XMI Reader UML Model Plugin Component 1.0 Specification

1. Design

The XMI Reader UML Model Plugin component is a plugin for XMI Reader component. It implements the ContentHandler interface and is able to parse all the elements for the classes in the UML Model, except the state machine and the activity graph.

1.1 General approach

1.1.1 Anatomy of the proposed design

The basic tasks of this design can be broken up as follows:

- We will need to have the ability to parse out the XMI data that defines any Model Element.
  We will mostly deal with parser events such as startElement and endElement, which will hold the bulk of our processing.
- Actually we will even go further, and create something of a factory, which will generate the appropriate object based on the type of Element that is being read in XMI. Thus we will have the ability to plug-in new classes of model items (such as Expression, Class, or Method) into our factory. This factory is fully configurable and works on the assumption that each such class acts as a java bean and has a default constructor as well as well defined setters.

Please note that even though we are not modeling a StateMachine in this component I have decided to use a state machine model as an example of how to deal with xmi. This is a generic approach and this it doesn't really matter which ModelElement we use since all the elements in this component are derived in the very same manner as StateMachine is. So it imperative to note that this is used only as an example of xmi and its parsing rather than as a specific directive for this component. All elements (as listed in the UC) follow the same general idea. The designer simply felt that showing a simple yet relatively sophisticated element is very important. And I felt that state machine will actually do this trick rather well.

1.1.2 Basic Parsing approach

The basic aspect of this component is of course the parsing of the xmi. The approach that this design takes is quite straightforward:

- As we read the xmi (xml) the parser will fire off `startElement` and `endElement` events, which our handler will process.
- Each `startEvent` will tell us of a (possibly) new element of the graph which we need to translate into an object instance.
- For each <UML:XXX element we try to match it to a specific class and we instantiate through reflection.
  For example:
  ```
  <UML:Diagram xmi.id = 'I10ad419m10729d8da08mm7f54' isVisible = 'true'
  name = 'Class Diagram_1' zoom = '1.0'>
  ```
  Would result in creation of a new `DiagramImpl` instance which would have its `setVisible()`, `setName()` and `setZoom()` invoked (more on method invocation later)
- Once an element has been instantiated it will be placed in the `XMIReader's` foundElements map for future reference with the key being xmi.id This way when we later need to find this instance we simply look for an element with xmi.idref equal to the xmi.id just placed (since xmi uses references when an element is shared across the file)

- Note that some properties of an element will actually be other elements. The xmi specification uses notation such as <UML:outer:inner> where major is the main encompassing class, and minor is the included instance. This means that we will need to create an instance of whatever inner stands for, and then use the proper setter or add method (for collections or lists) to put the inner instance into the outer instance.

  As an example consider this snippet:

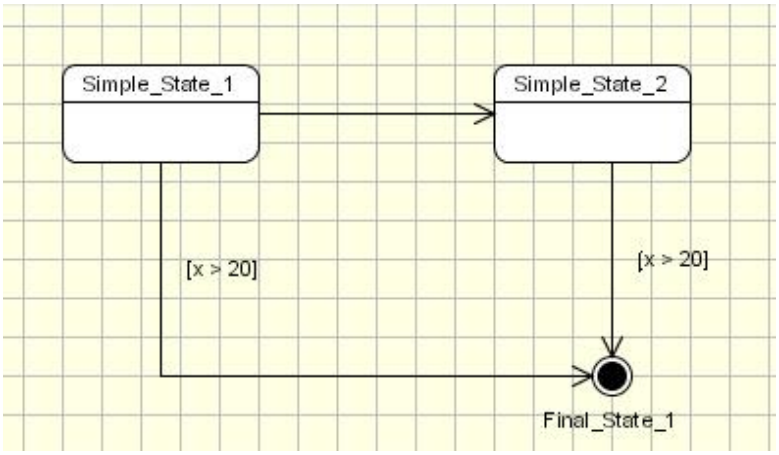
  ```
  <UML:StateVertex.outgoing>
      <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f2a'/>
  </UML:StateVertex.outgoing>
  ```

  Here the outer class is `StateVertex` and the inner class is `Transition`. This can only be established through consultation with the DTD. Some relevant examples to this component are provided later on.
- This process is basically repeated for each subsequent note/tag, which will eventually build all the pieces of the Model.

Please note that it is not viable to list all the possible aspect of the DTD (provided at \docs\ AS.dtd) What we will do here is show a generic approach which the developer can then expand on since it will involve the exact same approach as the provided example here. Even though state machine is used, it is a valid generic example of how to parse out and deal with the model elements.

Lets look at a simple example of a State Machine (StateMachine_1) and the XMI file for it:

Diagram 1. An example of a state machine graph



Here is the xml (xmi) for it in UML 1.5:

Listing 1. Example of state machine xmi (for diagram 1)

```
<UML:StateMachine xmi.id = 'I10ad419m10729d8da08mm7f50' name = 'StateMachine_1'
  isSpecification = 'false'>
  <UML:StateMachine.context>
    <UML:Class xmi.idref = 'I10ad419m10729d8da08mm7f51'/>
  </UML:StateMachine.context>
  <UML:StateMachine.top>
    <UML:CompositeState xmi.id = 'I10ad419m10729d8da08mm7f4f' name = ''
    isSpecification = 'false'
      isConcurrent = 'false'>
      <UML:CompositeState.subvertex>
        <UML:SimpleState xmi.id = 'I10ad419m10729d8da08mm7f45' name = 'Simple_State_1'
          visibility = 'public' isSpecification = 'false'>
```

```
          <UML:StateVertex.outgoing>
            <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f21'/>
            <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f18'/>
          </UML:StateVertex.outgoing>
        </UML:SimpleState>
        <UML:SimpleState xmi.id = 'I10ad419m10729d8da08mm7f3a' name = 'Simple_State_2'
          visibility = 'public' isSpecification = 'false'>
          <UML:StateVertex.outgoing>
            <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f2a'/>
          </UML:StateVertex.outgoing>
          <UML:StateVertex.incoming>
            <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f18'/>
          </UML:StateVertex.incoming>
        </UML:SimpleState>
        <UML:FinalState xmi.id = 'I10ad419m10729d8da08mm7f2f' name = 'Final_State_1'
          visibility = 'public' isSpecification = 'false'>
          <UML:StateVertex.incoming>
            <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f2a'/>
            <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f21'/>
          </UML:StateVertex.incoming>
        </UML:FinalState>
      </UML:CompositeState.subvertex>
    </UML:CompositeState>
  </UML:StateMachine.top>
  <UML:StateMachine.transitions>
    <UML:Transition xmi.id = 'I10ad419m10729d8da08mm7f2a' isSpecification = 'false'>
      <UML:Transition.guard>
        <UML:Guard xmi.id = 'I10ad419m10729d8da08mm7f05' name = '' visibility = 'public'
          isSpecification = 'false'>
          <UML:Guard.expression>
            <UML:BooleanExpression xmi.id = 'I10ad419m10729d8da08mm7f04' language = 'java'
              body = 'x &gt; 20'/>
          </UML:Guard.expression>
        </UML:Guard>
      </UML:Transition.guard>
      <UML:Transition.source>
        <UML:SimpleState xmi.idref = 'I10ad419m10729d8da08mm7f3a'/>
      </UML:Transition.source>
      <UML:Transition.target>
        <UML:FinalState xmi.idref = 'I10ad419m10729d8da08mm7f2f'/>
      </UML:Transition.target>
    </UML:Transition>
    <UML:Transition xmi.id = 'I10ad419m10729d8da08mm7f21' isSpecification = 'false'>
      <UML:Transition.guard>
        <UML:Guard xmi.id = 'I10ad419m10729d8da08mm7f0f' name = '' visibility = 'public'
          isSpecification = 'false'>
          <UML:Guard.expression>
            <UML:BooleanExpression xmi.id = 'I10ad419m10729d8da08mm7f0e' language = 'java'
              body = 'x &gt; 20'/>
          </UML:Guard.expression>
        </UML:Guard>
      </UML:Transition.guard>
      <UML:Transition.source>
        <UML:SimpleState xmi.idref = 'I10ad419m10729d8da08mm7f45'/>
      </UML:Transition.source>
      <UML:Transition.target>
        <UML:FinalState xmi.idref = 'I10ad419m10729d8da08mm7f2f'/>
      </UML:Transition.target>
    </UML:Transition>
    <UML:Transition xmi.id = 'I10ad419m10729d8da08mm7f18' isSpecification = 'false'>
      <UML:Transition.source>
        <UML:SimpleState xmi.idref = 'I10ad419m10729d8da08mm7f45'/>
      </UML:Transition.source>
      <UML:Transition.target>
        <UML:SimpleState xmi.idref = 'I10ad419m10729d8da08mm7f3a'/>
      </UML:Transition.target>
    </UML:Transition>
  </UML:StateMachine.transitions>
</UML:StateMachine>
```

The one thing to note is that this is just a simple xml file with a very specific (DTD is provided in \docs\AS.dtd) format. The way that we will have this handled is by simply walking down the DOM of the xmi and by pushing parsing specific elements.
For example we could envision that parsing the above information would produce the following order of events, where each event is depicted as an entry into an xml element:

Example 1. Elements that will be translated into actual objects

```
<UML:StateMachine>                                    // we build a StateMachine instance
  <UML:StateMachine.context>                          // We build and add context
  <UML:StateMachine.top>                              // We build and add a top state
    <UML:CompositeState>                              // which is a composite state
      <UML:CompositeState.subvertex>                  // which is made of 3 simple states
        <UML:SimpleState name = 'Simple_State_1'>     // This is the first state
          <UML:StateVertex.outgoing>                  // which has 2 outgoing transitions
            <UML:Transition>                          // outgoing transition 1a
            <UML:Transition>                          // outgoing transition 2a
        <UML:SimpleState name = 'Simple_State_2'>     // This is the second state
          <UML:StateVertex.outgoing>                  // which has 1 outgoing transition
            <UML:Transition>                          // outgoing transition 1b
          <UML:StateVertex.incoming>                  // which has 1 incoming transition
            <UML:Transition>                          // incoming transition 1b
        <UML:FinalState name = 'Final_State_1'>       // This is the final state
          <UML:StateVertex.incoming>                  // which has 2 incoming transitions
            <UML:Transition>                          // incoming transition 1c
            <UML:Transition>                          // incoming transition 2c
  <UML:StateMachine.transitions>                      // Actual transition details
    <UML:Transition>                                  // We build a transition element
      <UML:Transition.guard>                          // which includes a guard
        <UML:Guard>                                   // we build the guard
          <UML:Guard.expression>                      // which includes and expression
            <UML:BooleanExpression body = 'x&gt;20'/> // we build a boolean expression
      <UML:Transition.source>                         // Source of the transition
        <UML:SimpleState>                             // it starts in Simple_State_1
      <UML:Transition.target>                         // target of the transition
        <UML:FinalState>                              // it ends in Final_State_1
    <UML:Transition>                                  // Another transition element
      <UML:Transition.guard>                          // guard
        <UML:Guard>                                   // we build it
          <UML:Guard.expression>                      // the guard expression
            <UML:BooleanExpression body = 'x&gt;20'/> // we build it
      <UML:Transition.source>                         // it starts in Simple_State_1
      <UML:Transition.target>                         // target of the transition
        <UML:FinalState>                              // it ends in Final_State_1
    <UML:Transition>                                  // Another transition element
      <UML:Transition.source>                         // Source of the transition
        <UML:SimpleState>                             // it starts in Simple_State_1
      <UML:Transition.target>                         // target of the transition
        <UML:SimpleState>                             // it ends in Final_State_2
```

