

# **ATL Transformation Examples**

## **The MOF to UML ATL transformation *- version 0.1 -***

**September 2005**

**by**

***ATLAS group***

***LINA & INRIA***

***Nantes***

# Content

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
<b>2</b>	<b>The MOF to UML ATL transformation.....</b>	<b>1</b>
2.1	<i>Transformation overview.....</i>	<i>1</i>
2.2	<i>Metamodels.....</i>	<i>1</i>
2.3	<i>Rules specification .....</i>	<i>1</i>
2.4	<i>ATL code.....</i>	<i>2</i>
2.4.1	Helpers .....	2
2.4.2	Rules.....	2
<b>3</b>	<b>References .....</b>	<b>4</b>
<b>Appendix A</b>	<b>A simplified UML Core metamodel in KM3 format .....</b>	<b>5</b>
<b>Appendix B</b>	<b>A simplified MOF metamodel in KM3 format .....</b>	<b>9</b>
<b>Appendix C</b>	<b>The MOF to UML ATL code.....</b>	<b>12</b>



## 1 Introduction

The MOF (Meta Object Facility) [3] is an OMG standard enabling to describe metamodels through common semantics. The UML (Unified Modelling Language) Core standard [4] is the OMG common modelling language. Although, the MOF is primarily designed for metamodel definitions and UML Core for the design of models, the two standards handle very close notions. This document describes a transformation enabling to pass from the MOF to the UML semantics. The transformation is based on the UML Profile for MOF OMG specification [1]. Note that a similar UML Profile (for MOF) has been described in the scope of the NetBeans project [2].

## 2 The MOF to UML ATL transformation

### 2.1 Transformation overview

MOF to UML is a one-step transformation that produces a UML model from a MOF one. The UML models generated by this transformation are compliant with the Poseidon for UML tool [5].


### 2.2 Metamodels

The UML to MOF transformation is based on some subsets of the UML Core and the MOF metamodels. The exhaustive definition of these metamodels can be found in the OMG UML 1.5 specification [3] and OMG MOF 1.4 specification [4]. Appendix A and Appendix B respectively provide, expressed in the KM3 format [6], the UML and MOF metamodels that have been considered in the scope of this transformation.

### 2.3 Rules specification

The set of rules used to transform a MOF model into a UML model has been derived from the OMG UML Profile for MOF specification [1]:

- A UML Association, with its associated UML Generalizations, is generated from a MOF Association;
- A UML AssociationEnd, with its UML Multiplicity and its MultiplicityRange, is generated from a MOF AssociationEnd;
- A UML Parameter is generated from a MOF Parameter;
- A UML Attribute, with its UML Multiplicity and its MultiplicityRange, is generated from a MOF Attribute;
- A UML Class, with its associated UML Generalizations, is generated from a MOF Class. A given MOF Class is also associated with the root UML Model and the UML Stereotypes that may be required for the generated model;
- A UML Operation is generated from a MOF Operation;
- A UML Constraint is generated from a MOF Constraint;
- A UML TaggedValue is generated from a MOF Tag;
- A UML Import is generated from a MOF Dependency;
- A UML Package, with its associated UML Generalizations, is generated from a MOF Package.

	ATL Transformation Example	
	MOF to UML	Date 03/11/2005

## 2.4 ATL code

The ATL code for the MOF to UML transformation is provided in Appendix C. It consists of 11 helpers and 11 rules.

### 2.4.1 Helpers

The MOF to UML transformations define 4 constant helpers and 7 function ones. The **firstClass** constant helper

The **firstClass** constant helper calculates a MOF Class that is going to be considered as the reference class for the generation of unique elements (UML Model and Stereotypes) in the UML output model.

The **firstImport** constant helper calculates a sequence of MOF Import that is going to be considered as the reference for the generation of an “import” UML Stereotype. The helper selects a MOF Import among the clustered ones. The returned sequence contains 1 or 0 element (in case the MOF input model contains no clustered Import element).

The **firstClustered** constant helper is similar to the firstImport one, but builds a sequence of unclustered MOF Import elements.

The **firstMetamodel** constant helper calculates a sequence of MOF Package that is going to be considered as the reference for the generation of the “metamodel” UML Stereotype. The helper selects a MOF package among those of the input MOF model. The returned sequence contains 1 or 0 element (in case the MOF input model contains no Package element).

The **getOrdering()** and **getUMLOrdering()** helpers aim to translate the MOF boolean value encoding the ordering into a UML OrderingKind (`ok_unordered` / `ok_ordered`). The `getOrdering()` helper returns the UML OrderingKind that corresponds to the non-undefined ordering of the contextual MOF StructuralFeature or AssociationEnd. The `getUMLOrdering()` helper first checks whether the multiplicity, or the multiplicity.ordering attributes of the contextual element are undefined. In such a case, it returns the `ok_unordered` default ordering value. Otherwise, it returns the value provided by the call of `getOrdering()`.


The **getVisibility()** and **getUMLVisibility()** helpers aim to translate a MOF VisibilityKind data (`public_vis` / `private_vis` / `protected_vis`) into a UML VisibilityKind (`vk_public` / `vk_private` / `vk_protected`). The `getVisibility()` helper returns the UML visibility that corresponds to the non-undefined MOF visibility of the contextual model element. The `getUMLVisibility()` helper first checks whether the visibility of its contextual element is undefined. If so, it returns the `vk_public` default value. Otherwise, it returns the value computed by `getUMLVisibility()`.

The **getChangeability()** and **getUMLChangeability()** helpers aim to translate the MOF boolean value encoding changeability into a UML ChangeableKind (`ck_changeable` / `ck_frozen`). The `getChangeable()` helper returns the UML changeability that corresponds to the non-undefined MOF changeability of the contextual model element: `ck_changeable` if the `isChangeable` is true, `ck_frozen` otherwise. The `getUMLChangeability()` helper first checks whether the `isChangeable` attribute of its contextual element is undefined. If so, it returns the `ck_changeable` default value. Otherwise, it returns the value computed by `getUMLChangeability()`.

The **getUMLScope()** helper aims to translate a MOF ScopeKind data (`instance_level` / `classifier_level`) into a UML ScopeKind (`sk_instance` / `sk_classifier`). For this purpose, it returns the UML value that corresponds to the MOF value.

### 2.4.2 Rules

The **Association** rule generates a UML Association, along with its Generalization elements, for each MOF Association. The namespace element of the generated association corresponds to the container element of the input MOF Association. Its set of generalizations corresponds to the generalizations

	ATL Transformation Example	
	MOF to UML	Date 03/11/2005

generated by the rule. A Generalization is generated for each supertype of the input association. The namespace of each Generalization is initialized with the container of the input MOF Association. The child of a Generalization corresponds to the generated UML Association, whereas its parent corresponds to the currently iterated supertype of the input Association. Note that the constraint property is initialized by means of a collect operation due to the foreach instruction: it is therefore mandatory to assign each collection property with a collection of the same size than the one of the foreach reference collection (ma.supertype).

The **AssociationEnd** rule generates a UML AssociationEnd, with its Multiplicity and MultiplicityRange elements, for each MOF AssociationEnd. The association property of the generated AssociationEnd is set to the container of the input AssociationEnd. Its aggregation is translated from the MOF aggregation of the input element. Note that the targetScope, qualifier, and specification properties are set to default values. Its multiplicity is associated with the generated UML Multiplicity. The range of this last is associated with a single element set that contains the UML MultiplicityRange generated by the rule. Its lower and upper attribute are copied from the multiplicity of the input AssociationEnd.

