UML Model – Collaborations 1.0 Component Specification

1. Design

The UML Model – Collaborations component declares the interfaces from the UML 1.5 framework, from the Collaborations 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

Message
This interface extends ModelElement interface. The ModelElement interface comes form the Core Requirements component. A message defines a particular communication between instances that is specified in an interaction

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

CollaborationInstanceSet
This interface extends ModelElement interface. The ModelElement interface comes form the Core Requirements component. A collaboration instance set references a set of instances that jointly collaborate in performing the particular task specified by the collaboration of the collaboration instance. The instances in the collaboration instance set play the roles defined in the collaboration.

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

Collaboration
This interface extends Namespace and GeneralizableElement interfaces. The Namespace and GeneralizableElement interfaces come form the Core Requirements component. A collaboration describes how an operation or a classifier, like a use case, is realized by a set of classifiers and associations used in a specific way. The collaboration defines a set of roles to be played by instances and links, as well as a set of interactions that define the communication between

the instances when they play the roles. In the metamodel, a Collaboration contains a set of ClassifierRoles and AssociationRoles, which represent the Classifiers and Associations that take part in the realization of the associated Classifier or Operation. The Collaboration may also contain a set of Interactions that are used for describing the behavior performed by Instances conforming to the participating ClassifierRoles.

CollaborationImpl
This is a simple concrete implementation of Collaboration interface and extends GeneralizableElementAbstractImpl from the Core Requirements component. To facilitate complete implementation of methods in the interface, the methods in Namespace interface are implemented but all they do is defer to in internal concrete implementation of that Namespace. In fact, it will be a simple inner concrete extension of NamespaceAbstractImpl, which also comes from the Core Requirements component. As such, all methods in Collaboration are supported.

NamespaceImpl
Inner class that represents a concrete extension of Namespace. It simply provides a concrete wrapper of the NamespaceAbstractImpl for this class so it can make use of the logic in the Namespace branch of implementation.

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 Common Behavior 1.0
    - TC UML component defining the Common Behavior.

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

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.topcoder.uml.model.collaborations.collaborationroles
com.topcoder.uml.model.collaborations.collaborationinstances
com.topcoder.uml.model.collaborations.collaborationinteractions

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: `Message`. All other concrete entities are instantiated in this manner and are not shown here.

```
// Create an instance of sample entity
Message message = new MessageImpl();
```

4.3.2 Simple attribute management

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

```
// Create an instance of sample entity
Message message = new MessageImpl();
```

```
// Create sample entity with a simple attribute to manage
Stimulus conformingStimulus = new MockStimulus();

// Use setter
message.setConformingStimulus(conformingStimulus);
// Use getter
Stimulus retrievedConformingStimulus = message.getConformingStimulus();
```

4.3.3 *CollaborationInstanceSetImpl attribute management*

Manage a collection attribute: `CollaborationInstanceSet.participatingLinks`. All other collection attributes are managed in this manner and are not shown here.

```
// Create sample entity with a collection attribute to manage
CollaborationInstanceSet collaborationInstanceSet = new
CollaborationInstanceSetImpl();

// Create sample entity with a simple attribute to manage
Collaboration collaboration = new CollaborationImpl();

// Use setter
collaborationInstanceSet.setCollaboration(collaboration);
// Use getter
Collaboration retrievedCollaboration =
collaborationInstanceSet.getCollaboration();

// Use single-entity add method
Instance instance1 = new MockInstance();
collaborationInstanceSet.addParticipatingInstance(instance1);
// There is now one supplier in the collection
// Use multiple-entity add method
Collection<Instance> col1 = new ArrayList<Instance>();
Instance instance2 = new MockInstance();
Instance instance3 = new MockInstance();

// collection with 3 valid suppliers
col1.add(instance1);
col1.add(instance2);
col1.add(instance3);
collaborationInstanceSet.addParticipatingInstances(col1);

// There will now be 4 suppliers in the collection
// Use contains method to check for supplier presence
// This will be true
boolean present1 =
collaborationInstanceSet.containsParticipatingInstance(instance1);
boolean present2 =
collaborationInstanceSet.containsParticipatingInstance(instance2);
boolean present3 =
collaborationInstanceSet.containsParticipatingInstance(instance3);

// Use count method to get the number of suppliers
// The count will be 4
int count = collaborationInstanceSet.countParticipatingInstances();

// Get the Collection
Collection<Instance> collection1 =
collaborationInstanceSet.getParticipatingInstances();

// Use single-entity remove method
```

```
// This will be true, and the collection size is 3
boolean removed =
collaborationInstanceSet.removeParticipatingInstance(instance1);


// if instance1 has duplicates in this collection.
// Use multiple-entity remove method
Collection<Instance> col2 = new ArrayList<Instance>();
col2.add(instance2);
col2.add(instance3);

// This will be true, and the collection size is 1
boolean altered =
collaborationInstanceSet.removeParticipatingInstances(col2);

// Use clear method
// The collection size is 0 and contains no suppliers
collaborationInstanceSet.clearParticipatingInstances();


// Use single-entity add method
Link client1 = new MockLink();
collaborationInstanceSet.addParticipatingLink(client1);
// There is now one supplier in the collection
// Use multiple-entity add method
Collection<Link> col3 = new ArrayList<Link>();
Link client2 = new MockLink();
Link client3 = new MockLink();

// collection with 3 valid suppliers
col3.add(client1);
col3.add(client2);
col3.add(client3);
collaborationInstanceSet.addParticipatingLinks(col3);

// There will now be 4 suppliers in the collection
// Use contains method to check for supplier presence
// This will be true
present1 = collaborationInstanceSet.containsParticipatingLink(client1);
present2 = collaborationInstanceSet.containsParticipatingLink(client2);
present3 = collaborationInstanceSet.containsParticipatingLink(client3);

// Use count method to get the number of suppliers
// The count will be 4
count = collaborationInstanceSet.countParticipatingLinks();

// Get the Collection
Collection<Link> collection2 =
collaborationInstanceSet.getParticipatingLinks();

// Use single-entity remove method
// This will be true, and the collection size is 3
removed = collaborationInstanceSet.removeParticipatingLink(client1);


// if client1 has duplicates in this collection.
// Use multiple-entity remove method
Collection<Link> col4 = new ArrayList<Link>();
col4.add(client2);
col4.add(client3);

// This will be true, and the collection size is 1
altered = collaborationInstanceSet.removeParticipatingLinks(col4);
```

```
// Use clear method
// The collection size is 0 and contains no suppliers
collaborationInstanceSet.clearParticipatingLinks();
```