One very important aspect to note is that we are dealing with a pointer based data structure in the sense that some elements are defined using ids, which then get resolved into proper objects. Thus we need to be aware that in XMI we will have references and objects (classes)

Consider how transactions are actually defined. We have an actual physical definition of a transition as in:

Example 2. Transition definition in xmi

```
<UML:Transition xmi.id = 'I10ad419m10729d8da08mm7f2a' isSpecification = 'false'>
  <UML:Transition.guard>
    <UML:Guard xmi.id = 'I10ad419m10729d8da08mm7f05' name = '' visibility = 'public'
      isSpecification = 'false'>
      <UML:Guard.expression>
```

```
          <UML:BooleanExpression xmi.id = 'I10ad419m10729d8da08mm7f04' language = 'java'
            body = 'x &gt; 20'/>
        </UML:Guard.expression>
      </UML:Guard>
    </UML:Transition.guard>
    <UML:Transition.source>
      <UML:SimpleState xmi.idref = 'I10ad419m10729d8da08mm7f3a'/>
    </UML:Transition.source>
    <UML:Transition.target>
      <UML:FinalState xmi.idref = 'I10ad419m10729d8da08mm7f2f'/>
    </UML:Transition.target>
  </UML:Transition>
```

And then we have a reference to this as in:

Example 3. Transition reference example

```
<UML:StateVertex.incoming>
  <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f2a'/>
  <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f21'/>
</UML:StateVertex.incoming>
```

This means that our algorithm will have to basically build a mapping of ids to entities and then fill in the complete model with instances of classes that are substituted for the ids.

Note that the approach for an activity diagram will be exactly the same.

1.1.3 Mapping from xml tags elements to actual UML model

The first thing is to know how to create proper model objects based on parsed elements. This actually rather uncomplicated and works on the premise that a number of tags will map directly to implementation classes. For example the `<UML:StateMachine …/>` tag maps directly into our StateMachineImpl class, and thus when we encounter this tag we will be most probably creating (unless it is a reference) a new StateMachineImpl.
Here is a table that maps the main classes used here to their xmi tags:

Table 1. State Machine mapping from xmi to Object Model as an example of mapping

Xmi element	Object Model element	Description
UML:StateMachine	StateMachineImpl	Class
UML:Transition	TransitionImpl	Class
UML:Guard	GuardImpl	Class
UML:PseudoState	PseudoStateImpl	Class
UML:CompositeState	CompositeStateImpl	Class
UML:SimpleState	SimpleStateImpl	Class
UML:FinalState	FinalStateImpl	Class

But we have a special case here as well:

1. If the element has a attribute named `xmi.id`, we first check if a reference object was created for this (due to a forward `xmi.idref` declaration) and create a new create a new object if it wasn't.
2. If this attribute doesn't exist we then look for an `xmi.idref` attribute.
    a. If we have the attribute then we use its value to lookup to see if this object was already created.
    b. If it has been created we simply fetch it. And if it has not been created, we create it and place the empty object instance into a table of forward references.

1.1.4 Using reflection to map setters to their xmi counterparts

The object instances that we create are empty, and they need to be filled up with the data from xmi.
The most challenging aspect of this task is to parse out enough information to match the data with the proper setter in the class. The idea is that we will use the fact that the DTD definition for the xmi UML is exactly matched in the setters and getters of all the models in the supporting components. This means that when we encounter something like < `UML:Transition.stateMachine …/>` we can very easily 'guess' the correct setter of the TransitionImpl class (which maps to `UML:Transition`) will be setStateMachine.
This means that most of the time we will easily be able to construct the correct setter call. But there are special cases that deal with Collections, Lists and boolean variables. We can follow this simple algorithm as follows (assume some elementName):

1. We check if we have a setElementName method in the instance.
2. [If failed] we next check if this was a Collection and we look for an addElementName method

Another case is if the element name is in the format of isElementName.

3. We check if we have a setElementName (we drop the 'is')

These are currently the only three possibilities (based on the DTD) for setters.

1.1.5 How do we know when we have a setter?

1. Anytime we have an attribute we have a setter (except that we do not treat the xmi.id and xmi.idref) as setters.
2. Any time we have a tag in the format of `<UML.inner.outer … />` the outer is treated as a setter. So for example if we had `<UML:Transition.guard . . .>` guard would be the outer element and we would have a setter in the form of `setGuard(…)`

1.1.6 Special cases for enum types

For all the enum parts the easiest way is to simply create a helper class (internal), which based on the name of the attribute would create the proper enum instance.