The **Parameter** rule generates a UML Parameter for each MOF Parameter. Its kind is translated from the MOF kind of the input Parameter. The generated Parameter has no default value.


The **Attribute** rule generates a UML Attribute, with its Multiplicity and MultiplicityRange elements, for each MOF Attribute. As a UML Feature, the generated Attribute is attached to its container through its owner (and not its namespace) property. It is initialized with the container of the input MOF Attribute. Note that the targetScope of the generated Attribute is set to the [sk\\_instance](#) default value. The generated Attribute has no default value. Its multiplicity is associated with the generated UML Multiplicity. The range of this last is associated with a single element set that contains the UML MultiplicityRange generated by the rule. Its lower and upper attribute are copied from the multiplicity of the input Attribute.

The **FirstClass** rule generates a UML Class, along with its associated Generalization elements, as well as the UML Model and the UML Stereotypes unique elements, from the reference MOF Class that is computed by the firstClass helper. The namespace element of the generated class corresponds to the container element of the input MOF Class. The link to the elements contained by the generated Class is encoded by the feature property (and not the ownedElement one). It is initialized with the contents of the input MOF Class. The powertypeRange and isActive properties are set to default values. Its set of generalizations corresponds to the generalizations generated by the rule. A Generalization is generated for each supertype of the input class. The namespace of each Generalization is initialized with the container of the input MOF Class. The child of a Generalization corresponds to the generated UML Class, whereas its parent corresponds to the currently iterated supertype of the input Class. Note that the constraint property is initialized by means of a collect operation due to the foreach instruction: it is therefore mandatory to assign each collection property with a collection of the same size than the one of the foreach reference collection (mc.supertype).

The generated Model is simply initialized with a default name value. The different UML!Stereotype are generated if their respective reference Sequences are not empty. Each stereotype is initialized with its name ('clustering', 'import' or 'metamodel') and the name of the base class it is associated with (respectively Dependency for the 2 first ones, and Package). Their namespace is set to the UML!Model generated by the rule.

The **OtherClass** rule is similar to the previous one, except that it applies to the MOF Classes that are different from the one provided by the firstClass helper. It only generates UML Classes along with their Generalization elements.

The **Operation** rule generates a UML Operation from a MOF Operation. Like an Attribute, as a Feature element, each generated UML Operation is attached to its container by the owner property which is set to the container of the input MOF Operation. The parameter of the generated Operation is initialized with the contents of the MOF Operation. Finally, the concurrency, isAbstract, isLeaf, and isRoot properties are set to default values.

	ATL Transformation Example	
	MOF to UML	Date 03/11/2005

The **Constraint** rule generates a UML Constraint from a MOF Constraint. The namespace of the generated Constraint is initialized with the container of the input MOF Constraint.

The **TaggedValue** rule generates a UML TaggedValue from a MOF Tag. The namespace of the generated TaggedValue is initialized with the container of the input MOF Tag. Its dataValue property corresponds to the values of the MOF Tag, whereas its tag is initialized with the tagId of the MOF Tag. The model element to which the generated tag refers corresponds to the first element of the elements set of the MOF Tag. The referenceValue property is initialized with an empty set.

The **Import** rule generates a UML Dependency from a MOF Import. The namespace of the generated Dependency is initialized with the container of the input MOF Import. If the isClustered attribute of the input Import is true, the generated Dependency is associated with a “clustered” stereotype, otherwise it is associated with an “import” stereotype. The client elements of the generated Dependency correspond to a Sequence composed of the only container of the input Import. Its set of supplier elements is composed of the importedNamespace of the input Import.

The **Package** rule generates a UML Package, along with its Generalization elements, for each MOF Package. The namespace element of the generated package corresponds to the container element of the input MOF Package. The link to the elements contained by the generated Package is encoded by the ownedElement property, and is initialized with the contents of the input MOF Class. The generated UML Package is associated with the “metamodel” stereotype. A Generalization is generated for each supertype of the input package. The namespace of each Generalization is initialized with the container of the input MOF Package. The child of a Generalization corresponds to the generated UML Package, whereas its parent corresponds to the currently iterated supertype of the input Package. Note that the constraint property is initialized by means of a collect operation due to the foreach instruction: it is therefore mandatory to assign each collection property with a collection of the same size than the one of the foreach reference collection (mp.supertype).

### 3 References

- [1] OMG/UML Profile for MOF, OMG Formal Specification. formal/04-02-06, 2004. Available at <http://www.omg.org/docs/formal/04-02-06.pdf>.
- [2] NetBeans/Sun Microsystems. UML Profile for MOF. Available at <http://mdr.netbeans.org/uml2mof/profile.html>.
- [3] OMF/UML (Unified Modeling Language) 1.5 specification. formal/03-03-01, 2003.
- [4] OMG/MOF Meta Object Facility (MOF) 1.4 specification. formal/2002-04-03, 2002.
- [5] Gentleware. Poseidon for UML, information and download available at <http://www.gentleware.com/index.php>.
- [6] KM3 User Manual. The Eclipse Generative Model Transformer (GMT) project, <http://eclipse.org/gmt/>.

