ATL Transformation Examples

The MOF to UML
ATL transformation

- version 0.1 -

September 2005

by

ATLAS group
LINA & INRIA
Nantes
## Content

1  Introduction ................................................................................................................................................. 1

2  The MOF to UML ATL transformation.................................................................................................... 1
   2.1  Transformation overview ..................................................................................................................... 1
   2.2  Metamodels .......................................................................................................................................... 1
   2.3  Rules specification ............................................................................................................................... 1
   2.4  ATL code .............................................................................................................................................. 2
      2.4.1  Helpers ............................................................................................................................................ 2
      2.4.2  Rules ................................................................................................................................................ 2

3  References .................................................................................................................................................... 4

Appendix A  A simplified UML Core metamodel in KM3 format .............................................................. 5
Appendix B  A simplified MOF metamodel in KM3 format ........................................................................... 9
Appendix C  The MOF to UML ATL code ................................................................................................. 12
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.
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
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 Dependacy 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.
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


Appendix A  A simplified UML Core metamodel in KM3 format

```java
package Core {
    abstract class Element {
    }

    abstract class ModelElement extends Element {
        reference taggedValue[*] container : TaggedValue oppositeOf modelElement;
        reference clientDependency[*] : Dependency oppositeOf client;
        reference constraint[*] : Constraint oppositeOf constrainedElement;
        reference stereotype[*] : Stereotype;
        reference comment[*] : Comment oppositeOf annotatedElement;
        reference sourceFlow[*] : Flow oppositeOf source;
        reference targetFlow[*] : Flow oppositeOf target;
        reference templateParameter[*] ordered container : TemplateParameter oppositeOf
        template;
        reference namespace[0-1] : Namespace oppositeOf ownedElement;
        attribute name[0-1] : String;
        attribute visibility[0-1] : VisibilityKind;
        attribute isSpecification : Boolean;
    }

    abstract class GeneralizableElement extends ModelElement {
        reference generalization[*] : Generalization oppositeOf child;
        attribute isRoot : Boolean;
        attribute isLeaf : Boolean;
        attribute isAbstract : Boolean;
    }

    abstract class Namespace extends ModelElement {
        reference ownedElement[*] container : ModelElement oppositeOf namespace;
    }

    abstract class Classifier extends GeneralizableElement, Namespace {
        reference powertypeRange[*] : Generalization oppositeOf powertype;
        reference feature[*] ordered container : Feature oppositeOf owner;
    }

    class Class extends Classifier {
        attribute isActive : Boolean;
    }

    class DataType extends Classifier {
    }

    abstract class Feature extends ModelElement {
        reference owner[0-1] : Classifier oppositeOf feature;
        attribute ownerScope : ScopeKind;
    }

    abstract class StructuralFeature extends Feature {
        attribute multiplicity[0-1] : Multiplicity;
        attribute changeability[0-1] : ChangeableKind;
        attribute targetScope[0-1] : ScopeKind;
        attribute ordering[0-1] : OrderingKind;
    }

    class AssociationEnd extends ModelElement {
```
reference association : Association oppositeOf connection;
reference specification[*] : Classifier;
reference participant : Classifier;
reference qualifier[*] ordered container : Attribute oppositeOf associationEnd;
attribute isNavigable : Boolean;
attribute ordering[0-1] : OrderingKind;
attribute aggregation[0-1] : AggregationKind;
attribute targetScope[0-1] : ScopeKind;
attribute multiplicity[0-1] : Multiplicity;
attribute changeability[0-1] : ChangeableKind;

class Interface extends Classifier {
}

class Constraint extends ModelElement {
    reference constrainedElement[*] ordered container : ModelElement oppositeOf constraint;
    attribute body[0-1] : BooleanExpression;
}

abstract class Relationship extends ModelElement {
}

class Association extends GeneralizableElement, Relationship {
    reference connection[2-*] ordered container : AssociationEnd oppositeOf association;
}

abstract class Attribute extends StructuralFeature {
    reference associationEnd[0-1] : AssociationEnd oppositeOf qualifier;
    attribute initialValue[0-1] : Expression;
}

