Diagram Interchange 1.0 Component Specification

1. Design

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

For more information on the usage of the Diagram Interchange model, see its OMG specification (http://www.omg.org).

All the concrete classes can be extended in an extra-package to accommodate further functionalities.

All the classes in this component are serializable.

1.1 Design Patterns

None.

1.2 Industry Standards

UML 1.5.

1.3 Required Algorithms

There were no required algorithms in the requirements and the development of this component is straightforward. It is for this reason that no sequence diagram is provided with this design.

1.4 Component Class Overview

1.4.1 Package com.topcoder.diagraminterchange

Public class GraphNode extends GraphElement:

This class and GraphEdge are the core classes of Diagram Interchange and every visible model element is represented by them. The base class of GraphNode and GraphEdge is GraphElement.This class is mutable and is not thread-safe.

Public class GraphEdge extends GraphElement:

This class and GraphNode are the core classes of Diagram Interchange and every visible model element is represented by them. The base class of GraphEdge and GraphNode is GraphElement. A GraphEdge has two GraphConnectors which are called anchors and are its connection end point. This class is mutable and is not thread-safe.

Public abstract class GraphElement extends DiagramElement:

GraphElements are linked via GraphConnector. A GraphElement can own any number of GraphConnectors, called anchorages. GraphElement extends DiagramElement. A GraphElement can contain any number of DiagramElements. This class is abstract but has no abstract methods. This class is mutable and is not thread-safe.

Public abstract class DiagramElement implements Serializable:

This is the base class of GraphElement, LeafElement and Reference. This class is

abstract but has no abstract methods. This class is mutable and is not thread-safe.

Public class Diagram extends GraphNode:

This class represents a Diagram, that is a special GraphNode. This is the topmost GraphElement of any graph or diagram. This class is mutable and is not thread-safe.

Public abstract class LeafElement extends DiagramElement:

A LeafElement is used when a model element has an optional representation. A LeafElement can be an Image, a TextElement or a GraphicPrimitive. This class is abstract but has no abstract methods. This class is thread-safe because it has no attributes, but subclasses extending this one are not required to be thread-safe.

Public class Image extends LeafElement:

This class represents an Image. It references an image file. This class is mutable and is not thread-safe.

Public class TextElement extends LeafElement:

This class represents a TextElement, which can be used for the text part of a model element. This class is mutable and is not thread-safe.

Public abstract class GraphicPrimitive extends LeafElement:

This class is the base for all GraphicPrimitives. This specification define the Polyline and the Ellipse, but other custom classes can extend this class in an extra-package. This class is abstract but has no abstract methods. This class is thread-safe because it has no attributes, but subclasses extending this one are not required to be thread-safe.

Public class Polyline:

This class represents the Polyline graphic primitive, that is a line passing through all ordered waypoints. This class is mutable and is not thread-safe.

Public class Ellipse:

This class represents the Ellipse graphic primitive. This class is mutable and is not thread-safe.

Public class DiagramLink implements Serializable:

This class is used to link GraphElements with Diagrams. This class is mutable and is not thread-safe.

Public class GraphConnector implements Serializable:

GraphConnector links GraphElement instances. This class allows linking of a GraphEdge with a GraphNode or another GraphEdge; GraphConnector does not permit two GraphNode instances to be linked. This class is mutable and is not thread-safe.

Public class Reference extends DiagramElement:

A DiagramElement can contain references to other diagram element. This class is mutable and is not thread-safe.

Public class Property implements Serializable:

Properties specify the appearance of DiagramElements. There is a set of standard properties but non-standard ones could be added too. This class is mutable and is not thread-safe.

Public abstract class SemanticModelBridge:

This class is used for different purposes. This is the base class of SimpleSemanticModelElement, Uml1SemanticModelBridge and CoreSemanticModelBridge. This class is abstract but has no abstract methods. This class is mutable and is not thread-safe.

Public class SimpleSemanticModelElement extends SemanticModelBridge:

This class is used to represent elements with a simple semantic model. This class is mutable and is not thread-safe.

Public class Uml1SemanticModelBridge extends SemanticModelBridge:

This class is used to represent elements that have an UML semantic model. This class is mutable and is not thread-safe.

Public class CoreSemanticModelBridge extends SemanticModelBridge:

This class is used to represent elements that have a core semantic model. This class is thread-safe because it has no attributes, but subclasses extending this one are not required to be thread-safe.

Public class Point implements Serializable:

This is the basic implementation of Point. This class is mutable and is not thread-safe.

Public class BezierPoint extends Point:

This is the basic implementation of BezierPoint. This class is mutable and is not thread-safe.

Public class Dimension implements Serializable:

This is the basic implementation of Dimension. This class is mutable and is not thread-safe.

1.5 Component Exception Definitions

None.

1.6 Thread Safety

The classes in this component are not thread-safe, since thread safety is not required. All the classes, except CoreSemanticModelBridge, are mutable: this is the main reason why this component is not thread safe. Instead, the calling application is required to ensure thread safety, for example by declaring all methods synchronized.

2. Environment Requirements

2.1 Environment

At minimum, Java 1.5 is required for compilation and executing test cases.

2.2 TopCoder Software Components

UML Model Core 1.0. The Uml1SemanticModelBridge has an association with the Element class.

NOTE: The default location for TopCoder Software component jars is ../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

`com.topcoder.diagraminterchange`

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

4.2 Required steps to use the component

- Extract the component distribution.
- Follow the demo

4.3 Demo

This demo show how to use the API to instantiate a class of this component and how to access its simple attributes, collection attributes and lists attributes.

For this purpose, GraphEdge is taken as example, but the same API is used for all attributes of this component classes.

```
// Create a new instance
GraphEdge edge = new GraphEdge();

// Access simple attributes
edge.setPosition(Point);
```

```
Point edgePosition = edge.getPosition();

// Access collection attributes
edge.addLink(DiagramLink1);
edge.addLink(DiagramLink2);
boolean hasBeenRemoved = edge.removeLink(DiagramLink1);
Collection<DiagramLink> edgeLinks = edge.getLinks();
boolean isIn = edge.containsLink(DiagramLink2);
int howMany = edge.countLinks();
edge.clearLinks();

// Access list attributes
edge.addWaypoint(BezierPoint2);
edge.addWaypoint(BezierPoint3);
edge.addWaypoint(2, BezierPoint4);
edge.setWaypoint(0, BezierPoint1);
Point removed = edge.removeWaypoint(3); // Remove BezierPoint4
boolean hasBeenRemoved = edge.removeWaypoint(BezierPoint2);
List<Point> edgeWaypoints = edge.getWaypoints();
boolean isIn = edge.containsWaypoint(BezierPoint1);
int index = edge.indexOf(BezierPoint1);
int howMany = edge.countWaypoints();
edge.clearWaypoints();
```

5 Future Enhancements

Provide more specific implementations of the interfaces of Diagram Interchange Framework 2.0.