We only have the following possibilities (from the DTD):

```
<!-- ========= UML:Data_Types ========= -->
<!ENTITY % UML:AggregationKind '(none|aggregate|composite)'>
<!ENTITY % UML:CallConcurrencyKind '(sequential|guarded|concurrent)'>
<!ENTITY % UML:ChangeableKind '(changeable|frozen|addOnly)'>
<!ENTITY % UML:OrderingKind '(unordered|ordered)'>
<!ENTITY % UML:ParameterDirectionKind '(in|inout|out|return)'>
<!ENTITY % UML:ScopeKind '(instance|classifier)'>
<!ENTITY % UML:VisibilityKind '(public|protected|private|package)'>
<!ENTITY % UML:PseudostateKind '(choice|deepHistory|fork|initial|join|
   junction|shallowHistory)'>
```

and they are mapped as follows:

`UML:AggregationKind` -- is mapped to -- `UML:AssociationEnd.aggregation` (`aggregation` attribute basically)
`UML:CallConcurrencyKind` -- is mapped to -- `UML:Operation.concurrency` (`concurrency` attribute basically)
`UML:ChangeableKind` -- is mapped to -- `UML:StructuralFeature.changeability` (`changeability` attribute basically)

UML:OrderingKind -- is mapped to -- UML:StructuralFeature.ordering (ordering attribute basically)
UML:ParameterDirectionKind -- is mapped to -- UML:Parameter.kind (kind attribute basically)
UML:ScopeKind -- is mapped to -- UML:StructuralFeature.targetScope (targetScope attribute basically)
UML:VisibilityKind -- is mapped -- UML:ModelElement.visibility (visibility attribute basically)
UML:PseudostateKind -- is mapped to -- UML:Pseudostate.kind (kind attribute basically)

Note that I have listed all the possible enums from the DTD but for our handler will not care about UML:PseudostateKind since this is handled by another handler.

So using the mentioned private helper we create the proper Enum instance.

1.1.7 Once we have an object in our hand we can start setting its contents.Element ids and reference ids

We also need to make a distinction when we are using xmi.id vs. xmi.idref. The difference is crucial to parsing. The xmi.id refers to the actual entity (i.e. we will be creating an instance for it) whereas xmi.idref is a reference to such an entity. In effect it is a pointer to the definition of an entity.
Please note that the xmi.id xmi.idref is what the XMIReader component uses when its XMIReader.foundElements and XMIReader.forwardReferences are being used.
xmi.id is used as a key in foundElements and xmi.idref is usually used in forwardReferences.

1.2 Design Patterns

Factory pattern is used to create instances of different model classes based on some key.

1.3 Industry Standards

UML 1.5
XMI 1.2

1.4 Required Algorithms

The main algorithm is of course the actual parsing of the document for the specific elements.

1.4.1 Parsing the document's elements

The implemented SAX handler is really doing things through a call back. The main methods that we care about are startElement and endElement, which tell us that we have entered or exited an element.
The general aspect will be as follows:

startElement:
For each element that we use reflection as stated in section 1.1.2 and 1.1.3:
Lets say that we obtained an element with name "UML:StateMachine"
Based on the element name this would have happened:

```
// We create a new StateMachine instance (stateMachine)
Object temp =  getModelElementFactory.createModel("UML:StateMachine");
// We set all the provided data in properties according to reflection as
```

```
// described in 1.1.3
// We also get the xmi.id and we use this id as follows:
getXMIReader().putElement(xmi.id, stateMachine);
// we also set the last object to be StateMachine
setLastObject(StateMachine);
```

Note that similar ideas will apply to all the UML elements. The developer should have no issues expanding the above simple exposition.

endElement:

```
if(name  == "UML:ModelElement"){
       // use the xmi.id to get the actual object
       ModelElement element =
                      (ModelElement)(getXMIReader().getElement(xmi.id));
       // Add this to the uml model manager
       getUMLModelManager().setModel(element);
}
```

1.4.2 Managing Properties

During parsing, "forward references" will likely be encountered. `XMIHandler` implementations can still capture properties about elements that have not been fully defined. Specifically, the `XMIReader` can keep track of which defined elements are "waiting on" forward references. In this XMI fragment, the referenced Stereotype might not have been defined yet, but the `XMIHandler` for Stereotypes can capture the forward reference:

```
<UML:Operation xmi.id = 'I3d057933m10e2fd324d0mm7242'
             name = 'XMIReaderConfigException'
             visibility = 'public'
             isSpecification = 'false'
             ownerScope = 'instance'
             isQuery = 'false'
             concurrency = 'sequential'
             isRoot = 'false'
             isLeaf = 'false'
             isAbstract = 'false'>

        <UML:ModelElement.stereotype>
             <UML:Stereotype xmi.idref = 'I3fc39a51m10e2acac631mm7e57'/>
        </UML:ModelElement.stereotype>
...
```

To capture this, the XMIHandler for Stereotypes would execute:

```
reader.putElementProperty("I3fc39a51m10e2acac631mm7e57",
                    " UML:ModelElement.stereotype", operationObject);
```

This connects the `operationObject` with the as-yet-undefined `UML:Stereotype` object whose id is "`I3fc39a51m10e2acac631mm7e57`", indicating that the `UML:Stereotype` property is the property that should be used to connect the two.