abstract class BehavioralFeature extends Feature {
    reference parameter[*] ordered container : Parameter oppositeOf behavioralFeature;
    attribute isQuery : Boolean;
}

class Operation extends BehavioralFeature {
    attribute concurrency[0-1] : CallConcurrencyKind;
    attribute isRoot : Boolean;
    attribute isLeaf : Boolean;
    attribute isAbstract : Boolean;
    attribute specification[0-1] : String;
}

class Parameter extends ModelElement {
    reference type : Classifier;
    reference behavioralFeature[0-1] : BehavioralFeature oppositeOf parameter;
    attribute defaultValue[0-1] : Expression;
    attribute kind : ParameterDirectionKind;
}

class Method extends BehavioralFeature {
    reference specification : Operation;
    attribute body : ProcedureExpression;
}

class Generalization extends Relationship {
    reference powertype[0-1] : Classifier oppositeOf powertypeRange;
    reference child : GeneralizableElement oppositeOf generalization;
    attribute discriminator[0-1] : String;
}

class AssociationClass extends Association, Class {
}
class Dependency extends Relationship {
    reference client[1-*] : ModelElement oppositeOf clientDependency;
    reference supplier[1-*] : ModelElement;
}

class Abstraction extends Dependency {
    attribute mapping[0-1] : MappingExpression;
}

abstract class PresentationElement extends Element {
    reference subject[*] : ModelElement;
}

class Usage extends Dependency {
}

class Binding extends Dependency {
    reference argument[1-*] ordered container : TemplateArgument oppositeOf
    binding;
}

class Component extends Classifier {
    reference deploymentLocation[*] : Node oppositeOf deployedComponent;
    reference residentElement[*] container : ElementResidence oppositeOf
    "container";
    reference implementation[*] : Artifact;
}

class Node extends Classifier {
    reference deployedComponent[*] : Component oppositeOf deploymentLocation;
}

class Permission extends Dependency {
}

class Comment extends ModelElement {
    reference annotatedElement[*] : ModelElement oppositeOf comment;
    attribute body : String;
}

class Flow extends Relationship {
    reference source[*] : ModelElement oppositeOf sourceFlow;
    reference target[*] : ModelElement oppositeOf targetFlow;
}

class ElementResidence {
    reference "container" : Component oppositeOf residentElement;
    reference resident : ModelElement;
    attribute visibility[0-1] : VisibilityKind;
}

class TemplateParameter {
    reference template : ModelElement oppositeOf templateParameter;
    reference parameter container : ModelElement;
    reference defaultElement[0-1] : ModelElement;
}

class Primitive extends DataType {
}

class Enumeration extends DataType {
    reference "literal"[1-*] ordered container : EnumerationLiteral oppositeOf
    "enumeration";
}

