UML Model – Core Auxiliary Elements 1.0 Component Specification

1. Design

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

1.1 Design Patterns

None

1.2 Industry Standards

UML 1.5

1.3 Required Algorithms

There are no complex algorithms in this design.

1.4 Component Class Overview

Comment
This interface extends ModelElement interface. The ModelElement interface comes form the Core Requirements component. A comment is an annotation attached to a model element or a set of model elements. It has no semantic force but may contain information useful to the modeler.

CommentImpl
This is a simple concrete implementation of Comment interface and extends ModelElementAbstractImpl from the Core Requirements component. As such, all methods in Comment are supported.

TemplateParameter
Simple, base interface. Defines the relationship between a template (a ModelElement) and its parameter (a ModelElement). A ModelElement with at least one templateParameter association is a template (by definition). In the metamodel, TemplateParameter reifies the relationship between a ModelElement that is a template and a ModelElement that is a dummy placeholder for a template argument.

TemplateParameterImpl
This is a simple, concrete implementation of TemplateParameter interface.

TemplateArgument
Simple, base interface. Reifies the relationship between a Binding and one of its arguments (a ModelElement).

TemplateArgumentImpl

This is a simple, concrete implementation of TemplateArgument interface.

1.5 Component Exception Definitions

This component defines no custom exceptions.

The general approach to parameter handling is not to do it. The architectural decision was to allow the beans to hold any state, and delegate to the users of these beans to decide what is legal and when it is legal. The exception here is the collection attributes. They will not allow null elements to be passed.

1.6 Thread Safety

This component is not thread-safe, and there is no requirement for it to be thread-safe. In fact, the PM discourages method synchronization. Thread safety will be provided by the application using these implementations.

The classes are made non-thread-safe by the presence of mutable members and collections. In order to provide thread-safety, if that is ever desired, all simple member accessors and collections would need to be synchronized.

2. Environment Requirements

2.1 Environment

JDK 1.5

2.2 TopCoder Software Components

- TC UML Core Requirements 1.0
    - TC UML component defining the Core Requirements.
- TC UML Core Dependencies 1.0
    - TC UML component defining the Core Dependencies.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.topcoder.uml.model.core.auxiliaryelements

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

The demo will demonstrate the usage of these beans. It will show them being instantiated, then used via their interface. This will be the typical usage of such simple entities under any scenario. This demo will focus on showing how a simple and collection attribute is managed, with the understanding that all other attributes are managed in exactly the same manner, and therefore not shown here.

4.3.1 Instantiation

Create an instance of sample entity: `TemplateParameter`. All other concrete entities are instantiated in this manner and are not shown here.

```
// Create an instance of sample entity
TemplateParameter templateParameter = new TemplateParameterImpl();
```

4.3.2 Simple attribute management

Manage a simple attribute: `TemplateParameter.parameter`. All other simple attributes are managed in this manner and are not shown here.

```
// Create sample entity with a simple attribute to manage
TemplateParameter templateParameter = an instance of TemplateParameterImpl

// Use setter
ModelElement parameter = some valid value
templateParameter.setParameter(parameter);

// Use getter
ModelElement retrievedParameter = templateParameter.getParameter();
```

4.3.3 Collection attribute management

Manage a collection attribute: `Comment.annotatedElements`.

```
// Create sample entity with a collection attribute to manage
Comment comment = an instance of CommentImpl

// Use single-entity add method
ModelElement ann1 = some valid annotatedElement
comment.addAnnotatedElement(ann1);
// There is now one annotatedElement in the collection

// Use multiple-entity add method
```

```
Collection<ModelElement> col1 = collection with 5 valid annotatedElements
comment.addAnnotatedElements(col1);
// There will now be 6 annotatedElements in the collection

// Use contains method to check for annotatedElement presence
boolean present = comment.containsAnnotatedElement(ann1);
// This will be true

// Use count method to get the number of annotatedElements
int count = comment.countAnnotatedElements();
// The count will be 6

// Use single-entity remove method
boolean removed = comment.removeAnnotatedElement(ann1);
// This will be true, and the collection size is 5, regardless
// if ann1 has duplicates in this collection.

// Use multiple-entity remove method
Collection<ModelElement> col2 = collection with 3 valid annotatedElements,
which is a subset of col1.
boolean altered = comment.removeAnnotatedElements(col2);
// This will be true, and the collection size is 2

// Use clear method
comment.clearAnnotatedElements();
// The collection size is 0 and contains no annotatedElement
```

5. Future Enhancements

Providing a complete model, or moving to UML 2.