Later, when the Stereotype is actually found, its XMI fragment might look like this:

```
<UML:Stereotype xmi.id = 'I3fc39a51m10e2acac631mm7e57' name = 'create'
       isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
       isAbstract = 'false'>
     <UML:Stereotype.baseClass>BehavioralFeature</UML:Stereotype.baseClass>
</UML:Stereotype>
```

The Stereotype handler could then instantiate a `StereotypeObject` with all its properties, and then tell all the "waiting" objects about the newly created `StereotypeObject`.

```
Map<String, List> waiting;
waiting = reader.getElementProperties("I3fc39a51m10e2acac631mm7e57");
```

Then for each property, and each object on the list of that property, they can be told what the Stereotype object is. Finally, the `StereotypeHandler` would actually define the Stereotype object in the reader:

```
reader.putElement("I3fc39a51m10e2acac631mm7e57", stereotypeObject);
```

which will remove the corresponding entry in the `forwardReferences` map.

Lets take a different example:

```
<UML:FinalState xmi.id = 'I10ad419m10729d8da08mm7f2f'
                  name = 'Final_State_1'
                  visibility = 'public'
                  isSpecification = 'false'>
      <UML:StateVertex.incoming>
          <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f2a'/>
         <UML:Transition xmi.idref = 'I10ad419m10729d8da08mm7f21'/>
      </UML:StateVertex.incoming>
</UML:FinalState>
```

step 1 -- XMIReader will call the handler for UML:FinalState which is handler1
(a) ---- handler 1 will create the object (object1) for this element and set the attribute properties to the object.

step 2 -- XMIReader will call the handler for UML:StateVertex.incoming which is also done by handler 1
(a) ---- handler 1will do nothing at this point

step 3 -- XMIReader will call the handler for UML:Transition which lets assume is done by handler2 (for illustrative purposes)
(a) ---- handler2 does the following:
reader.putElementProperty("I10ad419m10729d8da08mm7f2a",
" UML:StateVertex.incoming ", operationObject);
where operation object is the object1 created in step 1(a)

step 4 --XMIReader will call the handler for UML:Transition which lets assume is done by handler2 (for illustrative purposes)
(a) ---- handler2 does the following:
reader.putElementProperty("I10ad419m10729d8da08mm7f21",
" UML:StateVertex.incoming ", operationObject);
where operation object is the object1 created in step 1(a)

So the first question is how does handler2 get the reference to object1 and know about "UML:StateVertex.incoming"?

How do we that?

```
handlers = reader.getActiveHandlers().
lastIndex = handlers.size()-1;
String property = handlers.get(lastIndex-1).getLastProperty();
Object refObject = handlers.get(lastIndex-2).getLastRefObject();
```

Note that for property `lastIndex-1` was used, and for `refObject` `lastIndex-2` was used. The handler on `lastIndex` position should be the currently active handler.

1.4.3 Dealing with bad formatting of the xml

Since the DTD for the XMI is provided it makes sense for the application to validate the document before it is parsed (using a validating parser) but nevertheless the component should act defensively when it comes to creating object instances based on the supplied xmi. There are a couple aspects of this:

- When setting a property it is possible that the value will violate the actual object's specified range of allowable objects.
- When setting a property based on the input xmi it is possible that the actual property specified doesn't map to anything on the object. In other words, there is no setter that can be used based on the property specified in the xmi.

Whenever any such error occurs we throw the `ElementCreationException` since we are unable to actually create the object. Again this should not happen if the input xmi is correct and has been validated.

1.5 Component Class Overview

1.5.1 com.topcoder.uml.xmi.reader.handlers.umlmodel

XMI2ModelHandler <<concrete class>>
This is a concrete (but indirect) implementation of the `ContentHandler` interface. This is done by extending the `DefaultXMIHandler` abstract class, which implements some convenience methods that deal with XMIReader. This is important since we need to utilize `XMIReader` instance in the actual implementation.
This class is responsible for parsing out all the uml model elements except for State Machine and Activity Graph model elements. This handler uses the ModelElementFactory to instantiate the proper model class instance. Basically as we parse the xml elements we use the element designation name (like `UML:TagDefinition` for example) as a key to create an instance of a class in the Object Model which represents the `UML:TagDefinition` which in our case (depending on configuration of course) would be `TagDefinition`.

Thread Safety : This class is mutable and so is not thread safe. As this class is only intended to be used inside the XMIReader component, the thread safety should be handled by XMIReader component.

1.5.2 com.topcoder.xmi.reader.handlers.modelfactory

ModelElementFactory <<concrete class>>
This is a configurable factory, which creates UML Model class instances based on a mapping from xml element name to a class name of the model element that currently represents it.
This implementation is thread-safe. Even though it is not anticipated that many thread will be hitting the handler, it was decided that it might be worthwhile to have a single copy of the factory (to save memory and configuration read time)

Synchronization should be done in a block to minimize performance impact.

1.6 Component Exception Definitions

1.6.1 Custom Exceptions

ElementCreationException:
Thrown when the ModelElementFactory cannot create the requested instance. This could be due to reflection issues, sandbox security issues, or something else.

ConfigurationException:
It will be thrown by the two ModelElementFactory constructors if there are issues with configuration (i.e. wrong format or missing variables) or if Configuration Manager thrown ConfigManagerException .

1.6.2 System and general exceptions

In general this component will check for empty string input or null reference parameter input. It will also check of arrays have any null references or empty strings.

IllegalArgumentException:
This exception is used for invalid arguments. Invalid arguments in this design are used on some occasions with null input elements. The exact details are documented for each method in its documentation.

SAXException:
This exception is used to signal to the invoking SAX parser that the handler is unable to continue parsing the given data.