class EnumerationLiteral extends ModelElement {
    reference "enumeration" : Enumeration oppositeOf "literal";
}
```java
class Stereotype extends GeneralizableElement {
    reference stereotypeConstraint[*] container : Constraint;
    reference definedTag[*] container : TagDefinition oppositeOf owner;
    attribute icon[0-1] : String;
    attribute baseClass[1-*] : String;
}

class TagDefinition extends ModelElement {
    reference owner[0-1] : Stereotype oppositeOf definedTag;
    attribute tagType[0-1] : String;
    attribute multiplicity[0-1] : Multiplicity;
}

class TaggedValue extends ModelElement {
    reference type: TagDefinition;
    reference referenceValue[*] : ModelElement;
    attribute modelElement : ModelElement oppositeOf taggedValue;
    attribute dataValue[*] : String;
}

class ProgrammingLanguageDataType extends DataType {
    attribute expression : TypeExpression;
}

class Artifact extends Classifier {
}

class TemplateArgument {
    reference binding : Binding oppositeOf argument;
    reference modelElement : ModelElement;
}
```
Appendix B  A simplified MOF metamodel in KM3 format

```java
package Model {

    abstract class ModelElement {
        -- derived
        reference requiredElements[*] : ModelElement;
        reference constraints[*] : Constraint oppositeOf constrainedElements;
        reference "container"[0..1] : Namespace oppositeOf contents;
        attribute name : String;
        -- derived
        attribute qualifiedName[1..*] ordered : String;
        attribute annotation : String;
        operation findRequiredElements(kinds : String, recursive : Boolean) : Boolean;
        operation isRequiredBecause(otherElement : ModelElement, reason : String) : Boolean;
        operation isFrozen() : Boolean;
        operation isVisible(otherElement : ModelElement) : Boolean;
    }

    enumeration VisibilityKind {
        literal public_vis;
        literal protected_vis;
        literal private_vis;
    }

    abstract class Namespace extends ModelElement {
        reference contents[*] ordered container : ModelElement oppositeOf "container";
        operation lookupElement(name : String) : ModelElement;
        operation resolveQualifiedName(qualifiedName : String) : ModelElement;
        operation findElementsByType(ofType : Class, includeSubtypes : Boolean) : ModelElement;
        operation nameIsValid(proposedName : String) : Boolean;
    }

    abstract class GeneralizableElement extends Namespace {
        reference supertypes[*] ordered : GeneralizableElement;
        attribute isRoot : Boolean;
        attribute isLeaf : Boolean;
        attribute isAbstract : Boolean;
        attribute visibility : VisibilityKind;
        operation allSupertypes() : GeneralizableElement;
        operation lookupElementExtended(name : String) : ModelElement;
        operation findElementsByTypeExtended(ofType : Class, includeSubtypes : Boolean) : ModelElement;
    }

    abstract class TypedElement extends ModelElement {
        reference type : Classifier;
    }

    abstract class Classifier extends GeneralizableElement {
    }

    class Class extends Classifier {
        attribute isSingleton : Boolean;
    }

    class MultiplicityType {
```
abstract class DataType extends Classifier {
(attribute lower : Integer;
attribute upper : Integer;
attribute isOrdered : Boolean;
attribute isUnique : Boolean;
)

abstract class PrimitiveType extends DataType {

}
class EnumerationType extends DataType {
    attribute labels[1-*] ordered : String;
}
class CollectionType extends DataType, TypedElement {
    attribute multiplicity : MultiplicityType;
}
class StructureType extends DataType {

}
class StructureField extends TypedElement {

}
class AliasType extends DataType, TypedElement {

}
enumeration ScopeKind {
    literal instance_level;
    literal classifier_level;
}
abstract class Feature extends ModelElement {
    attribute scope : ScopeKind;
    attribute visibility : VisibilityKind;
}
abstract class StructuralFeature extends Feature, TypedElement {
    attribute multiplicity : MultiplicityType;
    attribute isChangeable : Boolean;
}
class Attribute extends StructuralFeature {
    attribute isDerived : Boolean;
}
class Reference extends StructuralFeature {
    reference referencedEnd : AssociationEnd;
    -- derived
    reference exposedEnd : AssociationEnd;
}
abstract class BehavioralFeature extends Feature, Namespace {
}
class Operation extends BehavioralFeature {
    reference exceptions[*] ordered : Exception;
    attribute isQuery : Boolean;
}
class Exception extends BehavioralFeature {

}
class Association extends Classifier {
    attribute isDerived : Boolean;
}

e数erentiation AggregationKind {
    literal none;
    literal shared;
    literal composite;
}

class AssociationEnd extends TypedElement {
    attribute isNavigable : Boolean;
    attribute aggregation : AggregationKind;
    attribute multiplicity : MultiplicityType;
    attribute isChangeable : Boolean;
    operation otherEnd() : AssociationEnd;
}

class Package extends GeneralizableElement {
}

class Import extends ModelElement {
    reference importedNamespace : Namespace;
    attribute visibility : VisibilityKind;
    attribute isClustered : Boolean;
}

e数erentiation DirectionKind {
    literal in_dir;
    literal out_dir;
    literal inout_dir;
    literal return_dir;
}

class Parameter extends TypedElement {
    attribute direction : DirectionKind;
    attribute multiplicity : MultiplicityType;
}

class Constraint extends ModelElement {
    reference constrainedElements[1-*] : ModelElement oppositeOf constraints;
    attribute expression : String;
    attribute language : String;
    attribute evaluationPolicy : EvaluationKind;
}