## Appendix A A simplified UML Core metamodel in KM3 format

```
1 package Core {
2     abstract class Element {
3     }
4
5     abstract class ModelElement extends Element {
6         reference taggedValue[*] container : TaggedValue oppositeOf modelElement;
7         reference clientDependency[*] : Dependency oppositeOf client;
8         reference constraint[*] : Constraint oppositeOf constrainedElement;
9         reference stereotype[*] : Stereotype;
10        reference comment[*] : Comment oppositeOf annotatedElement;
11        reference sourceFlow[*] : Flow oppositeOf source;
12        reference targetFlow[*] : Flow oppositeOf target;
13        reference templateParameter[*] ordered container : TemplateParameter oppositeOf
14    template;
15        reference namespace[0-1] : Namespace oppositeOf ownedElement;
16        attribute name[0-1] : String;
17        attribute visibility[0-1] : VisibilityKind;
18        attribute isSpecification : Boolean;
19    }
20
21
22    abstract class GeneralizableElement extends ModelElement {
23        reference generalization[*] : Generalization oppositeOf child;
24        attribute isRoot : Boolean;
25        attribute isLeaf : Boolean;
26        attribute isAbstract : Boolean;
27    }
28
29
30    abstract class Namespace extends ModelElement {
31        reference ownedElement[*] container : ModelElement oppositeOf namespace;
32    }
33
34    abstract class Classifier extends GeneralizableElement, Namespace {
35        reference powertypeRange[*] : Generalization oppositeOf powertype;
36        reference feature[*] ordered container : Feature oppositeOf owner;
37    }
38
39    class Class extends Classifier {
40        attribute isActive : Boolean;
41    }
42
43    class DataType extends Classifier {
44    }
45
46    abstract class Feature extends ModelElement {
47        reference owner[0-1] : Classifier oppositeOf feature;
48        attribute ownerScope : ScopeKind;
49    }
50
51    abstract class StructuralFeature extends Feature {
52        reference type : Classifier;
53        attribute multiplicity[0-1] : Multiplicity;
54        attribute changeability[0-1] : ChangeableKind;
55        attribute targetScope[0-1] : ScopeKind;
56        attribute ordering[0-1] : OrderingKind;
57    }
58
59    class AssociationEnd extends ModelElement {
```

```
60         reference association : Association oppositeOf connection;
61         reference specification[*] : Classifier;
62         reference participant : Classifier;
63         reference qualifier[*] ordered container : Attribute oppositeOf associationEnd;
64         attribute isNavigable : Boolean;
65         attribute ordering[0-1] : OrderingKind;
66         attribute aggregation[0-1] : AggregationKind;
67         attribute targetScope[0-1] : ScopeKind;
68         attribute multiplicity[0-1] : Multiplicity;
69         attribute changeability[0-1] : ChangeableKind;
70     }
71
72     class Interface extends Classifier {
73     }
74
75     class Constraint extends ModelElement {
76         reference constrainedElement[*] ordered : ModelElement oppositeOf constraint;
77         attribute body[0-1] : BooleanExpression;
78     }
79
80     abstract class Relationship extends ModelElement {
81     }
82
83     class Association extends GeneralizableElement, Relationship {
84         reference connection[2-*] ordered container : AssociationEnd oppositeOf
85 association;
86     }
87
88     class Attribute extends StructuralFeature {
89         reference associationEnd[0-1] : AssociationEnd oppositeOf qualifier;
90         attribute initialValue[0-1] : Expression;
91     }
92
93     abstract class BehavioralFeature extends Feature {
94         reference parameter[*] ordered container : Parameter oppositeOf
95 behavioralFeature;
96         attribute isQuery : Boolean;
97     }
98
99     class Operation extends BehavioralFeature {
100         attribute concurrency[0-1] : CallConcurrencyKind;
101         attribute isRoot : Boolean;
102         attribute isLeaf : Boolean;
103         attribute isAbstract : Boolean;
104         attribute specification[0-1] : String;
105     }
106
107     class Parameter extends ModelElement {
108         reference type : Classifier;
109         reference behavioralFeature[0-1] : BehavioralFeature oppositeOf parameter;
110         attribute defaultValue[0-1] : Expression;
111         attribute kind : ParameterDirectionKind;
112     }
113
114     class Method extends BehavioralFeature {
115         reference specification : Operation;
116         attribute body : ProcedureExpression;
117     }
118
119     class Generalization extends Relationship {
120         reference parent : GeneralizableElement;
121         reference powertype[0-1] : Classifier oppositeOf powertypeRange;
122         reference child : GeneralizableElement oppositeOf generalization;
123         attribute discriminator[0-1] : String;
124     }
125
126     class AssociationClass extends Association, Class {
127     }
128
```





```
129     class Dependency extends Relationship {
130         reference client[1-*] : ModelElement oppositeOf clientDependency;
131         reference supplier[1-*] : ModelElement;
132     }
133
134     class Abstraction extends Dependency {
135         attribute mapping[0-1] : MappingExpression;
136     }
137
138     abstract class PresentationElement extends Element {
139         reference subject[*] : ModelElement;
140     }
141
142     class Usage extends Dependency {
143     }
144
145     class Binding extends Dependency {
146         reference argument[1-*] ordered container : TemplateArgument oppositeOf
147 binding;
148     }
149
150     class Component extends Classifier {
151         reference deploymentLocation[*] : Node oppositeOf deployedComponent;
152         reference residentElement[*] container : ElementResidence oppositeOf
153 "container";
154         reference implementation[*] : Artifact;
155     }
156
157     class Node extends Classifier {
158         reference deployedComponent[*] : Component oppositeOf deploymentLocation;
159     }
160
161     class Permission extends Dependency {
162     }
163
164     class Comment extends ModelElement {
165         reference annotatedElement[*] : ModelElement oppositeOf comment;
166         attribute body : String;
167     }
168
169     class Flow extends Relationship {
170         reference source[*] : ModelElement oppositeOf sourceFlow;
171         reference target[*] : ModelElement oppositeOf targetFlow;
172     }
173
174     class ElementResidence {
175         reference "container" : Component oppositeOf residentElement;
176         reference resident : ModelElement;
177         attribute visibility[0-1] : VisibilityKind;
178     }
179
180     class TemplateParameter {
181         reference template : ModelElement oppositeOf templateParameter;
182         reference parameter container : ModelElement;
183         reference defaultElement[0-1] : ModelElement;
184     }
185
186     class Primitive extends DataType {
187     }
188
189     class Enumeration extends DataType {
190         reference "literal"[1-*] ordered container : EnumerationLiteral oppositeOf
191 "enumeration";
192     }
193
194     class EnumerationLiteral extends ModelElement {
195         reference "enumeration" : Enumeration oppositeOf "literal";
196     }
197
```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
198     class Stereotype extends GeneralizableElement {
199         reference stereotypeConstraint[*] container : Constraint;
200         reference definedTag[*] container : TagDefinition oppositeOf owner;
201         attribute icon[0-1] : String;
202         attribute baseClass[1-*] : String;
203     }
204
205     class TagDefinition extends ModelElement {
206         reference owner[0-1] : Stereotype oppositeOf definedTag;
207         attribute tagType[0-1] : String;
208         attribute multiplicity[0-1] : Multiplicity;
209     }
210
211     class TaggedValue extends ModelElement {
212         reference type : TagDefinition;
213         reference referenceValue[*] : ModelElement;
214         reference modelElement : ModelElement oppositeOf taggedValue;
215         attribute dataValue[*] : String;
216     }
217
218     class ProgrammingLanguageDataType extends DataType {
219         attribute expression : TypeExpression;
220     }
221
222     class Artifact extends Classifier {
223     }
224
225     class TemplateArgument {
226         reference binding : Binding oppositeOf argument;
227         reference modelElement : ModelElement;
228     }
229 }
```

## Appendix B A simplified MOF metamodel in KM3 format

```
1 package Model {
2
3     abstract class ModelElement {
4         -- derived
5         reference requiredElements[*] : ModelElement;
6         reference constraints[*] : Constraint oppositeOf constrainedElements;
7         reference "container"[0-1] : Namespace oppositeOf contents;
8         attribute name : String;
9         -- derived
10        attribute qualifiedName[1-*] ordered : String;
11        attribute annotation : String;
12        operation findRequiredElements(kinds : String, recursive : Boolean) :
13        ModelElement;
14        operation isRequiredBecause(otherElement : ModelElement, reason : String) :
15        Boolean;
16        operation isFrozen() : Boolean;
17        operation isVisible(otherElement : ModelElement) : Boolean;
18    }
19
20    enumeration VisibilityKind {
21        literal public_vis;
22        literal protected_vis;
23        literal private_vis;
24    }
25
26    abstract class Namespace extends ModelElement {
27        reference contents[*] ordered container : ModelElement oppositeOf "container";
28        operation lookupElement(name : String) : ModelElement;
29        operation resolveQualifiedName(qualifiedName : String) : ModelElement;
30        operation findElementsByType(ofType : Class, includeSubtypes : Boolean) :
31        ModelElement;
32        operation nameIsValid(proposedName : String) : Boolean;
33    }
34
35    abstract class GeneralizableElement extends Namespace {
36        reference supertypes[*] ordered : GeneralizableElement;
37        attribute isRoot : Boolean;
38        attribute isLeaf : Boolean;
39        attribute isAbstract : Boolean;
40        attribute visibility : VisibilityKind;
41        operation allSupertypes() : GeneralizableElement;
42        operation lookupElementExtended(name : String) : ModelElement;
43        operation findElementsByTypeExtended(ofType : Class, includeSubtypes : Boolean)
44        : ModelElement;
45    }
46
47    abstract class TypedElement extends ModelElement {
48        reference type : Classifier;
49    }
50
51    abstract class Classifier extends GeneralizableElement {
52    }
53
54    class Class extends Classifier {
55        attribute isSingleton : Boolean;
56    }
57
58    class MultiplicityType {
```

