UML Project Configuration 1.0 Component Specification

1. Design

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 ...).

This design provides a UML project configuration manager which configures the project highly depending on the XML configuration file. The user can get the standard stereotypes, namespace and default project language by the configuration manager. The user can also register/un-register element formatters to providing initial formatting for the elements.

1.1 Design Patterns

- Facade Pattern is used by ProjectConfigurationManager to provide configuration functionalities for the project.
- Strategy Pattern is used by InitialElementFormatter and InitialDiagramElementFormatter to allowed customized formatter, and it is used by StandardStereotypeLoader to enable pluggable source of stereotypes.

1.2 Industry Standards

None

1.3 Required Algorithms

1.3.1 Format Java Exception Element

```
If the element is not a Class, simply return false.
Validate the element with the configuration name
as "exceptionStereotypeName"
If the return is false, simply return false.
Get the create stereotype with the configuration name
as "createStereotypeName"
Get the String and Throwable class with the configuration name
as "stringClassName" and "throwableClassName"
Create a message parameter named "message" and type is String.
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PUBLIC, stereotype is create
and parameter is message.
Create another message parameter, and a cause parameter named "cause"
with its type is Throwable.
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PUBLIC, stereotype is create
and parameter is the other message and cause.
Return true if the element is modified, otherwise return false.
```

1.3.2 Format C# Exception Element

```
If the element is not a Class, simply return false.
Validate the element with the configuration name
as "exceptionStereotypeName"
```

```
If the return is false, simply return false.
Get the create stereotype with the configuration name
as "createStereotypeName"
Get the String/Exception/SerializationInfo/StreamingContext class
with the configuration name
as "stringClassName", "throwableClassName", "serializationInfoClas
sName" and "streamingContextClassName"
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PUBLIC, stereotype is create.
Create a message parameter named "message" and type is String.
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PUBLIC, stereotype is create
and parameter is message.
Create another message parameter, and a innerException parameter
named "innerException" with its type is Exception.
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PUBLIC, stereotype is create
and parameter is the other message and innerException.
Create a parameter named "info" with its type is SerializationInfo,
another parameter named "context" with its type is StreamingContext.
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PROTECTED, stereotype is create
and parameter is the info and streamingContext.
Return true if the element is modified, otherwise return false.
```

1.3.3 *Format Java1.4 Enum Element*

```
If the element is not a Class, simply return false.
Validate the element with the configuration name as "enumStereotypeName"
Get the create stereotype with the configuration name
as "createStereotypeName"
Get the Enum class with the configuration name as "enumClassName"
Add an operation to the element by addOperation method, its name is
the same as the element, visibility is PRIVATE, stereotype is create.
Create a Generalization relationship, set the child to the element and
set the parent to the Enum class.
If the relationship is not contained by the element, add it to the element.
Return true if the element is modified, otherwise return false.
```

1.3.4 *Format Property Template Diagram Element*

```
Get an instance of the config manager
Get namespace for the the diagram element by
configManager.getPropertyObject(namespace, element.getClass())
If the elementProperty is null, throw ProjectConfigurationException
Get the names of the property by elementProperty.propertyNames()
For each name...
Get the value by configManager.getString(elementNamespace, name)
Create a property with the name as the key and the value as the value.
If the property is not contained by this element, add the property to
the element
... end each
Return true if the element is modified, otherwise return false.
```

1.3.5 Add Operation To Class

```
create an operationImpl.
Set its name to the name, and also the visibility.
Add the stereotypes and parameters to the operation.
Apply initial formatting for this operation by the
ProjectConfigurationManager got from umlModelManager
Add the operation to the class.
Return true.
```

1.4 Component Class Overview

ProjectConfigurationManager:

This class is main class of this component. 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.

InitialElementFormatter:

This interface specifies the contract for implementations of a model element formatter. This formatter will apply formatting to model elements.

AbstractElementFormatter:

This formatter will provide general function for its subclass, such as creating parameter, validate class, add operation to class, get operation and class.

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)

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)

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.

InitialDiagramElementFormatter:

This interface specifies the contract for implementations of a diagram element formatter. This formatter will apply formatting to diagram elements.

PropertyTemplateDiagramElementFormatter:

This formatter will add Property instances to the diagram element according to the XML file.

XMLStereotypeLoader:

This loader will load stereotypes from the XML file.

1.5 Component Exception Definitions

ProjectConfigurationException:

This exception will be thrown by the ProjectConfigurationManager, AbstractElementFormatter and InitialElementFormatter & InitialDiagramElementFormatter implementations when them encounters an exception trying to get configuration information for the configuration files. This exception will be exposed to the caller of the format method of the formatters. And it also will be exposed to the caller of the constructor, getStandardStereotypes, applyInitialFormatting and getDefaultProjectLanguage methods of ProjectConfigurationManager.

1.6 Thread Safety

This component is not thread-safe as the maps used in the ProjectConfigurationManager. This component considered to be used in single thread environment. If the ProjectConfigurationManager is used in the multi-thread, the user should ensure that don't change the map concurrently or change the map while apply formatting.

The other classes are thread-safe by being immutable.