1.7 Thread Safety

Implementation is not thread-safe, because the SAX parser doesn't require a handler to be thread safe and the thread safety issue should be handled in the XMI Reader component, this component is only a plugin for the XMI Reader component.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- **All UML Model - XXX 1.0 (except for State Machines 1.0 and Activity Graphs)**
  Used to model the UML elements.
- **Config Manager 2.1.5**
  Used to implement to read configuration detail for mapping xml element names (q names) with class names, which will be used (via reflection) to create model elements for the xml element.
- **XMI Reader 1.0**

This is the base component for which we are writing this plugin. We not only will plug-in the handler but we also use the XMIReader itself in this component to manager ids and reference ids for xml elements.

- **UML Model Manager 1.0**
  This is used to store the parsed out and created Activity Graph diagram
- **Base Exception 1.0**
  This is used as a common exception-handling base, which allows for chaining exceptions.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.xmi.reader.handlers.uml.model

3.2 Configuration Parameters

3.2.1 ModelElementFactory configuration for Config manager

Parameter	Description	Values
xml_to_element_mapping	These are 1 or more xml element name mapping to a class name used to create an object, which can store information about the xml element in the UML Model. These are comma delimited values, where the first element is the xml element name and the second element is a class name". Note that sometimes the xml element name might be in the format of <UML:outer.inner> where outer is the main encompassing class, and inner is the included instance ***Required***	"UML:DataType, com.topcoder.uml.model.core.classifiers.DataType" "UML:StereoType.definedTag, com.topcoder.uml.model.extensionmechanisms.TagDefinitionImpl"

The actual table of all these associations is shown below. Please note that `<UML:outer.inner>` which is shown with a yellow highlight in there is shown as a concatenation of `<UML:outer` followed by all the instances of `.inner>` that are affiliated with `<UML:outer`. This is done to save space.

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

To support the current UML Model the user should configure the following mappings

Xmi element	Corresponding class
UML:CallOperationAction	com.topcoder.uml.model.actions.messagingactions.CallOperationActionImpl
.operation	
.isAsynchronous	
UML:SendSignalAction	com.topcoder.uml.model.actions.messagingactions.SendSignalActionImpl
.signal	
UML:CreateObjectAction	com.topcoder.uml.model.actions.objectactions.CreateObjectActionImpl
.classifier	
UML:DestroyObjectAction	com.topcoder.uml.model.actions.objectactions.DestroyObjectActionImpl
UML:Message	com.topcoder.uml.model.collaborations.collaborationroles.MessageImpl
.interaction	
.activator	
.sender	
.receiver	
.predecessor	
.communicationConnection	
. procedure	
.conformingStimulus	
UML:CollaborationInstanceSet	com.topcoder.uml.model.collaborations.collaborationinstances.CollaborationInstanceSetImpl
.interactionInstanceSet	
.collaboration	
.participatingInstance	
.participatingLink	
.constrainingElement	
UML:Collaboration	com.topcoder.uml.model.collaborations.collaborationinteractions.CollaborationImpl
.interaction	
.representedClassifier	
.representedOperation	
.constrainingElement	
.usedCollaboration	
UML:Procedure	com.topcoder.uml.model.commonbehavior.procedure.ProcedureImpl
UML:Object	com.topcoder.uml.model.commonbehavior.instances.ObjectImpl
UML:Stimulus	com.topcoder.uml.model.commonbehavior.instances.StimulusImpl
.argument	
.sender	
.receiver	
.communicationLink	
.dispatchAction	
UML:Link	com.topcoder.uml.model.commonbehavior.links.LinkImpl
.association	
.connection	
UML:LinkEnd	com.topcoder.uml.model.commonbehavior.links.LinkEndImpl
.instance	
.link	
.associationEnd	
.qualifiedValue	
UML:Parameter	com.topcoder.uml.model.core.ParameterImpl
.defaultValue	
UML:Method	com.topcoder.uml.model.core.MethodImpl
.body	
.specification	
UML:Operation	com.topcoder.uml.model.core.OperationImpl
.specification	
UML:Attribute	com.topcoder.uml.model.core.AttributeImpl
.initialValue	
.associationEnd	
UML:Class	com.topcoder.uml.model.core.classifiers.ClassImpl
UML:Interface	com.topcoder.uml.model.core.classifiers.InterfaceImpl
UML:DataType	com.topcoder.uml.model.core.classifiers.DataTypeImpl
UML:EnumerationLiteral	com.topcoder.uml.model.core.classifiers.EnumerationLiteralImpl
.enumeration	
UML:Primitive	com.topcoder.uml.model.core.classifiers.PrimitiveImpl
UML:ProgrammingLanguageDataType	com.topcoder.uml.model.core.classifiers.ProgrammingLanguageDataTypeImpl
.expression	

