UML Model - Core Auxiliary Elements 1.0 Requirements Specification

1. Scope

1.1 Overview

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.

1.2 Logic Requirements

1.2.1 Package interfaces

The component will declare the interfaces from the Core - Auxiliary Elements package.

1.2.2 Interface implementations

For each interface, it will provide an implementation with the name <InterfaceName>Impl in case it is concrete, or <InterfaceName>AbstractImpl in case it is abstract. There in no difference between the two types of implementations, except the abstract modifier and the protected constructors for the abstract classes.

The interfaces and the classes will stay in the same package.

All the classes are concrete classes.

1.2.3 Constructor

The classes will provide a constructor with no arguments, which should be protected for the abstract classes. Other constructors are at the designers' choice.

1.2.4 Simple attributes

For the simple attributes, the interfaces will provide a getter and a setter. Nulls and empty strings should be accepted as valid values.

1.2.5 Collection attributes

For the collection attributes, the interfaces will provide a powerful API to access them:

- addElement(Element)
- removeElement(Element):boolean
- clearElements()
- getElements():Collection<Element>
- contains(Element):boolean
- countElements():int

1.2.6 List attributes

For the list attributes (the ordered attributes), there will be some extra methods:

- addElement(Element)
- addElement(index, Element)
- setElement(index,Element)
- removeElement(index):Element

- removeElement(Element):boolean
- clearElements()
- getElements():List<Element>
- contains(Element):boolean
- indexOf(Element):int
- countElements():int

The designer may provide other operations also, in case he feels they are necessary.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool as part of the UML Model.

1.5 Future Component Direction

Providing a complete model, or moving to UML 2.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The design must follow the interfaces found in the class diagram with the component interfaces. The designer is encouraged to add to the existing interfaces, but not to remove anything. Note that the interface fields from the diagrams should be expanded as mentioned above.

2.1.3 Environment Requirements

- Development language: Java 1.5
- Compile target: Java 1.5

2.1.4 Package Structure

com.topcoder.uml.model.core.auxiliaryelements

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

None

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None

3.2.2 TopCoder Software Component Dependencies:

- Other UML Model components

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.