```
60         attribute lower : Integer;
61         attribute upper : Integer;
62         attribute isOrdered : Boolean;
63         attribute isUnique : Boolean;
64     }
65
66     abstract class DataType extends Classifier {
67
68     }
69
70     class PrimitiveType extends DataType {
71
72     }
73
74     class EnumerationType extends DataType {
75         attribute labels[1-*] ordered : String;
76     }
77
78     class CollectionType extends DataType, TypedElement {
79         attribute multiplicity : MultiplicityType;
80     }
81
82     class StructureType extends DataType {
83
84     }
85
86     class StructureField extends TypedElement {
87
88     }
89
90     class AliasType extends DataType, TypedElement {
91
92     }
93
94     enumeration ScopeKind {
95         literal instance_level;
96         literal classifier_level;
97     }
98
99     abstract class Feature extends ModelElement {
100         attribute scope : ScopeKind;
101         attribute visibility : VisibilityKind;
102     }
103
104     abstract class StructuralFeature extends Feature, TypedElement {
105         attribute multiplicity : MultiplicityType;
106         attribute isChangeable : Boolean;
107     }
108
109     class Attribute extends StructuralFeature {
110         attribute isDerived : Boolean;
111     }
112
113     class Reference extends StructuralFeature {
114         reference referencedEnd : AssociationEnd;
115         -- derived
116         reference exposedEnd : AssociationEnd;
117     }
118
119     abstract class BehavioralFeature extends Feature, Namespace {
120
121     }
122
123     class Operation extends BehavioralFeature {
124         reference exceptions[*] ordered : Exception;
125         attribute isQuery : Boolean;
126     }
127
128     class Exception extends BehavioralFeature {
```

```
129
130     }
131
132     class Association extends Classifier {
133         attribute isDerived : Boolean;
134     }
135
136     enumeration AggregationKind {
137         literal none;
138         literal shared;
139         literal composite;
140     }
141
142     class AssociationEnd extends TypedElement {
143         attribute isNavigable : Boolean;
144         attribute aggregation : AggregationKind;
145         attribute multiplicity : MultiplicityType;
146         attribute isChangeable : Boolean;
147         operation otherEnd() : AssociationEnd;
148     }
149
150     class Package extends GeneralizableElement {
151     }
152
153
154     class Import extends ModelElement {
155         reference importedNamespace : Namespace;
156         attribute visibility : VisibilityKind;
157         attribute isClustered : Boolean;
158     }
159
160     enumeration DirectionKind {
161         literal in_dir;
162         literal out_dir;
163         literal inout_dir;
164         literal return_dir;
165     }
166
167     class Parameter extends TypedElement {
168         attribute direction : DirectionKind;
169         attribute multiplicity : MultiplicityType;
170     }
171
172     class Constraint extends ModelElement {
173         reference constrainedElements[1-*] : ModelElement oppositeOf constraints;
174         attribute expression : String;
175         attribute language : String;
176         attribute evaluationPolicy : EvaluationKind;
177     }
178
179     enumeration EvaluationKind {
180         literal immediate;
181         literal deferred;
182     }
183
184     class Constant extends TypedElement {
185         attribute value : String;
186     }
187
188     class Tag extends ModelElement {
189         reference elements[1-*] : ModelElement;
190         attribute tagId : String;
191         attribute values[*] ordered : String;
192     }
193 }
```

## Appendix C The MOF to UML ATL code

```
1  module MOF2UML;
2  create OUT : UML from IN : MOF;
3
4
5  uses strings;
6
7
8  -----
9  -- HELPERS -----
10 -----
11
12 -- This helper returns a MOF!Class that is considered as the reference Class
13 -- for the generation of unique target elements: the model and the possible
14 -- stereotypes.
15 -- CONTEXT: thisModule
16 -- RETURN: MOF!Class
17 helper def: firstClass : MOF!Class =
18     MOF!Class.allInstancesFrom('IN')->asSequence()->first();
19
20 -- This helper returns a clustered MOF!Import that is considered as the
21 -- reference Import for the generation of the 'clustered' stereotype.
22 -- CONTEXT: thisModule
23 -- RETURN: Sequence(MOF!Import)
24 helper def: firstClustered : Sequence(MOF!Import) =
25     Sequence{
26         MOF!Import.allInstancesFrom('IN')
27             ->select(e | e.isClustered)
28             ->asSequence()->first()
29     };
30
31 -- This helper returns an unclustered MOF!Import that is considered as the
32 -- reference Import for the generation of the 'import' stereotype.
33 -- CONTEXT: thisModule
34 -- RETURN: Set(MOF!Import)
35 helper def: firstImport : Sequence(MOF!Import) =
36     Sequence{
37         MOF!Import.allInstancesFrom('IN')
38             ->select(e | not e.isClustered)
39             ->asSequence()->first()
40     };
41
42 -- This helper returns a MOF!Package that is considered as the reference
43 -- Package for the generation of the 'import' stereotype.
44 -- CONTEXT: thisModule
45 -- RETURN: Set(MOF!Package)
46 helper def: firstMetamodel : Sequence(MOF!Package) =
47     Sequence{
48         MOF!Package.allInstancesFrom('IN')->asSequence()->first()
49     };
50
51 -- This helper returns the UML!OrderingKind that corresponds to the
52 -- non undefined MOF!ScopeKind of the contextual MOF!ModelElement.
53 -- The helper returns the '#ordered' or '#unordered' value depending on the
54 -- value of the MOF 'isOrdered' boolean attribute.
55 --
56 -- WARNING: the contextual MOF!ModelElement must be of either a
57 -- MOF!StructuralFeature or a MOF!AssociationEnd element.
58 --
59 -- CONTEXT: MOF!ModelElement
60 -- RETURN: UML!OrderingKind
61 helper context MOF!ModelElement def: getOrdering() : UML!OrderingKind =
62     if self.multiplicity.isOrdered
63     then
```

