Properties Panel 1.1 Component Specification

1 Design

1.1 Overview

The Properties Panel component provides a SWING panel that allows the user to set different properties of model elements and groups of elements. It also provides a way to signal the listeners of changes.

The panel will display properties that are relevant to the selected model element. When several model elements are selected, the panel is configured such that only those properties that are relevant to *all* of the model elements will be shown. The application can register listener(s) to the panel to listen to these user-made changes. The application can also register listener(s) that listen to change in model element being selected (for example, when viewing properties of a class attribute, the user may click on the link to the owner of the attribute—the class—to view the properties of the owner. The application may need to be informed of such changes so that it could process the diagram display appropriately).

The component provides several configurable properties, such as the look-and-feel of the GUI and the default values for multiplicities.

Note that this component will NOT modify the ModelElement whose properties is being displayed.

1.2 Design Patterns

Façade – The **PropertiesPanel** class acts as a façade. It provides methods to simplify configuration of the panel, including registering listeners.

Listener – The **PropertiesPanel** class uses the listener design pattern to notify application of modification of properties within the panel.

MVC – The component uses GUI elements that use MVC.

1.3 Industry Standards

UML 1.5

1.4 Required Algorithms

1.4.1 Retrieving all the namespace/classifier in a Model

Step 1: Retrieve the UMLModelManager.

Step 2: Retrieve the model using UMLModelManager.getModel().

Step 3: Retrieve all the namespaces using model.getOwnedElements().

To retrieve all the classifiers, perform the following additional step:

Step 4: for each namespaces retrieved in Step 3, retrieve the classifiers associated with the namespace using namespace.getOwnedElements().

1.4.2 Determining whether a PropertyPanel should be visible

The specific instances of PropertyPanel will have one or several related PropertyKind associated with them. To determine whether an instance should be displayed, use the rule given below. Note that if there are more than one PropertyKind supported by the instance, if any of the supported PropertyKind satisfies the rule, the instance should be visible.

Rules:

If PropertyKind is (note that all the interfaces mentioned are from UML Model):

1. NAME – isVisible always returns true
2. NAMESPACE – isVisible returns true only if all the ModelElement is of the type Package, Interface, Class, Enumeration (Class Diagram), Actor, UseCase, Subsystem (Use Case Diagram)
3. OWNER – isVisible returns true only if all the ModelElement is of the type Operation
4. TYPE – isVisible returns true only if all the ModelElement is of the type Attribute, Parameter, AssociationEnd (Class Diagram), or Object (Sequence Diagram).
5. VISIBILITY – isVisible returns true only if all the ModelElement is of the type Interface, Class, Enumeration, Attribute, Operation, AssociationEnd (Class Diagram), Actor, UseCase, Subsystem (Use Case Diagram), or Object (Sequence Diagram).
6. CHANGEABILITY – isVisible returns true only if all the ModelElement is of the type Attribute or AssociationEnd.
7. INITIAL_VALUE – isVisible returns true only if all the ModelElement is of the type Attribute or AssociationEnd.
8. GUARD – isVisible returns true only if all the ModelElement is of the type Transition.
9. AGGREGATION – isVisible returns true only if all the ModelElement is of the type AssociationEnd.
10. MULTIPLICITY – isVisible returns true only if all the ModelElement are Attribute or AssociationEnd.
11. KIND – isVisible returns true only if all the ModelElement are Parameter.
12. CONCURRENCY – isVisible returns true only if all the ModelElement are Operation.
13. ORDERING – isVisible returns true only if all the ModelElement are AssociationEnd.
14. PARAMETERS – isVisible returns true only if only 1 ModelElement is selected and it is of the type Operation.
15. STEREOTYPES – isVisible returns true if and only if 1 ModelElement is configured.
16. OWNER_LINK – isVisible returns true if all the ModelElement is of the type Parameter.
17. SUPPLIER_LINK – isVisible returns true if all the ModelElement is of the type Dependency.
18. CLIENT_LINK – isVisible returns true if all the ModelElement is of the type Dependency.

19. PARENT_LINK – isVisible returns true if all the ModelElement is of the type Generalization or Abstraction.
20. CHILD_LINK – isVisible returns true if all the ModelElement is of the type Generalization or Abstraction.
21. ASSOCIATION_LINK – isVisible returns true if all the ModelElement is of the type AssociationEnd.
22. BASE_LINK – isVisible returns true if all the ModelElement is of the type Extend or Include.
23. EXTENSION_LINK – isVisible returns true if all the ModelElement is of the type Extend.
24. ADDITION_LINK – isVisible returns true if all the ModelElement is of the type Include.
25. SOURCE_LINK – isVisible returns true if all the ModelElement is of the type Transition.
26. TARGET_LINK – isVisible returns true if all the ModelElement is of the type Transition.
27. INCOMING_TRANSITIONS – isVisible returns true if all the ModelElement is of the type StateVertex (with exception of Pseudostate with its kind set to PseudostateKind.INITIAL—this is an Initial Node, there is no incoming transition).
28. OUTGOING_TRANSITIONS– isVisible returns true if all the ModelElement is of the type StateVertex but not of the type FinalState.
29. ASSOCIATION_ENDS_LINK – isVisible returns true if all the ModelElement is of the type Association.
30. ACTION_LINK – isVisible returns true if all ModelElement is of the type Stimulus.
31. STIMULUS_LINK – isVisible returns true if all ModelElement is of the type Action.
32. NAMESPACE_LINK – isVisible returns true if all ModelElement is of the type Dependency, Generalization, Abstraction, Association, Include, Extend, or Object.
33. Modifiers (ABSTRACT, FINAL, ROOT, STATIC, ACTIVE, TRANSIENT, NAVIGABLE, ASYNCHRONOUS) – isVisible returns true if all the ModelElement fits into at least one of the modifiers (refer to section 1.4.5).

