UML Model Manager 1.0 Component Specification

1. Design

The UML Model Manager component provides one place where the UML Model, the ActivityGraphs and the Diagrams are kept. The component will be used in the TopCoder UML Tool as the central place for the model.

The UMLModelManager contains two ordered lists (using ordered lists and the fact that they are additional functionality is stated by PM in https://software.topcoder.com/forum/c_forum_message.jsp?f=24459277&r=24487343).

The two ordered lists used here (activityGraphs can contain ActivityGraph elements and diagrams can contain Diagram elements) can be accessed thru the required methods:

- `addElement(Element):void`
- `removeElement(Element):boolean`
- `getElements():List<Element>`

Also an additional feature is provided thru a powerful API used for an easy manipulation of the 2 lists mentioned:

- `addElement(index, Element):void`
- `addElements(Collection<Element>):void`
- `addElements(index, Collection<Element>):void`
- `setElement(index,Element):void`
- `removeElement(index):Element`
- `removeElements(Collection<Element>):boolean`
- `clearElements():void`
- `containsElement(Element):boolean`
- `indexOfElement(Element):int`
- `countElements():int`
- isElementsEmpty():boolean

If the element that should be added to the list already exist in the list do nothing, add only Elements that does not exist in the list to ensure that duplicates are not presented.

UMLModelManager contains several constructors (that initialize the Model field with default empty Model) useful for a fast construction of UMLModelManager instances using received parameter(s).

1.1 Design Patterns

- The **Singleton Pattern** is used in this design in the main class of this component UMLModelManager (actually it is a **pseudo-singleton** because public constructors are provided not only **public static** *getInstance*() method).

1.2 Industry Standards

- UML 1.5 – This component use ActivityGraph and Model classes from UML 1.5 framework.

1.3 Required Algorithms

There are no complicated algorithms in this design, only simple null checking, setters/getters and method calls from **java.util.ArrayList.**

1.4 Component Class Overview

- **public class UMLModelManager**:

Usage: This class provides a pseudo singleton instance, for easy access inside the application, as the normal usage involves just one model. This class keeps a Model instance, accessible through a getter. The model is initialized in the constructor with a default empty Model (new ModelImpl()). A list of ActivityGraph instances (an ordered list) is kept in this class and it is accessible through a getter. There are also methods from a powerful API that manipulates the activityGraphs list that supports ActivityGraph instances. Another list of Diagram instances (an ordered list) is kept in this class and it is accessible same through a getter. There are also methods from a powerful API that manipulates the diagrams list that supports Diagram instances. Also this class keeps a ProjectConfigurationManager instance, accessible through a getter.

1.5 Component Exception Definitions

There are no custom exceptions added in this design because of the simplicity of the component, the following exceptions from **java.lang** are enough to do a good error handling:

- **exception java.lang.IllegalArgumentException:**

This exception is used to check whether the parameters received by methods are appropriate (majority should be not null - only setProjectLanguage(String) can receive null parameter - and all Strings should not be empty).

- **exception java.lang.IndexOutOfBoundsException**

This exception is thrown if index is out of range (index < 0 || index > size of the list).

- **exception java.lang.IllegalStateException**

This exception is thrown in getProjectConfigurationManager if ProjectConfigurationManager instance was not yet set.

1.6 Thread Safety

This component is not thread safe. Thread safety was not a requirement for this component.

Thread safety can be achieved by using external synchronization schemes (PM stated in other UML Tool components that this approach will be used for many components from UML Tool), to assure something like a transaction. An atomic operation doesn't make sense for what the user is concerned (in most cases).

UMLModelManager class contains mutable fields and therefore it is not thread safe.

- **Note:** Even the thread-safety is not a requirement, one aspect regarding thread-safety should be ensured: thread-safety should be provided at the time of *getInstance*() if a new instance is created.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

Related UML Tool components like:

- **UML Model – Model Management 1.0** – the UMLModelManager class from this component keeps a Model instance.
- **UML Model – Activity Graphs 1.0** – the UMLModelManager class from this component keeps a list with ActivityGraph elements.
- **Diagram Interchange 1.0** – the UMLModelManager class from this component keeps a list with Diagram elements.
- **UML Project Configuration 1.0** – the UMLModelManager class from this component keeps a ProjectConfigurationManager instance.

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.

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

- com.topcoder.uml.modelmanager

3.2 Configuration Parameters

- None.

3.3 Dependencies Configuration

All the UML Tool components used should be configured properly.

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

Configure properly the components from Dependencies Configuration.