```
64         #ok_ordered
65     else
66         #ok_unordered
67     endif;
68
69 -- This helper returns the UML!OrderingKind that corresponds to the
70 -- MOF!ScopeKind of the contextual MOF!ModelElement.
71 -- If the multiplicity attribute of the contextual ModelElement, or its
72 -- isOrdered attribute is undefined, the helper returns 'ok_unordered'.
73 -- Otherwise, the helper returns the value computed by getOrdering().
74 --
75 -- WARNING: the contextual MOF!ModelElement must be of either a
76 -- MOF!StructuralFeature or a MOF!AssociationEnd element.
77 --
78 -- CONTEXT: MOF!ModelElement
79 -- RETURN: UML!OrderingKind
80 helper context MOF!ModelElement def: getUMLOrdering() : UML!OrderingKind =
81     if self.multiplicity.ocIsUndefined()
82     then
83         #ok_unordered
84     else
85         if self.multiplicity.isOrdered.ocIsUndefined()
86         then
87             #ok_unordered
88         else
89             self.getOrdering()
90         endif
91     endif;
92
93 -- This helper returns the UML!Visibility that corresponds to the
94 -- non undefined MOF!Visibility of the contextual MOF!ModelElement.
95 --
96 -- WARNING: the contextual MOF!ModelElement must be of either a MOF!Feature, a
97 -- MOF!Import or a MOF!GeneralizableElement entity.
98 --
99 -- CONTEXT: MOF!ModelElement
100 -- RETURN: UML!Visibility
101 helper context MOF!ModelElement def: getVisibility() : UML!Visibility =
102     let v : MOF!Visibility = self.visibility in
103     if v = #public_vis
104     then
105         #vk_public
106     else
107         if v = #protected_vis
108         then
109             #vk_protected
110         else
111             if v = #private_vis
112             then
113                 #vk_protected
114             else -- default
115                 #vk_public
116             endif
117         endif
118     endif;
119
120 -- This helper returns the UML!Visibility that corresponds to the
121 -- MOF!Visibility of the contextual MOF!ModelElement.
122 -- If the visibility of the contextual ModelElement is undefined, the helper
123 -- returns 'vk_public', otherwise, it returns the value provided by
124 -- getVisibility().
125 --
126 -- WARNING: the contextual MOF!ModelElement must be of either a MOF!Feature, a
127 -- MOF!Import or a MOF!GeneralizableElement entity.
128 --
129 -- CONTEXT: MOF!ModelElement
130 -- RETURN: UML!Visibility
131 helper context MOF!ModelElement def: getUMLVisibility() : UML!Visibility =
132     if self.visibility.ocIsUndefined()
```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
133         then
134             #vk_public
135         else
136             self.getVisibility()
137         endif;
138
139 -- This helper returns the UML!ChangeableKind that corresponds to the
140 -- non-undefined MOF!ChangeableKind of the contextual MOF!ModelElement.
141 -- The helper returns the '#ck_changable' or '#ck_frozen' value depending on
142 -- the value of the MOF 'isChangeable' boolean attribute.
143 --
144 -- WARNING: the contextual MOF!ModelElement must be of either a
145 -- MOF!StructuralFeature or a MOF!AssociationEnd element.
146 --
147 -- CONTEXT: MOF!ModelElement
148 -- RETURN: UML!ChangeableKind
149 helper context MOF!ModelElement def: getChangeability() : UML!ChangeableKind =
150     if self.isChangeable
151     then
152         #ck_changeable
153     else
154         #ck_frozen
155     endif;
156
157 -- This helper returns the UML!ChangeableKind that corresponds to the
158 -- MOF!ChangeableKind of the contextual MOF!ModelElement.
159 -- If changeability of the contextual MOF!ModelElement is undefined, the helper
160 -- returns the '#ck_changable' default value. Otherwise, it returns the value
161 -- computed by the getChangeability helper.
162 --
163 -- WARNING: the contextual MOF!ModelElement must be of either a
164 -- MOF!StructuralFeature or a MOF!AssociationEnd element.
165 --
166 -- CONTEXT: MOF!ModelElement
167 -- RETURN: UML!ChangeableKind
168 helper context MOF!ModelElement
169     def: getUMLChangeability() : UML!ChangeableKind =
170     if not self.isChangeable.ocllsUndefined()
171     then
172         self.getChangeability()
173     else
174         #ck_changable
175     endif;
176
177 -- This helper returns the UML!ScopeKind that corresponds to the MOF!ScopeKind
178 -- of the contextual MOF!Feature.
179 -- CONTEXT: MOF!Feature
180 -- RETURN: UML!ScopeKind
181 helper context MOF!Feature def: getUMLScope() : UML!ScopeKind =
182     if self.scope = #instance_level
183     then
184         #sk_instance
185     else
186         #sk_classifier
187     endif;
188
189
190 -----
191 -- RULES -----
192 -----
193
194 -- Rule 'Association'
195 -- This rule generates a UML!Association, along with its associated
196 -- UML!Generalizations from a MOF!Association.
197 -- Most properties of the generated association are copied from the input MOF
198 -- association properties. Its generalizations correspond to the Generalization
199 -- that are generated by the rule, whereas its specializations correspond to
200 -- the UML!Associations that are generated for the MOF!Associations that have
201 -- the input association as supertype.
```





## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
202 -- A UML!Generalization is generated fore each supertype of the input
203 -- MOF!Association. Its child corresponds to the generated UML association,
204 -- whereas its parent corresponds to the UML!Association generated for the
205 -- currently iterated supertype. Note that discriminator and powertype of the
206 -- generated Generalizations are set to default values since MOF defines no
207 -- corresponding properties.
208 rule Association {
209     from
210         ma : MOF!Association
211     to
212         ua : UML!Association (
213             -- Begin bindings inherited from ModelElement
214             name <- ma.name,
215             constraint <- ma.constraints,
216             namespace <- ma.container,
217             visibility <- ma.getUMLVisibility(),
218             -- taggedValue <-,
219             -- asArgument <-,
220             -- clientDependency <-,
221             -- implementationLocation <-,
222             -- presentation <-,
223             -- supplierDependency <-,
224             -- templateParameter <-,
225             -- stereotype<-,
226             -- End of bindings inherited from ModelElement
227
228             -- Begin bindings inherited from GeneralizableElement
229             isAbstract <- ma.isAbstract,
230             isLeaf <- ma.isLeaf,
231             isRoot <- ma.isRoot,
232             generalization <- mr
233             -- End of bindings inherited from GeneralizableElement
234         ),
235
236         mr : distinct UML!Generalization foreach(e in ma.supertypes) (
237             -- Begin bindings inherited from ModelElement
238             name <- ma.name,
239             constraint <- ma.supertypes->collect(e | e.constraints),
240             namespace <- ma.container,
241             visibility <- ma.getUMLVisibility(),
242             -- taggedValue <-,
243             -- asArgument <-,
244             -- clientDependency <-,
245             -- implementationLocation <-,
246             -- presentation <-,
247             -- supplierDependency <-,
248             -- templateParameter <-,
249             -- stereotype<-,
250             -- End of bindings inherited from ModelElement
251
252             child <- ua,
253             parent <- e,
254             discriminator <- '',
255             powertype <- OclUndefined
256         )
257     }
258
259 -- Rule 'AssociationEnd'
260 -- This rule generates a UML!AssociationEnd, along with its UML!Multiplicity,
261 -- and the MultiplicityRange of this last, from a MOF!AssociationEnd.
262 -- Most properties of the generated AssociationEnd are copied from those of
263 -- the input MOF AssociationEnd. Its multiplicity reference points to the
264 -- Multiplicity entity generated by the rule. The targetScope, qualifier and
265 -- specification properties are set to default values (MOF does not define
266 -- corresponding properties).
267 -- The range of the generated Multiplicity element is computed from the
268 -- multiplicity attribute of the input MOF!AssociationEnd.
269 rule AssociationEnd {
270     from
```

```

271         ma : MOF!AssociationEnd
272     to
273         ua : UML!AssociationEnd (
274             -- Begin bindings inherited from ModelElement
275             name <- ma.name,
276             constraint <- ma.constraints,
277             namespace <- ma.container,
278             -- visibility <-,
279             -- taggedValue <-,
280             -- asArgument <-,
281             -- clientDependency <-,
282             -- implementationLocation <-,
283             -- presentation <-,
284             -- supplierDependency <-,
285             -- templateParameter <-,
286             -- stereotype<-,
287             -- End of bindings inherited from ModelElement
288
289             association <- ma.container,
290             aggregation <-
291                 if ma.aggregation = #none
292                 then
293                     #ak_none
294                 else
295                     if ma.aggregation = #shared
296                     then
297                         #ak_aggregate
298                     else
299                         -- ma.aggregation = #composite
300                         #ak_composite
301                     endif
302                 endif,
303             changeability <- ma.getUMLChangeability(),
304             ordering <- ma.getUMLOrdering(),
305             isNavigable <- ma.isNavigable,
306             multiplicity <- um,
307             targetScope <- #sk_instance,
308             qualifier <- Sequence{},
309             specification <- Set{},
310             participant <- ma.type
311         ),
312         um : UML!Multiplicity (
313             range <- Set{ur}
314         ),
315         ur : UML!MultiplicityRange (
316             lower <- ma.multiplicity.lower,
317             upper <- ma.multiplicity.upper,
318             multiplicity <- um
319         )
320     )
321 }
322
323 -- Rule 'Parameter'
324 -- This rule generates a UML!Parameter from a MOF!Parameter.
325 -- Properties of the generated Parameter are copied from those of the input
326 -- Parameter, except the UML defaultValue attribute which has no MOF
327 -- equivalent. It is therefore set to 'oclUndefined'.
328 rule Parameter {
329     from
330         mp : MOF!Parameter
331     to
332         up : UML!Parameter (
333             -- Begin bindings inherited from ModelElement
334             name <- mp.name,
335             constraint <- mp.constraints,
336             -- namespace <- mp.container,
337             -- visibility <-,
338             -- taggedValue <-,
339             -- implementationLocation <-,

```