1.4.3 Configuring PropertiesPanel

This is the algorithm needed in `configurePanel` method of PropertiesPanel class:

Step 1: Call removeAll methods of leftPanel and rightPanel to clear them.

Step 2: For each XXXPropertyPanel instance, call its configurePanel method. Then call the isVisible method. If it returns true, add the panel#retrievePanel() to the respective JPanel (see below).

Step 3: Call PropertiesPanel#validate method.

Note that in Step 2, configurePanel will detect for grouping of ModelElement as well.

For each of the XXXPropertyPanel, they should be added (when needed) to the panel

given below, in the order given below (***Note***: Developers are free to play around with this rule, but they need to make sure that this part of the CS is updated):

leftPanel:

- NamePropertyPanel
- NamespacePropertyPanel
- OwnerPropertyPanel
- TypePropertyPanel
- InitialValuePropertyPanel
- GuardPropertyPanel
- IncomingTransitionPropertyPanel
- OutgoingTransitionPropertyPanel
- MultiplicityPropertyPanel
- ConcurrencyPropertyPanel
- AssociationEndsPropertyPanel
- All the PropertyPanel concrete implementations in the package: *com.topcoder.gui.panels.properties.propertypanel.links* (the order is Supplier, Client, Parent, Child, Association, Base, Extension, Addition, Source, Target, Owner, Action, Stimulus, and Namespace.
- VisibilityPropertyPanel
- ChangeabilityPropertyPanel
- ModifiersPropertyPanel
- KindPropertyPanel
- OrderingPropertyPanel
- AggregationPropertyPanel

rightPanel:

- StereotypeListPropertyPanel
- ParameterListPropertyPanel (this is a special case – see the GUI image below)

Note that the 2 panels will reside on the left and right part of the PropertiesPanel. The layouts could be found in Section 1.5. If more than one elements are selected, all panels will be displayed in top-bottom manner.

1.4.4 *Configuring default Multiplicity values*

Pseudocode is given as follow:

```
List<Multiplicity> multiplicities = new ArrayList<Multiplicity>();
foreach MultiplicityName in Multiplicities properties
    Multiplicity m = new MultiplicityImpl();
    foreach RangeName in MultiplicitiesName properties
        MultiplicityRange range = new MultiplicityRangeImpl();
        if LowerBound property exists and parsable as integer
            range.setLower(LowerBound as integer)
        if UpperBound property exists and parsable as integer
            range.setUpper(UpperBound as integer)
        range.setMultiplicity(m);
        m.addRange(range);
    multiplicities.add(m)
```

1.4.5 *Configuring ModifiersPropertyPanel*

ModifiresPropertyPanel is the most complex PropertyPanel implementation in the whole component. The modifier JCheckBox will only be displayed if all the ModelElement supported the modifier.

Here are the list of the modifiers and the ModelElement where the modifiers are supported (in bracket is the corresponding getter/setter):

abstract – Package, Interface, Class, Enumeration, Operation, Actor, UseCase, Subsystem (is/setAbstract)

final – Package, Interface, Operation, Actor, UseCase, Subsystem (is/setLeaf), Parameter[4],

root – Package, Interface/Class/Enumeration in a non-Classifier Namespace, Actor, UseCase, Subsystem (is/setRoot)

static – Interface/Class/Enumeration in a Classifier Namespace, Operation (is/setRoot), Attribute[2], AssociationEnd[2],

active – Class, Enumeration (is/setActive)

transient – Attribute[3], AssociationEnd[3]

navigable – AssociationEnd (is/setNavigable)

asynchronous – Stimulus, whose Action (Stimulus.getDispatchAction().getAction()) is a CallOperationAction instance (is/setAsynchronous).

Note:

[1] To check whether an Attribute is final, use getChangeability method, if it returns ChangeableKind.FROZEN, it is final, if it is CHANGEABLE, it is not final. Use the corresponding setter to set.

[2] To check whether an Attribute/AssociationEnd is static, use getOwnerScope/getTargetKind method respectively. If it returns ScopeKind.CLASSIFIER, it is static, if it returns INSTANCE, it is non-static. Use the corresponding setter to set.

[3] To check whether an Attribute/AssociationEnd is transient, check the TaggedValue with tagType “transient”. If the value is “true” then it is transient, if it is “false”, it is not

transient. To set, retrieve the TaggedValue with tagType “transient” and update its value accordingly (if such TaggedValue is not found, create a new TaggedValue).

[4] To check whether a Parameter is final, check for the TaggedValue with tagType “final”. If the value is “true” then it is final, if it is “false”, it is not final. To set, retrieve the TaggedValue with tagType “final” and update its value accordingly (if such TaggedValue is not found, create a new TaggedValue).

1.4.6 TaggedValue-related Algorithms

To retrieve a TaggedValue with tagType “type”:

```
for (TaggedValue taggedValue : ModelElement.getTaggedValue()) {
    TagDefinition tagDefinition = taggedValue.getType();
    if (tagDefinition.getTagType().equals(type) {
        // taggedValue is the required TaggedValue.
    }
)
```

To retrieve a TaggedValue’s value: `taggedValue.getDataValue();`

To create a new TaggedValue with tagType “type” and value “value”:

```
TagDefinition tagDefinition = new TagDefinitionImpl();
tagDefinition.setTagType(“type”);
TaggedValue taggedValue = new TaggedValueImpl();
taggedValue.setType(tagDefinition);
taggedValue.setDataValue(“value”);
// Now we can add the taggedValue to ModelElement using #addTaggedValue.
```

1.4.7 Understanding stereotypes list

Stereotype list is probably the most complex part of this component. A ModelElement may have stereotypes assign to it. They are stored in its stereotypes attributes and may be retrieved through getStereoTypes method.

