component.
ModelElement element1 = new MethodImpl();
ModelElement element2 = new MethodImpl();

// add extended element
stereotype.addExtendedElement(element1);
stereotype.addExtendedElement(element2);

// get all extended elements
```

```
Collection<ModelElement> elements =
 stereotype.getExtendedElements();

// check the existence of element1, the result should be true
existent = stereotype.containsExtendedElement(element1);

// count the number of extended elements, the result should be 2
ret = stereotype.countExtendedElements();

// remove extended elements
stereotype.removeExtendedElement(element1);

// clear all extended elements
stereotype.clearExtendedElements();

// create the object with another constructor
 stereotype = new StereotypeImpl("Method", tagDefs, elements);
```

4.3.2 Create TagDefinition and manage its attributes

```
 // create the object
TagDefinition tagDef = new TagDefinitionImpl();

// get/set tagType attribute
tagDef.setTagType("UML:Datatypes:String");
String tagType = tagDef.getTagType();

// get/set multiplicity attribute,
// MultiplicityImpl class is from the UML Data Types component
tagDef.setMultiplicity(new MultiplicityImpl());
Multiplicity multiplicity = tagDef.getMultiplicity();

// get/set its owner attribute
tagDef.setOwner(new StereotypeImpl());
Stereotype stereotype = tagDef.getOwner();

// manage its typedValues attribute
TaggedValue value1 = new TaggedValueImpl();
TaggedValue value2 = new TaggedValueImpl();

// add typed value
tagDef.addTypedValue(value1);
tagDef.addTypedValue(value2);

// get all typed values
Collection<TaggedValue> taggedValues = tagDef.getTypedValues();

// check the existence of value1, the result should be true
boolean existence = tagDef.containsTypedValue(value1);

// count the number of typed values, the result should be 2
int ret = tagDef.countTypedValues();

// remove the typed value
tagDef.removeTypedValue(value1);

// clear all typed values
tagDef.clearTypedValues();
```

```
    // create the object with another constructor
      tagDef = new TagDefinitionImpl(tagType, multiplicity, stereotype,
     taggedValues);
```

4.3.3 Create TaggedValue and manage its attributes

```
     // create the object
    TaggedValue taggedValue = new TaggedValueImpl();

    // get/set dataValue attribute
    taggedValue.setDataValue("true");
    String dataValue = taggedValue.getDataValue();

    // get/set type attribute
    taggedValue.setType(new TagDefinitionImpl());
    TagDefinition type = taggedValue.getType();
    // get/set modelElement attribute. MethodImpl class is from the
     // UML Model Core Requirement component.
    taggedValue.setModelElement(new MethodImpl());
    ModelElement element = taggedValue.getModelElement();

    // manage reference values, MethodImpl class is from the UML
     // Model Core Requirement component.
    ModelElement refValue1 = new MethodImpl();
    ModelElement refValue2 = new MethodImpl();

    // add reference value
    taggedValue.addReferenceValue(refValue1);
    taggedValue.addReferenceValue(refValue2);

    // get all reference values
    Collection<ModelElement> refValues =
taggedValue.getReferenceValues();

    // check the existence of refValue1, the result is true
    boolean existence = taggedValue.containsReferenceValue(refValue1);

    // count the number of reference values, the result should be 2
    int ret = taggedValue.countReferenceValues();

    // remove reference value
    taggedValue.removeReferenceValue(refValue1);

    // clear all reference values
     taggedValue.clearReferenceValues();
```

4.3.4 A simple customer scenario

Here is an example got from the UML Spec 1.5

Stereotype	Base Class	Tags	Description
persistent	Class	isPersistent	Classes of this stereotype may be persistent, depending on the value of the “isPersistent” tag.

Tag	Stereotype	Type	Multiplicity	Description
isPersistent	persistent	UML::Datatypes::Boolean	1	indicates whether the class is persistent or not

The two tables above contain the data representing a stereotype of Class with an associated tag definition.

Here is the code to create them:

```
 Stereotype stereotype = new StereotypeImpl();
// this method is from its base-class
stereotype.setName("ersistent");
stereotype.setBaseClass("Class");

TagDefinition tagDef = new TagDefinitionImpl();

// this method is from its base-class
tagDef.setName("isPersistent");
tagDef.setTagType("UML::Datatypes::Boolean");

// Multiplicity, MultiplicityRange and its implementation classes
// are all from the UML Model Data Types component.
Multiplicity multiplicity = new MultiplicityImpl();
MultiplicityRange range = new MultiplicityRangeImpl();
range.setLower(1);
range.setUpper(1);
multiplicity.setRange(range);

tagDef.setMultiplicity(multiplicity);

// add tagged value to this tagDef
TaggedValue value = new TaggedValueImpl();
value.setDataValue("true");
tagDef.addTypedValue(value);

// add this tagDef to the stereotype finally
 stereotype.addDefinedTag(tagDef);
```

5 Future Enhancements

Support for UML 2.0 could be added in the future.