```

340 --      presentation <-,
341 --      supplierDependency <-,
342 --      templateParameter <-,
343 --      asArgument <-,
344 --      clientDependency <-,
345 --      stereotype<-,
346 --      -- End of bindings inherited from ModelElement
347
348      kind <-
349          if mp.direction = #in_dir
350          then
351              #pdk_in
352          else
353              if mp.direction = #inout_dir
354              then
355                  #pdk_inout
356              else
357                  if mp.direction = #out_dir
358                  then
359                      #pdk_out
360                  else -- mp.direction = #return_dir
361                      #pdk_return
362                  endif
363              endif
364          endif,
365      type <- mp.type,
366      defaultValue <- OclUndefined
367  )
368 }
369
370 -- Rule 'Attribute'
371 -- This rule generates a UML!Attribute, along with its UML!Multiplicity, and
372 -- the UML!MultiplicityRange of this last, from a MOF!Attribute.
373 -- Most properties of the generated Attribute are copied from those of the
374 -- input MOF Attribute. Its multiplicity reference points to the Multiplicity
375 -- entity generated by the rule. The targetScope and initialValue properties
376 -- are set to default values (MOF does not define corresponding properties):
377 -- 'sk_instance' for targetScope and 'oclUndefined' for initialValue.
378 -- The range of the generated Multiplicity element is computed from the
379 -- multiplicity attribute of the input MOF!Attribute.
380 rule Attribute {
381     from
382         ma : MOF!Attribute
383     to
384         ua : UML!Attribute (
385             -- Begin bindings inherited from ModelElement
386             name <- ma.name,
387             constraint <- ma.constraints,
388             -- namespace <- ma.container,
389             visibility <- ma.getUMLVisibility(),
390             -- taggedValue <-,
391             -- asArgument <-,
392             -- clientDependency <-,
393             -- implementationLocation <-,
394             -- presentation <-,
395             -- supplierDependency <-,
396             -- templateParameter <-,
397             -- stereotype<-,
398             -- End of bindings inherited from ModelElement
399
400             -- Begin bindings inherited from Feature
401             ownerScope <- ma.getUMLScope(),
402             owner <- ma.container,
403             -- End of bindings inherited from Feature
404
405             -- Begin bindings inherited from StructuralFeature
406             changeability <- ma.getUMLChangeability(),
407             multiplicity <- um,
408             ordering <- ma.getUMLOrdering(),

```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
409         type <- ma.type,
410         targetScope <- #sk_instance,
411         -- End of bindings inherited from StructuralFeature
412
413         initialValue <- OclUndefined
414     ),
415
416     um : UML!Multiplicity (
417         range <- Set{ur}
418     ),
419
420     ur : UML!MultiplicityRange (
421         lower <- ma.multiplicity.lower,
422         upper <- ma.multiplicity.upper,
423         multiplicity <- um
424     )
425 }
426
427 -- Rule 'Class'
428 -- This rule generates a UML!Class, along with its associated
429 -- UML!Generalizations, the UML!Model, and the 'metamodel', 'import', and
430 -- 'clustering' UML!Stereotype from the reference MOF!Class provided by the
431 -- firstClass helper.
432 -- Most properties of the generated Class are copied from the input MOF!Class
433 -- properties. Its generalizations correspond to the Generalization that are
434 -- generated by the rule, whereas its specializations correspond to the
435 -- UML!Classes that are generated for the MOF!Classes that have the input Class
436 -- as supertype. The powertypeRange and isActive properties, which have no
437 -- equivalent in MOF, are set to default values.
438 -- A UML!Generalization is generated fore each supertype of the input
439 -- MOF!Class. Its child corresponds to the generated UML class, whereas its
440 -- parent corresponds to the UML!Class generated for the currently iterated
441 -- supertype. Note that discriminator and powertype of the generated
442 -- Generalizations are set to default values since MOF defines no corresponding
443 -- properties.
444 -- The generated Model is simply initialized with a default name value.
445 -- The different UML!Stereotype are generated if their respective reference
446 -- Sequences are not empty. Each stereotype is initialized with its name
447 -- ('clustering', 'import' or 'metamodel') and the name of the base class it is
448 -- associated with (respectively Dependency for the 2 first ones, and Package).
449 -- Their namespace is set to the UML!Model ('mo') generated by the rule.
450 rule FirstClass {
451     from
452         mc : MOF!Class (
453             mc = thisModule.firstClass
454         )
455     to
456         uc : UML!Class (
457             -- Begin bindings inherited from ModelElement
458             name <- mc.name,
459             constraint <- mc.constraints,
460             namespace <- mc.container,
461             visibility <- mc.getUMLVisibility(),
462             -- taggedValue <-,
463             -- asArgument <-,
464             -- clientDependency <-,
465             -- implementationLocation <-,
466             -- presentation <-,
467             -- supplierDependency <-,
468             -- templateParameter <-,
469             -- stereotype<-,
470             -- End of bindings inherited from ModelElement
471
472             -- Begin bindings inherited from GeneralizableElement
473             isAbstract <- mc.isAbstract,
474             isLeaf <- mc.isLeaf,
475             isRoot <- mc.isRoot,
476             generalization <- mr,
477             -- End of bindings inherited from GeneralizableElement
```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
478
479
480 -- Begin bindings inherited from Namespace
481 -- ownedElement <- mc.contents,
482 -- End of bindings inherited from Namespace
483
484 -- Begin bindings inherited from Classifier
485 feature <- mc.contents,
486 powertypeRange <- Set{},
487 -- End of bindings inherited from Classifier
488
489 isActive <- false
490
491 ),
492
493 mr : distinct UML!Generalization foreach(e in mc.supertypes) (
494 -- Begin bindings inherited from ModelElement
495 name <- mc.name,
496 constraint <- mc.supertypes->collect(e | e.constraints),
497 namespace <- mc.container,
498 visibility <- mc.getUMLVisibility(),
499 taggedValue <-,
500 -- asArgument <-,
501 -- clientDependency <-,
502 -- implementationLocation <-,
503 -- presentation <-,
504 -- supplierDependency <-,
505 -- templateParameter <-,
506 -- stereotype <-,
507 -- End of bindings inherited from ModelElement
508
509 child <- uc,
510 parent <- e,
511 discriminator <- '',
512 powertype <- OclUndefined
513
514 ),
515
516 mo : UML!Model (
517 -- Begin bindings inherited from ModelElement
518 name <- 'Model'--,
519 constraint <- Set{},
520 namespace <- mp.container,
521 visibility <- mp.getUMLVisibility(),
522 taggedValue <-,
523 -- asArgument <-,
524 -- clientDependency <-,
525 -- implementationLocation <-,
526 -- presentation <-,
527 -- supplierDependency <-,
528 -- templateParameter <-,
529 -- stereotype <- Set{},
530 -- End of bindings inherited from ModelElement
531
532 -- Begin bindings inherited from GeneralizableElement
533 isAbstract <- mp.isAbstract,
534 isLeaf <- mp.isLeaf,
535 isRoot <- mp.isRoot,
536 generalization <- mr,
537 -- End of bindings inherited from GeneralizableElement
538
539 -- Begin bindings inherited from Namespace
540 ownedElement <- mp.contents,
541 -- End of bindings inherited from Namespace
542
543 -- Begin bindings inherited from Package
544 elementImport <- Set{}
545 -- End Of bindings inherited from Package
546
547 ),
548
549 cl : distinct UML!Stereotype foreach(e in thisModule.firstClustered) (
550 -- Begin bindings inherited from ModelElement
```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
547         name <- 'clustering',
548         constraint <- Sequence{ Set{} },
549         namespace <- mo,
550         -- visibility <- mp.getUMLVisibility(),
551         -- taggedValue <-,
552         -- asArgument <-,
553         -- clientDependency <-,
554         -- implementationLocation <-,
555         -- presentation <-,
556         -- supplierDependency <-,
557         -- templateParameter <-,
558         stereotype <- Sequence{ Set{} },
559         -- End of bindings inherited from ModelElement
560
561         -- Begin bindings inherited from GeneralizableElement
562         isAbstract <- false,
563         isLeaf <- false,
564         isRoot <- false,
565         -- generalization <-,
566         -- End of bindings inherited from GeneralizableElement
567
568         stereotypeConstraint <- Sequence{ Set{} },
569         definedTag <- Sequence{ Set{} },
570         icon <- OclUndefined,
571         baseClass <- Sequence{ Set{'Dependency'} }
572     ),
573
574     im : distinct UML!Stereotype foreach(e in thisModule.firstImport) (
575         -- Begin bindings inherited from ModelElement
576         name <- 'import',
577         constraint <- Sequence{ Set{} },
578         namespace <- mo,
579         -- visibility <- mp.getUMLVisibility(),
580         -- taggedValue <-,
581         -- asArgument <-,
582         -- clientDependency <-,
583         -- implementationLocation <-,
584         -- presentation <-,
585         -- supplierDependency <-,
586         -- templateParameter <-,
587         stereotype <- Sequence{ Set{} },
588         -- End of bindings inherited from ModelElement
589
590         -- Begin bindings inherited from GeneralizableElement
591         isAbstract <- false,
592         isLeaf <- false,
593         isRoot <- false,
594         -- generalization <-,
595         -- End of bindings inherited from GeneralizableElement
596
597         stereotypeConstraint <- Sequence{ Set{} },
598         definedTag <- Sequence{ Set{} },
599         icon <- OclUndefined,
600         baseClass <- Sequence{ Set{'Dependency'} }
601     ),
602
603     mm : distinct UML!Stereotype foreach(e in thisModule.firstMetamodel) (
604         -- Begin bindings inherited from ModelElement
605         name <- 'metamodel',
606         constraint <- Sequence{ Set{} },
607         namespace <- mo,
608         -- visibility <- mp.getUMLVisibility(),
609         -- taggedValue <-,
610         -- asArgument <-,
611         -- clientDependency <-,
612         -- implementationLocation <-,
613         -- presentation <-,
614         -- supplierDependency <-,
615         -- templateParameter <-,
```

```

616         stereotype <- Sequence{ Set{} },
617         -- End of bindings inherited from ModelElement
618
619         -- Begin bindings inherited from GeneralizableElement
620         isAbstract <- false,
621         isLeaf <- false,
622         isRoot <- false,
623     --     generalization <-,
624         -- End of bindings inherited from GeneralizableElement
625
626         stereotypeConstraint <- Sequence{ Set{} },
627         definedTag <- Sequence{ Set{} },
628         icon <- OclUndefined,
629         baseClass <- Sequence{ Set{'Package'} }
630     )
631 }
632
633 -- Rule 'OtherClass'
634 -- This rule generates a UML!Class, along with its associated
635 -- UML!Generalizations for each MOF!Class that is distinct from the reference
636 -- class computed by the firstClass helper.
637 -- Most properties of the generated Class are copied from the input MOF!Class
638 -- properties. Its generalizations correspond to the Generalization that are
639 -- generated by the rule, whereas its specializations correspond to the
640 -- UML!Classes that are generated for the MOF!Classes that have the input Class
641 -- as supertype. The powertypeRange and isActive properties, which have no
642 -- equivalent in MOF, are set to default values.
643 -- A UML!Generalization is generated fore each supertype of the input
644 -- MOF!Class. Its child corresponds to the generated UML class, whereas its
645 -- parent corresponds to the UML!Class generated for the currently iterated
646 -- supertype. Note that discriminator and powertype of the generated
647 -- Generalizations are set to default values since MOF defines no corresponding
648 -- properties.
649 rule OtherClass {
650     from
651         mc : MOF!Class (
652             mc <> thisModule.firstClass
653         )
654     to
655         uc : UML!Class (
656             -- Begin bindings inherited from ModelElement
657             name <- mc.name,
658             constraint <- mc.constraints,
659             namespace <- mc.container,
660             visibility <- mc.getUMLVisibility(),
661             -- taggedValue <-,
662             -- asArgument <-,
663             -- clientDependency <-,
664             -- implementationLocation <-,
665             -- presentation <-,
666             -- supplierDependency <-,
667             -- templateParameter <-,
668             -- stereotype<-,
669             -- End of bindings inherited from ModelElement
670
671             -- Begin bindings inherited from GeneralizableElement
672             isAbstract <- mc.isAbstract,
673             isLeaf <- mc.isLeaf,
674             isRoot <- mc.isRoot,
675             generalization <- mr,
676             -- End of bindings inherited from GeneralizableElement
677
678             -- Begin bindings inherited from Namespace
679             -- ownedElement <- mc.contents,
680             -- End of bindings inherited from Namespace
681
682             -- Begin bindings inherited from Classifier
683             feature <- mc.contents,
684             powertypeRange <- Set{}},