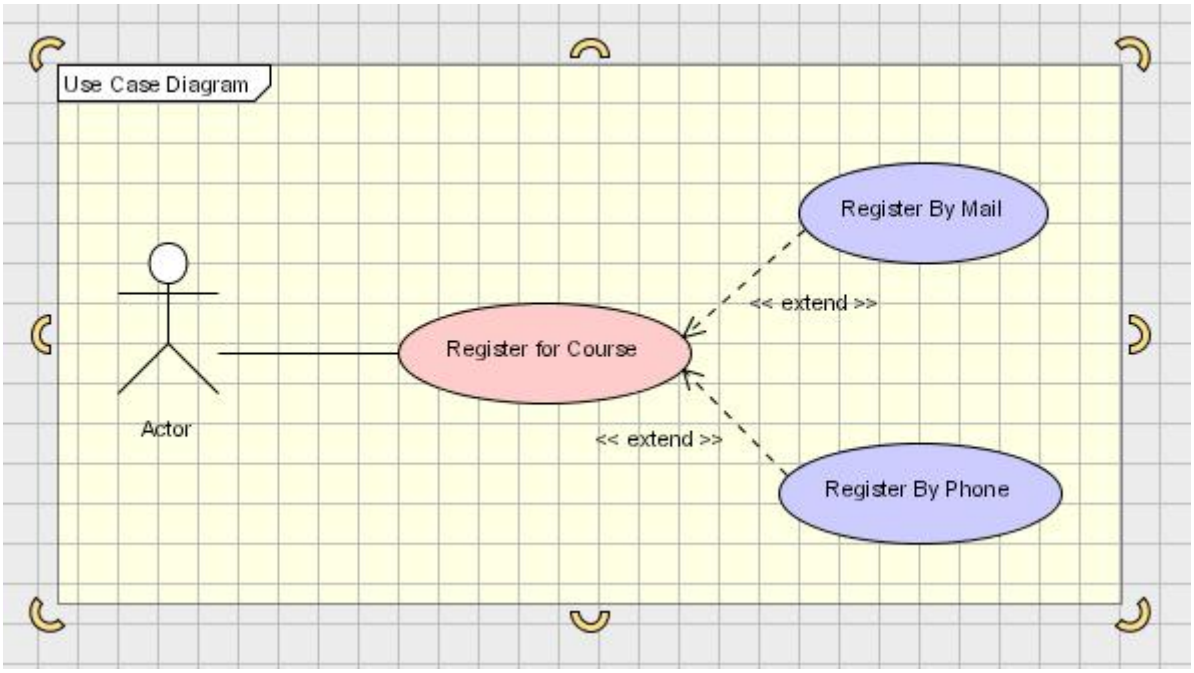
UML:Enumeration	com.topcoder.uml.model.core.classifiers.EnumerationImpl
.literal	
UML:Comment	com.topcoder.uml.model.core.auxiliaryelements.CommentImpl
.body	
.annotatedElement	
UML:TemplateParameter	com.topcoder.uml.model.core.auxiliaryelements.TemplateParameterImpl
.template	
.parameter	
.defaultElement	
UML:TemplateArgument	com.topcoder.uml.model.core.auxiliaryelements.TemplateArgumentImpl
.modelElement	
.binding	
UML:AssociationEnd	com.topcoder.uml.model.core.relationships.AssociationEndImpl
.association	
.qualifier	
.participant	
.specification	
UML:Association	com.topcoder.uml.model.core.relationships.AssociationImpl
.connection	
UML:Generalization	com.topcoder.uml.model.core.relationships.GeneralizationImpl
.discriminator	
.child	
.parent	
.powertype	
UML:AssociationClass	com.topcoder.uml.model.core.relationships.AssociationClassImpl
UML:Dependency	com.topcoder.uml.model.core.dependencies.DependencyImpl
.client	
.supplier	
UML:Binding	com.topcoder.uml.model.core.dependencies.BindingImpl
.argument	
UML:Abstraction	com.topcoder.uml.model.core.dependencies.AbstractionImpl
.mapping	
UML:Usage	com.topcoder.uml.model.core.dependencies.UsageImpl
UML:Stereotype	com.topcoder.uml.model.extensionmechanisms.StereotypeImpl
.icon	
.baseClass	
.definedTag	
.stereotypeConstraint	
UML:TagDefinition	com.topcoder.uml.model.extensionmechanisms.TagDefintionImpl
.tagType	
.multiplicity	
.owner	
UML:TaggedValue	com.topcoder.uml.model.extensionmechanisms.TaggedValueImpl
.dataValue	
.modelElement	
.type	
.referenceValue	
UML:Multiplicity	com.topcoder.uml.model.datatypes.MultiplicityImpl
.range	
UML:MultiplicityRange	com.topcoder.uml.model.datatypes.MultiplicityRangeImpl
.lower	
.upper	
.multiplicity	
UML:Expression	com.topcoder.uml.model.datatypes.expressions.ExpressionImpl
.language	
.body	
UML:BooleanExpression	com.topcoder.uml.model.datatypes.expressions.BooleanExpressionImpl
UML:MappingExpression	com.topcoder.uml.model.datatypes.expressions.MappingExpressionImpl
UML:ProcedureExpression	com.topcoder.uml.model.datatypes.expressions.PrecedureExpressionImpl
UML:TypeExpression	com.topcoder.uml.model.datatypes.expressions.TypeExpressionImpl
UML:ElementImport	com.topcoder.uml.model.modelmanagement.ElementImportImpl
.alias	
.package	
.importedElement	
UML:Package	com.topcoder.uml.model.modelmanagement.PackageImpl
.elementImport	
UML:Subsystem	com.topcoder.uml.model.modelmanagement.SubsystemImpl
UML:Model	com.topcoder.uml.model.modelmanagement.ModelImpl
UML:Actor	com.topcoder.uml.model.usecases.ActorImpl
UML:UseCase	com.topcoder.uml.model.usecases.UseCaseImpl
.extend	
.include	
.extensionPoint	
UML:Include	com.topcoder.uml.model.usecases.IncludeImpl
.addition	
.base	
UML:Extend	com.topcoder.uml.model.usecases.ExtendImpl
.condition	
.base	
.extension	
.extensionPoint	

4.3 Demo

4.3.1 Demonstrates the usage of XMIReader and XMI2ModelHandler to parse a xmi file for all the uml model elements except for State Machine and Activity Graph model elements

It is quite difficult to demo a handler, which is called internally by a parser.
The developer is encouraged to create a small demo based on parsing an actual UML diagram (for example class) with a number of basic elements on it.