Additionally, we may want to assign additional stereotypes to a ModelElement. Each different interfaces implementing ModelElement will have a different set of allowed stereotypes. They are all stored in ProjectConfigurationManager. However, to retrieve this default stereotypes, we have to use a mapping string provided in the configuration files (see section 3.2). There is different mapping string for different interfaces implementing ModelElement (the configuration name is named to the interface name). This mapping is pretty complex.

To figure out this mapping, we need to use the Class diagrams provided from TC UML Collaboration forum. They are included in the docs as reference (in the folder docs/uml_model). To discover the mapping string for an instance of ModelElement, we first obtain all the interfaces that it implements (call ModelElement.getClass().getInterfaces()). Subsequently we have to determine which of these interfaces that is the lowest in the hierarchy of TC UML (meaning that the interface implements all the other interfaces in the List—this condition is, by the way, guaranteed).

Once we have obtained the required interface, we need to retrieve the mapping string (it is stored in stereotypesMapping). Then we can call:

```
ProjectConfigurationManager manager =
```

```
        umlModelManager.getProjectConfigurationManager();
Collections<Stereotype> stereotypes = manager.getStandardStereotypes(
        umlModelManager.getProjectLanguage(), mappingString);
```

We also need to obtain stereotypes list that belong to the current model. We get the model using getModel method UMLModelManager. Then we retrieve the owned element using getOwnedElements method as follows:

```
Collection<ModelElement> ownedElements = model.getOwnedElements();
for (ModelElement e : ownedElements) {
    if (e instanceof Stereotype) {
        // add this to the list of stereotypes.
    }
}
```

1.4.7 The ModelElementChangeListener

This listener provides a way for application to detect for changes in properties within the panel. The listener will be notified through PropertiesPanel#fireModelElementPropertyChange method. The listener will have the information on which ModelElement is modified, what property is modified (in the form of PropertyKind enumeration), what is the operation (add, remove, or replace previous properties), and the Object to add/remove/replace the original property.

Note:

For stereotype, the Object passed will be a Set of selected stereotypes. However, there may be additional stereotype that is not yet registered in the model/UMLModelManager. It is up to the application logic to handle this stereotype (the application may choose to add this new stereotype to model.

For guard, the value being passed is the text value that the user filled in the guard textbox.

Examples:

When name of ModelElement `element` is changed, the listener will be called as follow:

```
stateChanged(element, PropertyKind.NAME, PropertyOperation.REPLACE,
        "new name");
```

When a new Parameter `param` is added on ModelElement `element`, the following call is made:

```
stateChanged(element, PropertyKind.PARAMETER, PropertyOperation.ADD,
param);
```

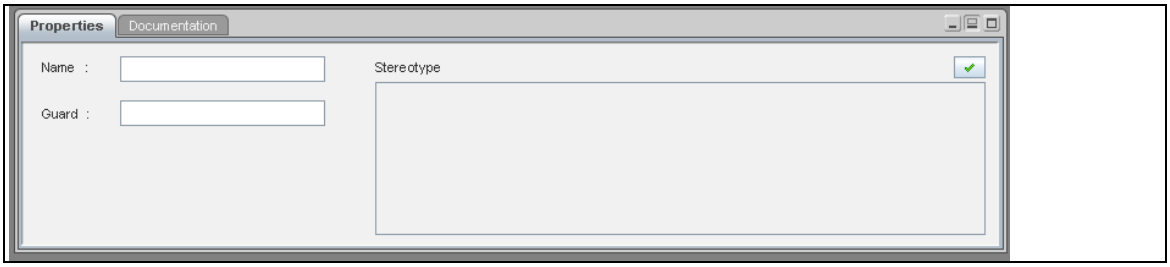
When a Parameter `param` is removed from ModelElement `element`, the following call is made:

```
stateChanged(element, PropertyKind.PARAMETER, PropertyOperation.REMOVE,
param);
```

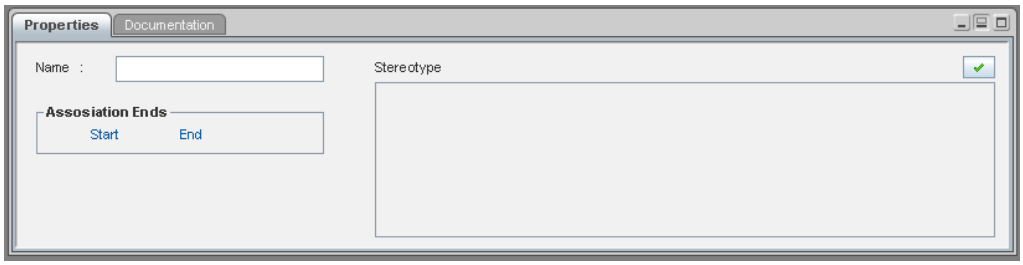
1.5 GUI Overview

The component GUI is illustrated in the following examples:

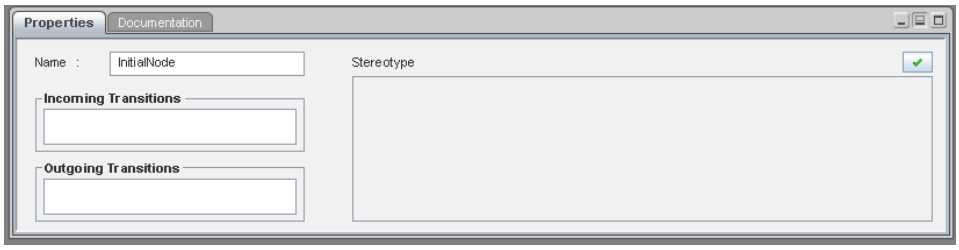
Transaction:



