Diagram Interchange 1.0 Requirements Specification

1. Scope

1.1 Overview

The Diagram Interchange component declares the interfaces from the Diagram Interchange 2.0 framework. It provides concrete implementations for each interface and provides powerful API to access the collection attributes.

1.2 Logic Requirements

1.2.1 Class implementations

Each interface in the Diagram Interchange will be a concrete class, as there are no multiple inheritances. All the classes will stay in the same package.

1.2.2 Constructor

The classes will provide a constructor with no arguments. Other constructors are at the designers' choice.

There are two limitations:

- the Image class should not inherit from the Image interface from Infrastructure::Profiles
- the CoreSemanticModelBridge should not be implemented, or it should not have an association with the Object interface from MOF::Reflection.

1.2.3 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.

1.2.4 Collection attributes

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

- addElement(ElementType)
- removeElement(ElementType):boolean
- clearElements()
- getElements():Collection< ElementType >
- containsElement(ElementType):boolean
- countElements():int

1.2.5 List attributes

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

- addElement(ElementType)
- addElement(index, ElementType)
- setElement(index, ElementType)
- removeElement(index): ElementType
- removeElement(ElementType):boolean
- clearElements()

- getElements():List< ElementType >
- containsElement(ElementType):boolean
- indexOfElement(ElementType):int
- countElements():int

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

The difference between List and Collection is given by the {ordered} attribute on the associations in the Diagram Interchange picture.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool to represent the diagrams.

1.5 Future Component Direction

None.

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. The designer is encouraged to add to the existing interfaces, but not to remove anything.

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.topcoder.diagraminterchange

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:

- UML Model - Core 1.0

**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.