2. Environment Requirements

2.1 Environment

- Development language: Java1.5
- Compile target: Java1.5

2.2 TopCoder Software Components

- UML Model components 1.0 defines the elements to be formatted and used in formatting
- Diagram Interchange 1.0 defines the elements to be formatted and used in formatting
- UML Model Manager 1.0 is used to get/add the necessary element from/to the model
- Standard Class Data Loader 1.0 is used to get the standard namespaces
- Configuration Manager 2.1.5 is used to read configuration from configuration files.
- UML_Model Core 1.0 provides the definition and implementation of ModelElement, Operation, Parameter and Feature.
- UML_Model Core Extension_Mechanisms 1.0 provides the definition and implementation of Stereotype.
- UML_Model Core Classifer 1.0 provides the definition and implementation of Class and Classifier.
- Base Exception 1.0 is used to provide a base for all custom exceptions.
- Object Factory 2.0.1 is used to create the configured object.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.uml.projectconfiguration

com.topcoder.uml.projectconfiguration.modelelementformatters

com.topcoder.uml.projectconfiguration.diagramelementformatters

com.topcoder.uml.projectconfiguration.stereotypeloader

3.2 Configuration Parameters

This component has the following configuration using the specified namespaces (or one provided by the application:

Namespace: com.topcoder.uml.projectconfiguration.ProjectConfigurationManager

Property Name	Description	Format	Required
stereotypes	Specifying the stereotypes according to the language and element type, please see 4.3.1.1.	com.topcoder.util.config. Property	Yes
defaultProjectLanguage	A non null, non empty string specifying the default language used for projects.	String	Yes

Namespace: com.topcoder.uml.projectconfiguration.ProjectConfigurationManager.objectfactory

The manage also will use the ObjectFactory to create the standard class data loader if it is not provide by the caller of the constructor:

Class Type	Object Factory Class Identifier
com.topcoder.uml.standardclassloader. StandardClassDataLoader	standardClassDataLoader

Namespace: com.topcoder.uml.projectconfiguration.modelelementformatters. AbstractElementFormatter

Property Name	Description	Format	Required
exceptionStereotypeName	A non null, non empty string specifying the name of the exception stereotype	String	Yes
enumStereotypeName	A non null, non empty string specifying the name of the enum stereotype	String	Yes
enumClassName	A non null, non empty string specifying the name of the enum class	String	Yes
createStereotypeName	A non null, non empty string specifying the name of the create stereotype	String	Yes
createStereotypeBaseClass	A non null, non empty string specifying the baseclass of the create stereotype	String	Yes
stringClassNam	A non null, non empty string specifying	String	Yes

e	the name of the String class		
throwableClassName	A non null, non empty string specifying the name of the Throwable class	String	Yes
exceptionClassName	A non null, non empty string specifying the name of the Exception class	String	Yes
serializationInfoClassName	A non null, non empty string specifying the name of the **SerializationInfo** class	String	Yes
streamingContextClassName	A non null, non empty string specifying the name of the **StreamingContext** class	String	Yes
operationStereotypeBaseClass	A non null, non empty string specifying the steretype's baseclass attribute of operation	String	Yes

Namespace: com.topcoder.uml.projectconfiguration.diagramelementformatters. PropertyTemplateDiagramElementFormatter

It contains the element properties specified for each element type (element,getClass). The every propertyof the element contains the properties of this type, please see 4.3.1.2.

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

Nothing special required

4.3 Demo

4.3.1 Configuration File Sample

4.3.1.1 Configuration File for ProjectConfigurationManager

```
<?xml version="1.0"?>
<CMConfig>
 <Property name="stereotypes">
   <Property name="Java">
     <Property name="Class">
         <Value>exception</Value>
         <Value>abstract</Value>
     </Property>
     <Property name="Operation">
         <Value>final</Value>
         <Value>abstract</Value>
     </Property>
   </Property>
   <Property name="CSharp">
     <Property name="Class">
         <Value>exception</Value>
         <Value>abstract</Value>
     </Property>
   </Property>
 </Property>
 <Property name="defaultProjectLanguage">
   <Value>Java</Value>
 </Property>
</CMConfig>
```

4.3.1.2 *Sample Configuration File for PropertyTemplateDiagramElementFormatter*

```
<?xml version="1.0"?>
<CMConfig>
  <Property name="com.topcoder.diagraminterchange.Reference">
     <Property name="property1">
       <Value>value1</Value>
     </Property>
     <Property name="property2">
        <Value>value2</Value>
     </Property>
  </Property>

  <Property name="com.topcoder.diagraminterchange.GraphNode">
    <Property name="property1">
      <Value>value1</Value>
    </Property>
  </Property>
</CMConfig>
```

4.3.2 *Simple Demo*

```
//create a project configuration manager
ProjectConfigurationManager configurationManager = new
ProjectConfigurationManager(umlModelManager,
classDataLoader, stereotypeLoader, NAMESPACE);

//set the conffigurationManager to umlModelManager

umlModelManager.setProjectConfigurationManager(configurationManager);

//Get the standard stereotypes for java class
List<Stereotype> javaClassStereotypes =
configurationManager.getStandardStereotypes("Java", "Class");

//Get the standard namespace for java
List<Namespace> javaNamespace =
configurationManager.getStandardNamespaces("Java");

//Add validator for the CreateDiagramAction
InitialElementFormatter formatter = new
JavaExceptionElementFormatter(NAMESPACE, umlModelManager, "Java");
configurationManager.addInitialElementFormatter("Java", formatter);
configurationManager.addInitialElementFormatter("CSharp", new
CSharpExceptionElementFormatter(NAMESPACE,
          umlModelManager, "CSharp"));
configurationManager.addInitialElementFormatter("Java14", new
Java14EnumElementFormatter(NAMESPACE,
          umlModelManager, "Java"));
configurationManager.addInitialDiagramFormatter("Java14",
          new PropertyTemplateDiagramElementFormatter(NAMESPACE));

//Apply initial formatting for model element
ModelElement exception = new ClassImpl();
configurationManager.applyInitialFormatting("Java", exception);

//Apply initial formatting for diagram element
DiagramElement diagram = new Reference();
configurationManager.applyInitialFormatting("Java", diagram);

//Get default project language
String defaultPoejectLanguage =
configurationManager.getDefaultProjectLanguage();
```

5. Future Enhancements

None