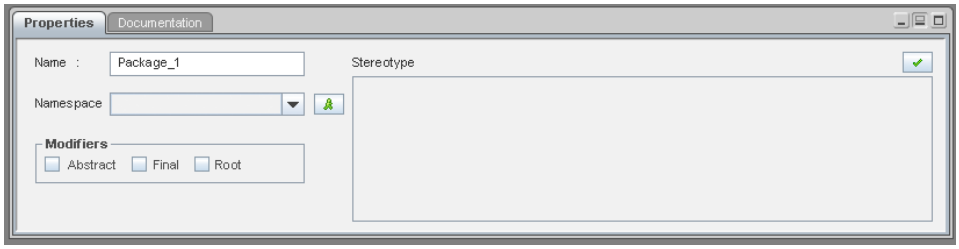
Aggregation:



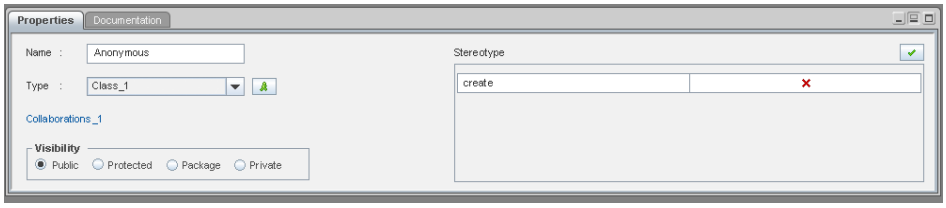
Action State:



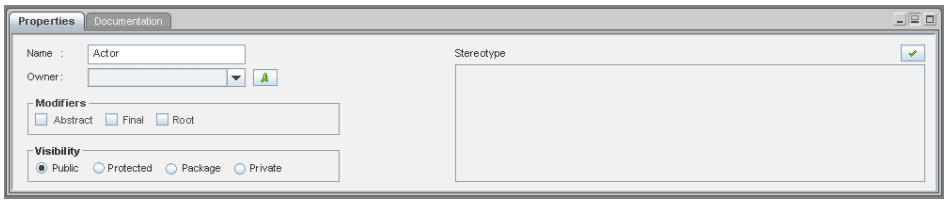
Package:



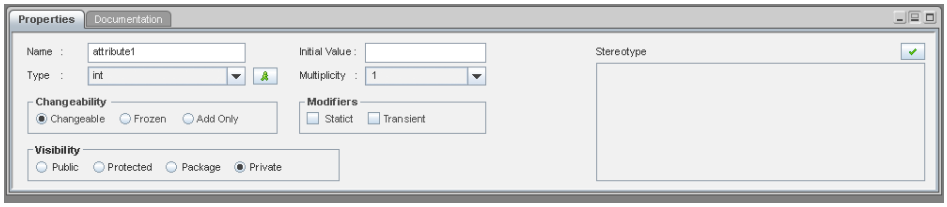
Lifeline:



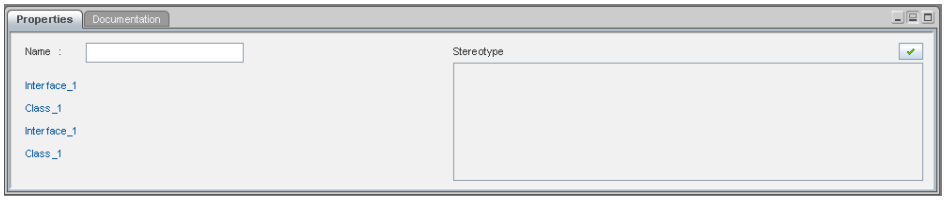
Use Case:



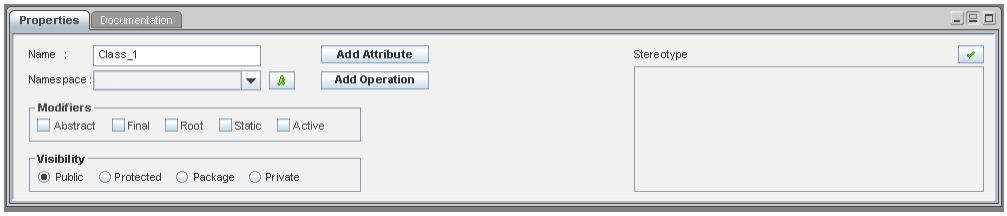
Attribute:



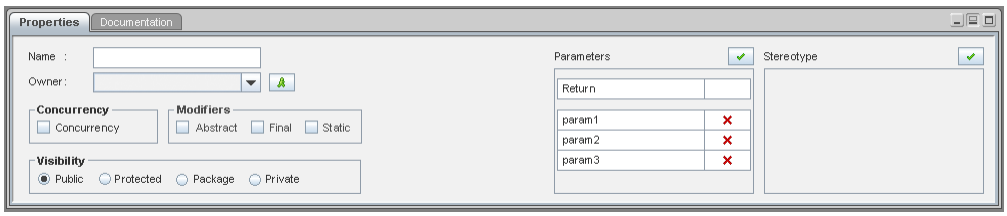
Realization:



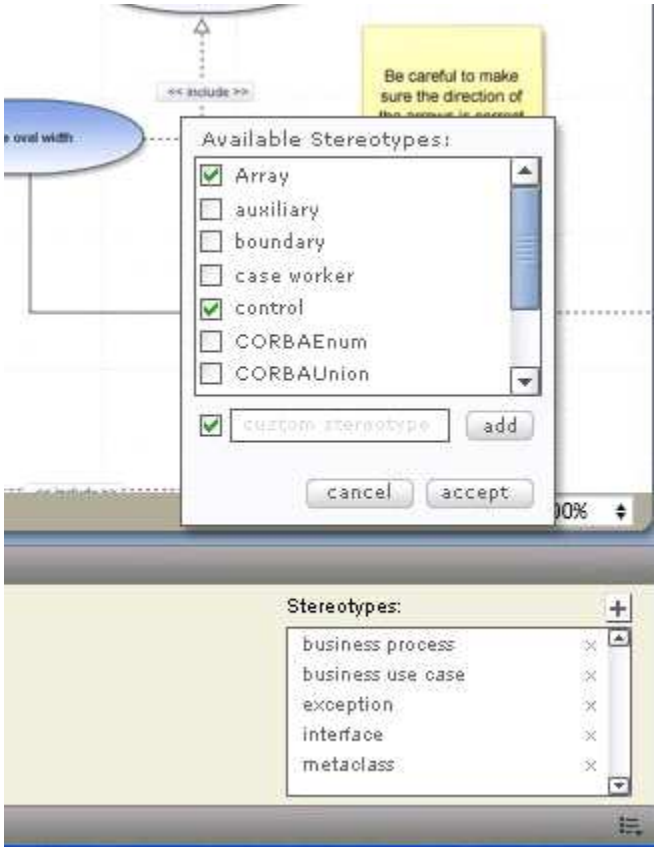
Class:



Operation:



The stereotype popup is shown below:



To ease developers, below are the necessary properties to be shown when a ModelElement is selected:

1.5.1 Class diagram elements

- Package
  - Name
  - Namespace - a combo box with all the namespaces from the UML Model. It should also have a button to select and edit the properties for the namespace.
  - Modifiers: abstract, final, root
  - Stereotypes list
- Interface
  - Name
  - Namespace - a combo box with all the namespaces from the UML Model. It should also have a button to select and edit the properties for the namespace.
  - Visibility: public, protected, package, private
  - Modifiers: abstract, final, root (static - if the namespace is a classifier)
  - Stereotypes list

- Class - same as the Interface, with an extra ‘active’ modifier
- Exception - same as the Class
- Enumeration - same as the Class
- Attribute
    - Name
    - Type - a combo box with all the Classifiers from the UML Model. It should also have a button to select and edit the properties for the classifier
    - Initial Value
    - Multiplicity - should be an editable combo box, with default values to be chosen. Custom values should be allowed. The values should be positive, the lower range should be lower than the upper range, if both are present, and multiple ranges are allowed, if comma separated.
    - Visibility: public, protected, package, private
    - Ordering: unordered, ordered, unspecified
    - Modifiers: static, transient
    - Changeability: changeable, frozen, add only
    - Stereotypes list
- Operation
    - Name
    - Owner - a combo box with all the Classifiers from the UML Model. It should also have a button to select and edit the properties for the classifier.
    - Concurrency - a “synchronized” check box
    - Modifiers: abstract, static, final
    - Visibility: public, protected, package, private
    - Parameters list
    - Stereotypes list
- Argument
    - Name
    - Type - a combo box with all the Classifiers from the UML Model. It should also have a button to select and edit the properties for the classifier.
    - Owner - a link to the Operation
    - Kind: in/out, in, out, return
    - Modifiers: final
    - Stereotypes list
- Dependency
    - Name
    - Namespace - a link to the namespace (the namespace is automatically selected as the base namespace of the relationship ends).

    - Supplier - a link to the supplier end
    - Client - a link to the client end
    - Stereotypes list
- Realization
    - Name
    - Namespace - a link to the namespace (the namespace is automatically selected as the base namespace of the relationship ends).
    - Parent - a link to the parent end
    - Child - a link to the child end
    - Stereotypes list
- Abstraction - same as for Realization
- Association
    - Name
    - Namespace - a link to the namespace (the namespace is automatically selected as the base namespace of the relationship ends).
    - Association ends - link towards the two association ends
    - Stereotypes list
- AssociationEnd
    - Name
    - Type - a combo box with all the Classifiers from the UML Model. It should also have a button to select and edit the properties for the classifier
    - Initial Value
    - Multiplicity - should be an editable combo box, with default values to be chosen. Custom values should be allowed. The values should be positive, the lower range should be lower than the upper range, if both are present, and multiple ranges are allowed, if comma separated.
    - Association - a link towards the association
    - Aggregation: none, aggregation, composition
    - Visibility: public, protected, package, private
    - Ordering: unordered, ordered, unspecified
    - Changeability: changeable, frozen, add only
    - Modifiers: static, transient, navigable
    - Stereotypes list

1.5.2 Use case diagram elements

- Actor
    - Name

    - Namespace - a combo box with all the namespaces from the UML Model. It should also have a button to select and edit the properties for the namespace.
    - Visibility: public, protected, package, private
    - Modifiers: abstract, final, root
    - Stereotypes list
- Use Case - same as for the Actor
- Subsystem - same as for the Actor
- Extend
    - Name
    - Namespace - a link to the namespace (the namespace is automatically selected as the base namespace of the relationship ends).
    - Base - a link to the base use case
    - Extension - a link to the extension use case
    - Stereotypes list
- Include
    - Name
    - Namespace - a link to the namespace (the namespace is automatically selected as the base namespace of the relationship ends).
    - Base - a link to the base use case
    - Addition - a link to the addition use case
    - Stereotypes list

1.5.3 Activity diagram elements

- Initial Node
    - Name
    - Outgoing transitions
    - Stereotypes list
- Object Flow State
    - Name
    - Incoming transitions
    - Outgoing transitions
    - Stereotypes list
- Action State - same as for Object Flow State
- Send Signal Action - same as for Object Flow State
- Accept Event Action - same as for Object Flow State
- Fork Node - same as for Object Flow State

- Join Node - same as for Object Flow State
- Decision Node - same as for Object Flow State
- Merge Node - same as for Object Flow State
- Flow Final Node
    - Name
    - Incoming transitions
    - Stereotypes list
- Final Node - same as for the Flow Final Node
- Transition
    - Name
    - Guard - a text input
    - Source - a link to the source state
    - Target - a link to the target state
    - Stereotypes list