Lets take this example as our example:



Here is the xmi for it:

```
<UML:Model xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f51' name = 'model 1' isSpecification = 'false'
           isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
   <UML:Namespace.ownedElement>
     <UML:Actor xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f49' name = 'Actor' visibility = 'public'
                isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
                isAbstract = 'false'/>
     <UML:Actor xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f3b' name = 'Actor' visibility = 'public'
                isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
                isAbstract = 'false'/>
     <UML:UseCase xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f34' name = 'Register for Course'
                  visibility = 'public' isSpecification = 'false' isRoot = 'false'
                  isLeaf = 'false' isAbstract = 'false'/>
     <UML:UseCase xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f2d' name = 'Register By Mail'
                  visibility = 'public' isSpecification = 'false' isRoot = 'false'
                  isLeaf = 'false' isAbstract = 'false'>
       <UML:UseCase.extend>
         <UML:Extend xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7f03'/>
       </UML:UseCase.extend>
     </UML:UseCase>
     <UML:Association xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f20' isSpecification = 'false'
                      isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
```

```
      <UML:Association.connection>
        <UML:AssociationEnd xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f26' visibility = 'public'
                            isSpecification = 'false' isNavigable = 'true'
                            ordering = 'unordered' aggregation = 'none'
          targetScope = 'instance' changeability = 'changeable'>
          <UML:AssociationEnd.multiplicity>
            <UML:Multiplicity xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f24'>
              <UML:Multiplicity.range>
                <UML:MultiplicityRange xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f25' lower = '1'
                                       upper = '1'/>
              </UML:Multiplicity.range>
            </UML:Multiplicity>
          </UML:AssociationEnd.multiplicity>
          <UML:AssociationEnd.participant>
            <UML:Actor xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7f3b'/>
          </UML:AssociationEnd.participant>
        </UML:AssociationEnd>
        <UML:AssociationEnd xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f23' visibility = 'public'
                            isSpecification = 'false' isNavigable = 'true'
                            ordering = 'unordered' aggregation = 'none'
                            targetScope = 'instance' changeability = 'changeable'>
          <UML:AssociationEnd.multiplicity>
            <UML:Multiplicity xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f21'>
              <UML:Multiplicity.range>
                <UML:MultiplicityRange xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f22' lower = '1'
                                       upper = '1'/>
              </UML:Multiplicity.range>
            </UML:Multiplicity>
          </UML:AssociationEnd.multiplicity>
          <UML:AssociationEnd.participant>
            <UML:UseCase xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7f34'/>
          </UML:AssociationEnd.participant>
        </UML:AssociationEnd>
      </UML:Association.connection>
    </UML:Association>
    <UML:Extend xmi.id = 'I1aa8eb7m10f6dbd4de0mm7f03' isSpecification = 'false'>
      <UML:Extend.base>
        <UML:UseCase xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7f34'/>
      </UML:Extend.base>
      <UML:Extend.extension>
        <UML:UseCase xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7f2d'/>
      </UML:Extend.extension>
    </UML:Extend>
    <UML:UseCase xmi.id = 'I1aa8eb7m10f6dbd4de0mm7ef4' name = 'Register By Phone'
                 visibility = 'public' isSpecification = 'false' isRoot = 'false'
                 isLeaf = 'false' isAbstract = 'false'>
      <UML:UseCase.extend>
        <UML:Extend xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7eed'/>
      </UML:UseCase.extend>
    </UML:UseCase>
    <UML:Extend xmi.id = 'I1aa8eb7m10f6dbd4de0mm7eed' isSpecification = 'false'>
      <UML:Extend.base>
        <UML:UseCase xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7f34'/>
      </UML:Extend.base>
      <UML:Extend.extension>
        <UML:UseCase xmi.idref = 'I1aa8eb7m10f6dbd4de0mm7ef4'/>
      </UML:Extend.extension>
    </UML:Extend>
  </UML:Namespace.ownedElement>
</UML:Model>
```

To build the model to this we would run this:

```
// get the associated uml model manager
XMI2ModelHandler handler = (XMI2ModelHandler) reader.getHandler("UML:Model");
UMLModelManager manager = handler.getUmlModelManager();

reader.parse(new File("test_files" + File.separator + "demo.xmi"));

// now we can access the model using xmi id values
```

```
Model model = (Model) reader.getElement("I1aa8eb7m10f6dbd4de0mm7f51");
System.out.println("Model(Got from the xmi id) name : " + model.getName());

// we can use the uml model manager to access the model as well
model = manager.getModel();
 System.out.println("Model(Got from the uml model manager) name : " + model.getName());
```

4.3.2 Demonstrates the functionality of ModelElementFactory

```
// Create a default instance
ModelElementFactory modelElementFactory = new ModelElementFactory();

// Create an instance with configuration data
modelElementFactory = new ModelElementFactory(ModelElementFactory.class.getName());

// Adds a new mapping to the class
modelElementFactory.addMapping("UML:Model",
    "com.topcoder.uml.model.modelmanagement.ModelImpl");

// Creates an actual instance maps to the specific xml element
modelElementFactory.createModelElement("UML:Model");

// Gets the class name for the given xml element
modelElementFactory.getMapping("UML:Model");

// Return the complete mapping
modelElementFactory.getAllMappings();

// Remove the xml element mapping
modelElementFactory.removeMapping("UML:Model");
```

5. Future Enhancements

- None at this time