UML Project Configuration 1.0 Requirements Specification

1. Scope

1.1 Overview

The UML Project Configuration component provides the configuration for a UML project according to a specific language. It provides the standard set of stereotypes for different model element types, the standard namespaces and provides the ability to apply custom formatting to newly created model elements (standard constructors for exceptions ...) and diagram elements (color templates ...).

1.2 Logic Requirements

1.2.1 ProjectConfigurationManager

This manager acts as a facade of the component. It provides methods to retrieve the standard stereotypes for an element type and the standard namespaces, according to the language.

It also provides a method to apply initial formatting to a model element and a method to apply initial formatting to a diagram element. These methods return a boolean flag indicating whether the passed in object was modified or not.

The manager needs to keep a reference to the UMLModelManager in order to pass it to the formatters, as these might require different elements from the model, or might add elements, if not present, to the model

1.2.2 Standard Class Data Loader

The manager will return the standard namespaces for a specific language from the Standard Class Data Loader component.

1.2.3 Standard stereotypes

The manager will return the standard stereotypes for a given element type, according to the language. The source should be pluggable and a concrete XML based implementation (or using Configuration Manager) should be provided.

1.2.4 InitialElementFormatter

This interface will apply formatting to model elements that will be provided by the component. The manager will ask every registered formatter to apply the formatting, so the concrete implementations should determine from the model element argument whether to apply the formatting or not.

The component will also provide concrete implementations of the interface:

The formatters will receive in the constructor a reference to the UMLModelManager.

1.2.4.1 JavaExceptionElementFormatter

This formatter will apply formatting to model elements of Class type that have the “exception” stereotype. It will add two constructors:

+ExceptionName(message:String)

+ExceptionName(message:String,cause:Throwable)

The constructors will have the <<create>> stereotype.

The names of the stereotypes should be configurable.

If the UML Model does not contain the stereotypes, they will be added. Same for the String and Throwable classes.

1.2.4.2 CSharpExceptionElementFormatter

This formatter will apply formatting to model elements of Class type that have the "Exception" stereotype. It will add four constructors:

+ExceptionName()

+ExceptionName(message:string)

+ExceptionName(message:string,innerException:Exception)

#ExceptionName(info:SerializationInfo,context:StreamingContext)

The constructors will have the <<create>> stereotype.

The names of the stereotypes should be configurable.

If the UML Model does not contain the stereotypes, they will be added. Same for the string, Exception, SerializationInfo and StreamingContext classes.

1.2.4.3 Java14EnumElementFormatter

This formatter will apply formatting to model elements of Class type that have the "enumeration" stereotype. It will add a private constructor with no arguments and will add a Generalization relationship towards Enum class from type Safe Enum component.

The names of the stereotypes should be configurable.

If the UML Model does not contain the <<create>> stereotype or the Enum class, they will be added. (Note that Enum class is abstract)

1.2.5 InitialDiagramElementFormatter

This interface will apply formatting to diagram elements an will be provided by the component. The manager will ask every registered formatter to apply the formatting, so the concrete implementations should determine from the diagram element argument (i.e. using its UML semantic model element) whether to apply the formatting or not.

The component will also provide a concrete implementation of the interface:

1.2.5.1 PropertyTemplateDiagramElementFormatter

This formatter will add Property instances to the diagram element. The "fill" property is just a sub case, so there won't be a special handling. The formatter must function properly for the graph nodes, being smart enough to determine particular cases, like exceptions (classes with <<exception>> stereotype, the required pseudo states ...). In case there isn't a special restriction that would block the formatter to work properly for other types of diagram elements (like graph edges ...), these should also be supported.

1.2.6 Default Project Language

It will provide a method to retrieve the default project language, to be used if such information is missing when a new project is created.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

The component will be used in the TopCoder UML Tool to load the design project's configuration, according to the project's language.

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 interface found in the class diagram with the component interfaces. The designer is encouraged to add to the existing interface, 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.uml.projectconfiguration

com.topcoder.uml.projectconfiguration.modelelementformatters

com.topcoder.uml.projectconfiguration.diagramelementformatters

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The initial elements formatters
- The initial graph node formatters
- The standard stereotypes source
- The standard class data loader

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 components
- Diagram Interchange
- UML Model Manager
- Standard Class Data Loader
- Configuration Manager 2.1.5 - recommended

**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.