4.3.4 *Collaboration attribute management*

Manage a collection attribute: `CollaborationInstanceSet.participatingLinks`. All other collection attributes are managed in this manner and are not shown here.

```
// Create sample entity with a collection attribute to manage
Collaboration collaboration = new CollaborationImpl();

// Use single-entity add method
CollaborationInstanceSet collaborationInstanceSet1 = new
CollaborationInstanceSetImpl();
collaboration.addCollaborationInstanceSet(collaborationInstanceSet1);
// There is now one supplier in the collection
// Use multiple-entity add method
Collection<CollaborationInstanceSet> col1 = new
ArrayList<CollaborationInstanceSet>();
CollaborationInstanceSet collaborationInstanceSet2 = new
CollaborationInstanceSetImpl();
CollaborationInstanceSet collaborationInstanceSet3 = new
CollaborationInstanceSetImpl();

// collection with 3 valid suppliers
col1.add(collaborationInstanceSet1);
col1.add(collaborationInstanceSet2);
col1.add(collaborationInstanceSet3);
collaboration.addCollaborationInstanceSets(col1);

// There will now be 4 suppliers in the collection
// Use contains method to check for supplier presence
// This will be true
boolean present1 =
collaboration.containsCollaborationInstanceSet(collaborationInstanceSet1);
boolean present2 =
collaboration.containsCollaborationInstanceSet(collaborationInstanceSet2);
boolean present3 =
collaboration.containsCollaborationInstanceSet(collaborationInstanceSet3);

// Use count method to get the number of suppliers
// The count will be 4
int count = collaboration.countCollaborationInstanceSets();

// Get the Collection
Collection<CollaborationInstanceSet> collection1 =
collaboration.getCollaborationInstanceSets();

// Use single-entity remove method
// This will be true, and the collection size is 3
boolean removed =
collaboration.removeCollaborationInstanceSet(collaborationInstanceSet1);


// if collaborationInstanceSet1 has duplicates in this collection.
// Use multiple-entity remove method
Collection<CollaborationInstanceSet> col2 = new
ArrayList<CollaborationInstanceSet>();
col2.add(collaborationInstanceSet2);
```

```
col2.add(collaborationInstanceSet3);

// This will be true, and the collection size is 1
boolean altered = collaboration.removeCollaborationInstanceSets(col2);

// Use clear method
// The collection size is 0 and contains no suppliers
collaboration.clearCollaborationInstanceSets();

// Create sample entity with a simple attribute to manage
Classifier classifier = new MockClassifier();

// Use setter
collaboration.setRepresentedClassifier(classifier);
// Use getter
Classifier retrievedClassifier = collaboration.getRepresentedClassifier();


// Create sample entity with a simple attribute to manage
Operation operation = new MockOperation();

// Use setter
collaboration.setRepresentedOperation(operation);
// Use getter
Operation retrievedOperation = collaboration.getRepresentedOperation();

// Create sample entity with a collection attribute to manage
CollaborationImpl collaborationImpl = new CollaborationImpl();

// Use single-entity add method
ModelElement modelElement1 = new MockModelElement();
ModelElement modelElement2 = new MockModelElement();
ModelElement modelElement3 = new MockModelElement();
// There is now one ModelElement in the collection
collaborationImpl.addOwnedElement(modelElement1);
collaborationImpl.addOwnedElement(modelElement2);
// Use contains method to check for supplier presence
// This will be true
present1 = collaborationImpl.containsOwnedElement(modelElement1);
present2 = collaborationImpl.containsOwnedElement(modelElement2);
present3 = collaborationImpl.containsOwnedElement(modelElement3);

// Use count method to get the number of suppliers
// The count will be 4
count = collaborationImpl.countOwnedElements();

// Get the Collection
Collection<ModelElement> collection4 =
collaborationImpl.getOwnedElements();

// Use single-entity remove method
// This will be true, and the collection size is 3
removed = collaborationImpl.removeOwnedElement(modelElement1);

// Use clear method
// The collection size is 0 and contains no suppliers
collaborationImpl.clearOwnedElements();
```

5. Future Enhancements

Providing a complete model, or moving to UML 2.