1.5.4 Sequence diagram elements

- Object
    - Name
    - Namespace - a link to the Collaboration it belongs to.
    - Type - a combo box with all the Classifiers from the UML Model. It should also have a button to select and edit the properties for the classifier
    - Visibility: public, protected, package, private
    - Stereotypes list
- Create Message Action
    - Name
    - Stimulus - a link to the stimulus, which should also have a link to the action
    - Stereotypes list
- Call Message Action
    - Name
    - Stimulus - a link to the stimulus, which should also have a link to the call operation action, which should have the “asynchronous” modifier
    - Stereotypes list
- Send Stimulus Message Action
    - Name
    - Stimulus - a link to the stimulus, which should also have a link to the send action
    - Stereotypes list

- Return Message Action
    - Name
    - Stimulus - a link to the stimulus
    - Stereotypes list
- Stimulus
    - A link to the Action associated with the Stimulus, if such Action exists

1.6 Component Class Overview

Package *com.topcoder.gui.panels.properties*

Class **PropertiesPanel** extends JPanel:
This class provides the main GUI panel for the component. As the name suggests, this GUI panel will display all the available properties of configured ModelElement(s). When the panel is configured with multiple ModelElement, only the common properties are displayed. Furthermore, the panel can be reconfigured with different ModelElement(s) on the go without recreating the whole GUI components again.

Interface **PropertyPanel**:
This interface provides a contract for a GUI panel that is responsible for one or more properties. The implementations will usually be as small a unit as possible. As such, the implementations will usually be responsible only for one or a few closely related property that can be displayed in a single JPanel. The implementations provide a GUI. They are also responsible for detecting changes to the supported properties and update the configured ModelElement(s) appropriately when a property is changed. The implementations are expected to be a sub-panel in the PropertiesPanel class.

Interface **ModelElementChangeListener** extends EventListener (from *java.util.EventListener*):
This interface provides a contract for a listener that can be notified when a property of a ModelElement is modified. An application can implement its application logic to response to a change in ModelElement's property.

Interface **ModelElementSelectionListener** extends EventListener (from *java.util.EventListener)*:
This interface provides a contract for a listener that can be notified when the PropertiesPanel switch its focus to properties of another ModelElement due to its internal workings. An application can then implement its application logic to response to this change.

Enum **PropertyKind**:
This provides an enumeration of all the property types supported by this component.

Enum **PropertyOperation:**

This enumeration contains the possible operation on a properties.

Package *com.topcoder.gui.panels.properties.propertypanel*

Class **AbstractPropertyPanel** implements PropertyPanel:
This class provides an abstract base for all the PropertyPanel interface implementations. The class provides the basic functionalities common to all PropertyPanel implementations.

Class **NamePropertyPanel** extends AbstractPropertyPanel:

This class provides the GUI for the property panel for name property of ModelElement.

Class **NamespacePropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for namespace property of ModelElement.

Class **NamespaceItem**:
This class provides a wrapper over Namespace object so that it can be displayed properly in a JComboBox while still preserving the ease of access to the corresponding Namespace instance.

Class **OwnerPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for owner property of ModelElement.

Class **TypePropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for type property of ModelElement.

Class **VisibilityPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for visibility property of ModelElement.

Class **InitialValuePropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for initial value property of ModelElement.

Class **GuardPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for guard property of ModelElement.

Class **ChangeabilityPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for changeability property of ModelElement.

Class **ModifiersPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for modifiers of ModelElement.

Class **ModifiersItemListener** (an inner class of ModifiersPropertyPanel):
This class provides an item listener for each modifier checkbox in Modifiers property panel. It provides the update logic as well.

Class **IncomingTransitionsPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for incoming transitions of ModelElement.

Class **OutgoingTransitionsPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for outgoing transitions of ModelElement.

Class **TransitionListItem**:
This class provides a list item wrapper for Transition ModelElement. This enables a Transition to be displayed as plain text in a JList while still being able to retrieve the corresponding Transition object with relative ease.

Class **MultiplicityPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for multiplicity of ModelElement.

Class **MultiplicityItem** (an inner class in MultiplicityPropertyPanel):
This class provides a wrapper for Multiplicity object. This enables a Multiplicity to be displayed as plain text in a JComboBox while still being able to retrieve the

corresponding Multiplicity object with relative ease.

Class **KindPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for kind property of ModelElement.

Class **ConcurrencyPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for concurrency property of ModelElement.

Class **OrderingPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for ordering property of ModelElement.

Class **AssociationEndsPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for association ends property of ModelElement.

Class **AggregationPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel for aggregation property of ModelElement.

Class **ParameterListPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel that displays a list of parameters for a ModelElement. The display also supports adding new parameter and deleting parameter.

Inner Class **ParameterItem** (in ParameterListPropertyPanel):
This class provides a wrapper for Parameter instance so that it can be displayed easily by the JTable. At the same time, the Parameter instance can be easily recovered back.

Class **StereotypeListPropertyPanel** extends AbstractPropertyPanel:
This class provides the GUI for the property panel that displays a list of stereotypes for a ModelElement. The display supports adding new Stereotype and managing Stereotype (through inner class AddStereotypeJDialog) and deleting Stereotype.

Inner Class **AddStereotypeJDialog** extends JDialog:
This class provides a modal JDialog GUI for adding and removing stereotypes. It also enables adding custom stereotypes.

Class **StereotypeItem**:
This class provides a wrapper for Stereotype instance so that it can be displayed easily by the JTable. At the same time, the Stereotype instance can be easily recovered back.