```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
685         -- End of bindings inherited from Classifier
686
687         isActive <- false
688     ),
689
690     mr : distinct UML!Generalization foreach(e in mc.supertypes) (
691         -- Begin bindings inherited from ModelElement
692         name <- mc.name,
693         constraint <- mc.supertypes->collect(e | e.constraints),
694         namespace <- mc.container,
695         visibility <- mc.getUMLVisibility(),
696         -- taggedValue <- ,
697         -- asArgument <- ,
698         -- clientDependency <- ,
699         -- implementationLocation <- ,
700         -- presentation <- ,
701         -- supplierDependency <- ,
702         -- templateParameter <- ,
703         -- stereotype <- ,
704         -- End of bindings inherited from ModelElement
705
706         child <- uc,
707         parent <- e,
708         discriminator <- '',
709         powertype <- OclUndefined
710     )
711 }
712
713 -- Rule 'Operation'
714 -- This rule generates a UML!Operation from a MOF!Operation.
715 -- Properties of the generated operation are copied from those of the input MOF
716 -- operation, except concurrency, isAbstract, isLeaf and isRoot, which are not
717 -- defined in MOF, and therefore set to default values.
718 rule Operation {
719     from
720         mo : MOF!Operation
721     to
722         uo : UML!Operation (
723             -- Begin bindings inherited from ModelElement
724             name <- mo.name,
725             constraint <- mo.constraints,
726             -- namespace <- mo.container,
727             visibility <- mo.getUMLVisibility(),
728             -- taggedValue <- ,
729             -- asArgument <- ,
730             -- clientDependency <- ,
731             -- implementationLocation <- ,
732             -- presentation <- ,
733             -- supplierDependency <- ,
734             -- templateParameter <- ,
735             -- stereotype <- ,
736             -- End of bindings inherited from ModelElement
737
738             -- Begin bindings inherited from Feature
739             ownerScope <- mo.getUMLScope(),
740             owner <- mo.container,
741             -- End of bindings inherited from Feature
742
743             -- Begin bindings inherited from BehavioralFeature
744             isQuery <- mo.isQuery,
745             parameter <- mo.contents,
746             -- End of bindings inherited from BehavioralFeature
747
748             concurrency <- #cck_guarded,
749             isAbstract <- false,
750             isLeaf <- false,
751             isRoot <- false
752         )
753 }
```