4.3 Demo

It is not very easy to create a demo for a component that contains only one concrete class and no major interactions. The demo of this component is simple (the component is simple itself) so the following scenarios from **Demo.java** can be encountered:

```
package com.topcoder.uml.modelmanager;

import com.topcoder.diagraminterchange.Diagram;
import com.topcoder.uml.model.activitygraphs.ActivityGraph;
import com.topcoder.uml.model.activitygraphs.ActivityGraphImpl;
import com.topcoder.uml.model.modelmanagement.Model;
import
com.topcoder.uml.projectconfiguration.ProjectConfigurationManager;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;


public class Demo {

private UMLModelManager umlModelManager;
private ProjectConfigurationManager projectConfigurationManager = new
ProjectConfigurationManager();
private Model model;

private String instanceName = "New Instance";
private String projectLanguage = "Java";
private int index = 0;
private int count = 0;
private boolean status = false;

private ActivityGraph activityGraph = new ActivityGraphImpl();
private Collection<ActivityGraph> activityGraphs = new
ArrayList<ActivityGraph>();
private List<ActivityGraph> activityGraphList = new
ArrayList<ActivityGraph>();

private Diagram diagram = new Diagram();
private Collection<Diagram> diagrams = new ArrayList<Diagram>();
private List<Diagram> diagramList = new ArrayList<Diagram>();

public Demo() {
}

public void test() {


// 1. Creation of UMLModelManager

// 1.1 Create a new UMLModelManager using a instance name
umlModelManager = new UMLModelManager(instanceName);
// umlModelManager can also be created using new
UMLModelManager(newInstance);
// new UMLModelManager(newInstance, projectConfigurationManager);
// new UMLModelManager(newInstance, projectConfigurationManager,
projectLanguage);

// 1.2 umlModelManager can be created using public static getInstance()
method
umlModelManager = UMLModelManager.getInstance();


// 2. Get the Model instance
model = umlModelManager.getModel();
```

```
// 3. Get the ProjectConfigurationManager instance

projectConfigurationManager =
umlModelManager.getProjectConfigurationManager();


// 4. Get instance name
instanceName = umlModelManager.getInstanceName();


// 5. Manage project language
// 5.1 Set project language
umlModelManager.setProjectLanguage(projectLanguage);

// 5.2 Retrieve project language
projectLanguage = umlModelManager.getProjectLanguage();


// 6. Manage ActivityGraphs list

// 6.1 Register a new ActivityGraph at the end of the activityGraphs
list
umlModelManager.addActivityGraph(activityGraph);

// 6.2 Register a new ActivityGraph at the specified position in the
activityGraphs list
umlModelManager.addActivityGraph(index, activityGraph);

// 6.3 Register a collection of ActivityGraphs in the activityGraphs
list
umlModelManager.addActivityGraphs(activityGraphs);

// 6.4 Register a collection of ActivityGraphs at the specified
position in the activityGraphs list
umlModelManager.addActivityGraphs(index, activityGraphs);

// 6.5 Set a ActivityGraph at the specified position in the
activityGraphs list
umlModelManager.setActivityGraph(index, activityGraph);

// 6.6 Unregister a ActivityGraph from specified position in the
activityGraphs list
umlModelManager.removeActivityGraph(index);

// 6.7 Unregister specified ActivityGraph from the activityGraphs list
umlModelManager.removeActivityGraph(activityGraph);

// 6.8 Unregister a Collection of ActivityGraphs from the
activityGraphs list
umlModelManager.removeActivityGraphs(activityGraphs);

// 6.9 Clear all elements from the activityGraph list
umlModelManager.clearActivityGraphs();

// 6.10 Get the list of ActivityGraphs
```

```
activityGraphList = umlModelManager.getActivityGraphs();

// 6.11 Check if the specified ActivityGraph status in the
activityGraphs list
status = umlModelManager.containsActivityGraph(activityGraph);

// 6.12 Get the index where the given ActivityGraph exist in the
activityGraphs list
index = umlModelManager.indexOfActivityGraph(activityGraph);

// 6.13 Get the number of ActivityGraphs from the activityGraph list
count = umlModelManager.countActivityGraphs();

// 6.14 Check if the activityGraphs list is empty or not
status = umlModelManager.isActivityGraphsEmpty();


// 7. Manage Diagrams list

// 7.1 Register a new Diagram at the end of the diagrams list
umlModelManager.addDiagram(diagram);

// 7.2 Register a new Diagram at the specified position in the diagrams
list
umlModelManager.addDiagram(index, diagram);

// 7.3 Register a collection of Diagrams
umlModelManager.addDiagrams(diagrams);

// 7.4 Register a collection of Diagrams at the specified position in
the diagrams list
umlModelManager.addDiagrams(index, diagrams);

// 7.5 Set a Diagram at the specified position in the diagrams list
umlModelManager.setDiagram(index, diagram);

// 7.6 Unregister a Diagram from specified position in the diagrams
list
umlModelManager.removeDiagram(index);

// 7.7 Unregister specified Diagram from the diagrams list
umlModelManager.removeDiagram(diagram);

// 7.8 Unregister a Collection of Diagrams from the diagrams list
umlModelManager.removeDiagrams(diagrams);

// 7.9 Clear all elements from the diagram list
umlModelManager.clearDiagrams();

// 7.10 Get the list of Diagrams
diagramList = umlModelManager.getDiagrams();

// 7.11 Check if the specified Diagram status in the diagrams list
status = umlModelManager.containsDiagram(diagram);

// 7.12 Get the index where the given Diagram exist in the diagrams
list
```

```
index = umlModelManager.indexOfDiagram(diagram);

// 7.13 Get the number of Diagrams from the diagram list
count = umlModelManager.countDiagrams();

// 7.14 Check if the diagrams list is empty or not
status = umlModelManager.isDiagramsEmpty();

}
}
```

5. Future Enhancements

- None.