Class **CustomTableModel** (package-private):
This class provides a custom TableModel for displaying the custom tables for parameters and stereotypes list. It is heavily used in ParameterListPropertyPanel, StereotypeListPropertyPanel and AddStereotypeJDialog.

Package *com.topcoder.gui.panels.properties.propertypanel.links*

Class **AbstractLinkPropertyPanel** implements PropertyPanel
This class provides an abstract base for all the PropertyPanel interface implementations that provides a single link to another ModelElement in the model. The class provides the basic functionalities common to all PropertyPanel implementations and a common GUI for all the subclasses. The common GUI consists of a JLabel with the property name and another JLabel with the property

value that acts as a link to another JLabel.

Class **SupplierLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for supplier link property of ModelElement. It provides a link to the supplier.

Class **ClientLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for client link property of ModelElement. It provides a link to the client.

Class **ParentLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for parent link property of ModelElement. It provides a link to the parent.

Class **ChildLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for child link property of ModelElement. It provides a link to the child.

Class **AssociationLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for association link property of ModelElement. It provides a link to the association.

Class **BaseLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for base link property of ModelElement. It provides a link to the base.

Class **ExtensionLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for extension link property of ModelElement. It provides a link to the extension.

Class **AdditionLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for addition link property of ModelElement. It provides a link to the addition.

Class **SourceLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for source link property of ModelElement. It provides a link to the source.

Class **TargetLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for target link property of ModelElement. It provides a link to the target.

Class **OwnerLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for owner link property of ModelElement. It provides a link to the owner.

Class **ActionLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for action link property of ModelElement. It provides a link to the action.

Class **StimulusLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for stimulus link property of ModelElement. It provides a link to the stimulus.

Class **NamespaceLinkPropertyPanel** extends AbstractLinkPropertyPanel:
This class provides the GUI for the property panel for namespace link property of ModelElement. It provides a link to the namespace.

1.7 Component Exception Definitions

IllegalArgumentException (from *java.lang*):

This Java exception is used to indicate invalid arguments when needed.

PropertiesPanelConfigurationException extends BaseException (custom):
This exception is used to indicate errors during retrieval of configuration properties through Configuration Manager.

1.8 Thread Safety

The component is not thread-safe as most of the classes are mutable. However, this should not be a problem as the GUI is usually run in a single-threaded AWT thread, which eliminates many thread-safety issues. In normal usage, it is unlikely that thread-safety is going to be an issue. Because of this, the additional cost in execution time for making the component thread-safe can not be justified.

2 Environment Requirements

2.1 Environment

- Java 1.5 is required

2.2 TopCoder Software Components

- Base Exception 1.0: provides the base class for the custom exception.
- Configuration Manager 2.1.5: used by the PropertiesPanel class to retrieve configuration properties.
- UML Model 1.0: used by the PropertyPanel interface implementations to retrieve property values and update them. This component contains all the required data structures. This component includes the entire components marked as UML Model – XXX 1.0 in TopCoder Software Catalog.
- UML Model Manager 1.0: used to retrieve the current model and all the namespaces and classifiers contained within the model. Also used to obtain model-specific stereotypes list.
- UML Project Configuration 1.0: used to retrieve the default stereotypes list.

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

3 Installation and Configuration

3.1 Package Name

com.topcoder.gui.panels.properties

com.topcoder.gui.panels.properties.propertypanel

com.topcoder.gui.panels.properties.propertypanel.links

3.2 Configuration Parameters

This component depends on a configuration file for its configuration. The configuration file is readable using Configuration Manager Component.

Parameter	Description	Typical Value
LookAndFeelClass	Fully-qualified class name of the LookAndFeel class to be used. **Optional**	Fully-qualified class name
LinkButtonImagePath	The path of the link button image (for buttons that act as a link to another model element). **Required**	Path to Image
AddButtonImagePath	The path of the add button image (for buttons that act as an add button). Used in Parameter List and Stereotype List. **Required**	Path to Image
DeleteButtonImagePath	The path of the delete button image (for buttons that act as a delete button). Used in Parameter List and Stereotype List. **Required**	Path to Image
Multiplicities	Contains all the default multiplicities. **Required**	A property container
Multiplicities.<MultiplicityName>	Contains the configuration for the multiplicity. At least 1 such container is required within Multiplicities. **Required**	A property container
Multiplicities.<MultiplicityName> .<RangeName>	Contains the configuration for the given multiplicity range. At least 1 such container must exist for every MultiplicityName. **Required**	A property container
Multiplicities.<MultiplicityName> .<RangeName>.LowerBound	The lower bound of the multiplicity range. If it is not given, * is assumed. **Optional**	Any integer > 0 within int bound or a character *
Multiplicities.<MultiplicityName> .<RangeName>.UpperBound	The upper bound of the multiplicity range. If it is not given, * is assumed. **Optional**	Any integer > 0 within int bound or a character *
StereotypeMapping	Contains all the stereotype mappings. **Required**	A property container.
StereotypeMapping.Package	The stereotype mapping for ModelElement Package interface. **Required**	A String.
StereotypeMapping.Interface	The stereotype mapping for ModelElement	A String.