```
754
755 -- Rule 'Constraint'
756 -- This rule generates a UML!Constraint from a MOF!Constraint.
757 -- Properties of the generated constraint are copied from the input constraint,
758 -- except body which is set by default to the 'oclUndefined' value.
759 rule Constraint {
760     from
761         mc : MOF!Constraint
762     to
763         uc : UML!Constraint (
764             -- Begin bindings inherited from ModelElement
765             name <- mc.name,
766             constraint <- mc.constraints,
767             namespace <- mc.container,
768             visibility <-,
769             taggedValue <-,
770             asArgument <-,
771             clientDependency <-,
772             implementationLocation <-,
773             presentation <-,
774             supplierDependency <-,
775             templateParameter <-,
776             stereotype<-,
777             -- End of bindings inherited from ModelElement
778
779             constrainedElement <- mc.constrainedElements,
780             body <- OclUndefined
781         )
782 }
783
784 -- Rule 'Tag'
785 -- This rule generates a UML!TaggedValue from a MOF!Tag.
786 -- Note that the type of the generated Tag is copied from the MOF!Tag tagId
787 -- attribute. The model element the generated TaggedValue is attached to
788 -- corresponds to the first element of the elements collection of the input
789 -- MOF!Tag entity. Finally, as MOF only provides support for dataValues, the
790 -- referenceValue of the generated UML!TaggedValue element is initialized
791 -- with an empty set.
792 rule TaggedValue {
793     from
794         mt : MOF!Tag
795     to
796         ut : UML!TaggedValue (
797             -- Begin bindings inherited from ModelElement
798             name <- mt.name,
799             constraint <- mt.constraints,
800             namespace <- mt.container,
801             visibility <-,
802             taggedValue <-,
803             asArgument <-,
804             clientDependency <-,
805             implementationLocation <-,
806             presentation <-,
807             supplierDependency <-,
808             templateParameter <-,
809             stereotype<-,
810             -- End of bindings inherited from ModelElement
811
812             dataValue <- mt.values,
813             type <- mt.tagId,
814             modelElement <- mt.elements->asSequence()->first(),
815             referenceValue <- Set{}
816         )
817 }
818
819 -- Rule 'Import'
820 -- This rule generates a UML!Dependency from a MOF!Import entity.
821 -- The client of the generated Dependency corresponds to the container of the
822 -- input Import, whereas its supplier corresponds to the importedNamespace of
```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
823 -- the Import.
824 -- The namespace of the generated package corresponds to the model ('mo')
825 -- generated by the FirstClass rule, whereas, according to the value of the
826 -- isClustered attribute, its stereotype corresponds either to the clustering
827 -- ('cl') or import ('im') stereotype generated by FirstClass.
828 rule Import {
829     from
830         mi : MOF!Import
831     to
832         ud : UML!Dependency (
833             -- Begin bindings inherited from ModelElement
834             name <- mi.name,
835             constraint <- mi.constraints,
836             -- implementationLocation <-,
837             -- presentation <-,
838             -- supplierDependency <-,
839             -- templateParameter <-,
840             namespace <- thisModule.resolveTemp(thisModule.firstClass, 'mo'),
841             visibility <-,
842             -- taggedValue <-,
843             stereotype <-
844                 Set{
845                     if mi.isClustered
846                     then
847                         thisModule.resolveTemp(thisModule.firstClass,
848 'cl')
849                     else
850                         thisModule.resolveTemp(thisModule.firstClass,
851 'im')
852                     endif
853                 },
854             -- End of bindings inherited from ModelElement
855             client <- Sequence{mi.container},
856             supplier <- Sequence{mi.importedNamespace}
857         )
858     }
859 }
860
861 -- Rule 'Package'
862 -- This rule generates a UML Package with its associated Generalizations from a
863 -- MOF Package.
864 -- Most properties of the generated Package are copied from the input
865 -- MOF!Package properties. Its generalizations correspond to the Generalization
866 -- that are generated by the rule, whereas its specializations correspond to
867 -- the UML!Packages that are generated for the MOF!Packages that have the input
868 -- Package as supertype. The powertypeRange and isActive properties, which have
869 -- no equivalent in MOF, are set to default values. The namespace of the
870 -- generated package corresponds to the model ('mo') generated by the
871 -- FirstClass rule, whereas its stereotype corresponds to the metamodel ('mm')
872 -- stereotype generated by this rule.
873 -- A UML!Generalization is generated fore each supertype of the input
874 -- MOF!Package. Its child corresponds to the generated UML Package, whereas its
875 -- parent corresponds to the UML!Package generated for the currently iterated
876 -- supertype. Note that discriminator and powertype of the generated
877 -- Generalizations are set to default values since MOF defines no corresponding
878 -- properties.
879 rule Package {
880     from
881         mp : MOF!Package
882     to
883         up : UML!Package (
884             -- Begin bindings inherited from ModelElement
885             name <- mp.name,
886             constraint <- mp.constraints,
887             namespace <- thisModule.resolveTemp(thisModule.firstClass, 'mo'),
888             visibility <- mp.getUMLVisibility(),
889             -- taggedValue <-,
890             -- asArgument <-,
891             -- clientDependency <-,
```



## ATL Transformation Example

MOF to UML

Date 03/11/2005

```
892 -- implementationLocation <-,
893 -- presentation <-,
894 -- supplierDependency <-,
895 -- templateParameter <-,
896 -- stereotype <-
897     Set{thisModule.resolveTemp(thisModule.firstClass, 'mm')},
898 -- End of bindings inherited from ModelElement
899
900 -- Begin bindings inherited from GeneralizableElement
901 isAbstract <- mp.isAbstract,
902 isLeaf <- mp.isLeaf,
903 isRoot <- mp.isRoot,
904 generalization <- mr,
905 -- End of bindings inherited from GeneralizableElement
906
907 -- Begin bindings inherited from Namespace
908 -- ownedElement <- ,
909 -- End of bindings inherited from Namespace
910
911     elementImport <- Set{}
912 ),
913
914 mr : distinct UML!Generalization foreach(e in mp.supertypes) (
915     -- Begin bindings inherited from ModelElement
916     name <- mp.name,
917     constraint <- mp.supertypes->collect(e | e.constraints),
918     namespace <- mp.container,
919     visibility <- mp.getUMLVisibility(),
920 -- taggedValue <- ,
921 -- asArgument <- ,
922 -- clientDependency <- ,
923 -- implementationLocation <- ,
924 -- presentation <- ,
925 -- supplierDependency <- ,
926 -- templateParameter <- ,
927 -- stereotype <- ,
928 -- End of bindings inherited from ModelElement
929
930     child <- up,
931     parent <- e,
932     discriminator <- '',
933     powertype <- OclUndefined
934 )
935 }
```