UML_Model Core_Relationships 1.0 Component Specification

1. Design

The UML Model - Core Relationships component declares the interfaces from the UML 1.5 framework, from the Core - Relationships 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

No algorithms are necessary as the interactions are very simple.

1.4 Component Class Overview

Class Name:

Association
This interface defines the contract for a association An association defines a semantic relationship between classifiers. An Association has at least two AssociationEnds. Each end is connected to a Classifier - the same Classifier may be connected to more than one AssociationEnd in the same Association. The Association represents a set of connections among instances of the Classifiers. This interface extends the GeneralizableElement and Relationship interfaces and add more specific methods that apply to a association Implementations are not required to be thread safe.

AssociationImpl
This is the default implementation of the Association interface. An association defines a semantic relationship between classifiers. An Association has at least two AssociationEnds. Each end is connected to a Classifier - the same Classifier may be connected to more than one AssociationEnd in the same Association. The Association represents a set of connections among instances of the Classifiers. This class also extends GeneralizableElementAbstractImpl to reuse the code that exists in the abstract class. This class has a connection list for which a powerful api is provided. This class is mutable and not thread safe. Since it is a data class it doesn't really makes sense to make it thread safe.

Relationship
This interface defines the contract for uml relationships. A relationship is a connection among model elements. It simply extends ModelElement interface and adds no new method. Implementations of this interface are not required to be thread safe.

RelationshipAbstractImpl
This abstract class represent an abstract implementation of the Relationship interface. A relationship is a connection among model elements. It extends the abstract ModelElementAbstractImpl which has all the methods from ModelElement implemented (no code is necessary in this class as everything is implemented in ModelElementAbstractImpl abstract class). This class is abstract as it makes no sense for it to be concrete. Concrete implementations are not required to be thread safe.

Generalization
This interface defines the contract for a generalization. A generalization is a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element (it has all of its properties, members, and relationships) and may contain additional information. This interface extends Relationship interface and add more specific methods that apply to a generalization Implementations are not required to be thread safe.

GeneralizationImpl
This is the default implementation of the Generalization interface. A generalization is a taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element (it has all of its properties, members, and relationships) and may contain additional information. This class also extends RelationshipAbstractImpl to reuse the code that exists in the abstract class. This class has several simple attributes(discriminator,child, parent, powertype) that describe a generalization. For these simple attributes setters and getters are provided. This class is mutable and not thread safe. Since it is a data class it doesn't really makes sense to make it thread safe.

AssociationClass
This interface defines the contract for an association class. An association class is an association that is also a class. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not any of the classifiers. This interface extends Association and Class interfaces and add no other methods. Implementations are not required to be thread safe.

AssociationClassImpl
This is the concrete implementation of the AssociationClass interface. An association class is an association that is also a class. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not any of the classifiers. This class also extends RelationshipAbstractImpl to reuse the code that exists in the abstract class. It has an association field of Association type in order to reuse the code because ,in Java, a concrete class cannot extend two classes. This class has several simple attributes(discriminator,child, parent, powertype) that describe a generalization. For these simple attributes setters and getters are provided. This class is mutable and not thread safe. Since it is a data class it doesn't really makes sense to make it thread safe.

AssociationEnd
This interface defines the contract for an association end. An association end is an endpoint of an association, which connects the association to a classifier. Each association end is part of one association. It extends ModelElement interface and add more specific methods that apply to an association end. Implementations are not required to be thread safe.

AssociationEndImpl
This is the default implementation of the AssociationEnd interface. An association end is an endpoint of an association, which connects the association to a classifier. Each association end is part of one association. This class also extends ModelElementAbstractImpl to reuse the code that exists in the abstract class. This class has several simple attributes(isNavigable, ordering, aggregation, targetKind, multiplicity, changeability, association, participant) that describe an association end. For these simple attributes setters and getters are provided. This class also provides a list attribute(qualifiers) and a collection attribute(specifications) for which a more powerful api is provided. This class is mutable and not thread safe. Since it is a data class it doesn't really makes sense to make it thread safe.

1.5 Component Exception Definitions

None defined.

1.6 Thread Safety

The component is not thread safe because all the classes are mutable. It doesn't really make sense to make this component thread safe since all the classes are data classes. The application should ensure that the classes are used in a thread safe manner. To make the component thread safe, we will have to synchronize all the methods and this will only add an unnecessary overhead.

2. Environment Requirements

2.1 Environment

Java 1.5.

2.2 TopCoder Software Components

Uml Model Data Types 1.0

Uml Model Classifiers 1.0

Uml Model Core Requirements 1.0

2.3 Third Party Components

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.model.core.relationships

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

See demo.

4.3 Demo

The demo will show the api that is provided for a list attribute, a collection attribute and a simple attribute.

1)Demo for the api of a list attribute; Association class is used to show this

```
Association association = new AssociationImpl();
```

```
// add a connection
AssociationEnd connection1 = new AssociationEndImpl();
association.addConnection(connection1);

// add another connection at the beginning of the list
AssociationEnd connection2 = new AssociationEndImpl();
association.addConnection(0, connection2);

// set another connection on the second position of the list
AssociationEnd connection3 = new AssociationEndImpl();
association.setConnection(1, connection3);

// get the number of the connections; the size=2
System.out.println("size=" + association.countConnections());

// get the index of connection3; it will be 1
System.out.println("index=" + association.indexOfConnection(connection3));

// check if connection3 is contained; it will print true
System.out.println(association.containsConnection(connection3));

// get a copy of the connection list
List<AssociationEnd> conn = association.getConnections();

// get the size of the conn; the size=2
System.out.println("size=" + conn.size());

// get the index of connection3; it will be 1
System.out.println("index=" + conn.indexOf(connection3));

// check if connection3 is contained; it will print true
System.out.println(association.containsConnection(connection3));

// remove connection3; it will print true
System.out.println(association.removeConnection(connection3));

// remove the connection at index 0
AssociationEnd ae = association.removeConnection(0);

// the returned value should be connection2. It will print true
System.out.println(ae == connection2);

// to clear all associations from the list the following call can be made
association.clearConnections();
```

2)Demo for a collection attribute; AssociationEnd class is used

```
AssociationEnd ae = new AssociationEndImpl();

// add a 2 specifications
Classifier specification1 = new ClassifierImpl();
ae.addSpecification(specification1);
Classifier specification2 = new ClassifierImpl();
ae.addSpecification(specification2);

// count specifications; it will print 2
System.out.println(ae.countSpecifications());
// check if specification2 is contained; it will print true
System.out.println(ae.containsSpecification(specification2));

// get a collection of all specifications
Collection<Classifier> specs = ae.getSpecifications();

// get the size of specs; it will print 2
System.out.println(specs.size());
// check if specification1 is contained; it will print true
System.out.println(specs.contains(specification1));
// check if specification2 is contained; it will print true
System.out.println(specs.contains(specification2));

// remove specification2; it will print true
System.out.println(ae.removeSpecification(specification2));

// clear all specifications
ae.clearSpecifications();
```

3)Demo for a simple attribute; Generalization class is used

```
Generalization generalization = new GeneralizationImpl();

//set the discriminator field; all the three calls are valid
generalization.setDiscriminator(" ");
generalization.setDiscriminator(null);
generalization.setDiscriminator("test");

//get the discriminator field
System.out.println(generalization.getDiscriminator());
```

4)AssociationClassImpl has an extra constructor (added for convenience)

```
Association association = new AssociationImpl();
```

```
AssociationClass ac=new AssociationClassImpl(association);
```

5. Future Enhancements

None.