	Interface interface. **Required**	
StereotypeMapping.Class	The stereotype mapping for ModelElement Class interface. **Required**	A String.
StereotypeMapping.Enumeration	The stereotype mapping for ModelElement Enumeration interface. **Required**	A String.
StereotypeMapping.Attribute	The stereotype mapping for ModelElement Attribute interface. **Required**	A String.
StereotypeMapping.Operation	The stereotype mapping for ModelElement Operation interface. **Required**	A String.
StereotypeMapping.Parameter	The stereotype mapping for ModelElement Parameter interface. **Required**	A String.
StereotypeMapping.Dependency	The stereotype mapping for ModelElement Dependency interface. **Required**	A String.
StereotypeMapping.Generalization	The stereotype mapping for ModelElement Generalization interface. **Required**	A String.
StereotypeMapping.Abstraction	The stereotype mapping for ModelElement Abstraction interface. **Required**	A String.
StereotypeMapping.Association	The stereotype mapping for ModelElement Association interface. **Required**	A String.
StereotypeMapping.AssociationEnd	The stereotype mapping for ModelElement AssociationEnd interface. **Required**	A String.
StereotypeMapping.Actor	The stereotype mapping for ModelElement Actor interface. **Required**	A String.
StereotypeMapping.UseCase	The stereotype mapping for ModelElement UseCase interface. **Required**	A String.
StereotypeMapping.Subsystem	The stereotype mapping for ModelElement Subsystem interface. **Required**	A String.
StereotypeMapping.Include	The stereotype mapping for ModelElement Include interface. **Required**	A String.
StereotypeMapping.Extend	The stereotype mapping for ModelElement Extend interface. **Required**	A String.
StereotypeMapping.SimpleState	The stereotype mapping for ModelElement SimpleState interface. **Required**	A String.
StereotypeMapping.ObjectFlowState	The stereotype mapping for ModelElement ObjectFlowState interface. **Required**	A String.
StereotypeMapping.FinalState	The stereotype mapping for ModelElement FinalState interface. **Required**	A String.
StereotypeMapping.ActionState	The stereotype mapping for ModelElement ActionState interface. **Required**	A String.
StereotypeMapping.Pseudostate	The stereotype mapping for ModelElement Pseudostate interface. **Required**	A String.
StereotypeMapping.Transition	The stereotype mapping for ModelElement Transition interface. **Required**	A String.

StereotypeMapping.Object	The stereotype mapping for ModelElement Object interface. **Required**	A String.
StereotypeMapping .CreateObjectAction	The stereotype mapping for ModelElement CreateObjectAction interface. **Required**	A String.
StereotypeMapping .CallOperationAction	The stereotype mapping for ModelElement CallOperationAction interface. **Required**	A String.
StereotypeMapping .SendSignalAction	The stereotype mapping for ModelElement SendSignalAction interface. **Required**	A String.
StereotypeMapping.Stimulus	The stereotype mapping for ModelElement Stimulus interface. **Required**	A String.

3.3 Dependencies Configuration

The configuration provided in section 3.2 must be provided through Configuration Manager.

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 the component by pointing the Configuration Manager at the configuration file described above
- See Demo for examples of usage.

4.3 Demo

4.3.1 Using Properties Panel

```
UMLModelManager umlModelManager = TestHelper.createUMLModelManager();
ModelElement element = new GuardImpl();
List<ModelElement> elements = new ArrayList<ModelElement>();
elements.add(element);

// Creates a new JPanel to contain the PropertiesPanel instance.
JPanel pane = new JPanel();

// Creates a new PropertiesPanel.
PropertiesPanel propertiesPanel = new PropertiesPanel(umlModelManager);

// Add the PropertiesPanel to JPanel.
pane.add(propertiesPanel);

// The following code will show the available properties
```

```
        // for a ModelElement element
        propertiesPanel.configurePanel(element);

        // The following code will show only the common properties
        // for the ModelElement instances. If elements only contains
        // 1 ModelElement, this call is exactly the same as the code above.
        propertiesPanel.configurePanel(elements);
```

4.3.2 Demo on Property Change Listener for Properties Panel

```
        // Creates a new PropertiesPanel.
        PropertiesPanel propertiesPanel = new
PropertiesPanel(TestHelper.createUMLModelManager());

        // Implements a listener that is notified when a property of a
        // ModelElement is changed through Properties Panel.
        ModelElementChangeListener propertyListener = new ModelElementChangeListener() {
            public void stateChanged(ModelElement modelElement, PropertyKind property,
                PropertyOperation op, Object o) {
                System.out.println("Receive State Changed Event : ModelElement is of ["
                    + modelElement.getClass().getSimpleName() + "] type, Property kind is
                    [" + property
                    + "], property operation is [" + op + "], the object is of ["
                    + (o == null ? null : o.getClass().getSimpleName()) + "] type, value
is ["
                    + (o == null ? null : o.toString()) + "]");
            }
        };

        // Adds the listener to the PropertiesPanel.
        propertiesPanel.addModelElementChangeListener(propertyListener);

        // Removes the listener from the PropertiesPanel.
        propertiesPanel.removeModelElementChangeListener(propertyListener);

        // Removes all the listener from the Properties Panel.
        propertiesPanel.removeAllModelElementChangeListeners();
```

4.3.2 Demo on Selection Change Listener for Properties Panel

```
        // Creates a new PropertiesPanel.
        PropertiesPanel propertiesPanel = new
PropertiesPanel(TestHelper.createUMLModelManager());

        // Implements a listener that is notified when the PropertiesPanel
        // switch focus to another ModelElement due to internal event
        // (such as a user clicking a link button for a Namespace to
        // view the property of that Namespace).
        ModelElementSelectionListener selectionListener = new
ModelElementSelectionListener() {
            public void selectionChanged(ModelElement modelElement) {
                System.out.println("Receive Selection Changed Event : ModelElement is of
["
                    + modelElement.getClass().getSimpleName() + "] type");
            }
        };

        // Adds the listener to the PropertiesPanel.
        propertiesPanel.addModelElementSelectionListener(selectionListener);

        // Removes the listener from the PropertiesPanel.
        propertiesPanel.removeModelElementSelectionListener(selectionListener);

        // Removes all the listener from the Properties Panel.
        propertiesPanel.removeAllModelElementSelectionListeners();
```

5 Future Enhancements

- More checks for properties can be added (currently not all properties have checks).
- The GUI may also be improved further.
- Support for future UML specification can be easily provided with minimal modifications.