e数erentiation EvaluationKind {
    literal immediate;
    literal deferred;
}

class Constant extends TypedElement {
    attribute value : String;
}

class Tag extends ModelElement {
    reference elements[1-*] : String;
    attribute tagId : String;
    attribute values[*] ordered : String;
}
Appendix C The MOF to UML ATL code

```plaintext
module MOF2UML;
create OUT : UML from IN : MOF;
uses strings;

-- HELPERS ---------------------------------------- ----------------------------
-- This helper returns a MOF!Class that is considered as the reference Class
-- for the generation of unique target elements: the model and the possible
-- stereotypes.
-- CONTEXT: thisModule
-- RETURN: MOF!Class
helper def: firstClass : MOF!Class =
    MOF!Class.allInstancesFrom('IN') ->asSequence() ->first();

-- This helper returns a clustered MOF!Import that is considered as the
-- reference Import for the generation of the 'clustered' stereotype.
-- CONTEXT: thisModule
-- RETURN: Sequence(MOF!Import)
helper def: firstClustered : Sequence(MOF!Import) =
    Sequence{
        MOF!Import.allInstancesFrom('IN')
        ->select(e | e.isClustered)
        ->asSequence() ->first()
    };

-- This helper returns an unclustered MOF!Import that is considered as the
-- reference Import for the generation of the 'import' stereotype.
-- CONTEXT: thisModule
-- RETURN: Set(MOF!Import)
helper def: firstImport : Sequence(MOF!Import) =
    Sequence{
        MOF!Import.allInstancesFrom('IN')
        ->select(e | not e.isClustered)
        ->asSequence() ->first()
    };

-- This helper returns a MOF!Package that is considered as the reference
-- Package for the generation of the 'import' stereotype.
-- CONTEXT: thisModule
-- RETURN: Set(MOF!Package)
helper def: firstMetamodel : Sequence(MOF!Package) =
    Sequence{
        MOF!Package.allInstancesFrom('IN') ->asSequence() ->first()
    };

-- This helper returns the UML!OrderingKind that corresponds to the
-- non undefined MOF!ScopeKind of the contextual MOF!ModelElement.
-- The helper returns the '#ordered' or '#unordered' value depending on the
-- value of the MOF 'isOrdered' boolean attribute.
-- WARNING: the contextual MOF!ModelElement must be of either a
-- MOF!StructuralFeature or a MOF!AssociationEnd element.
-- CONTEXT: MOF!ModelElement
-- RETURN: UML!OrderingKind
helper context MOF!ModelElement def: getOrdering() : UML!OrderingKind =
    if self.multiplicity.isOrdered
    then
```
#ok_ordered

#ok_unordered

endif;

-- This helper returns the UML!OrderingKind that corresponds to the
-- MOF!ScopeKind of the contextual MOF!ModelElement.
-- If the multiplicity attribute of the contextual ModelElement, or its
-- isOrdered attribute is undefined, the helper returns 'ok_unordered'.
-- Otherwise, the helper returns the value computed by getOrdering().
-- WARNING: the contextual MOF!ModelElement must be of either a
-- MOF!StructuralFeature or a MOF!AssociationEnd element.

helper context MOF!ModelElement def: getUMLOrdering() : UML!OrderingKind =
if self.multiplicity.oclIsUndefined() then
#ok_unordered
else
if self.multiplicity.isOrdered.oclIsUndefined() then
#ok_unordered
else
self.getOrdering()
endif
endif;

-- This helper returns the UML!Visibility that corresponds to the
-- non undefined MOF!Visibility of the contextual MOF!ModelElement.
-- WARNING: the contextual MOF!ModelElement must be of either a MOF!Feature, a
-- MOF!Import or a MOF!GeneralizableElement entity.

helper context MOF!ModelElement def: getVisibility() : UML!Visibility =
let v : MOF!Visibility = self.visibility in
if v = #public_vis then
#vk_public
else
if v = #protected_vis then
#vk_protected
else
if v = #private_vis then
#vk_protected
else -- default
#vk_public
endif
endif;

-- This helper returns the UML!Visibility that corresponds to the
-- MOF!Visibility of the contextual MOF!ModelElement.
-- If the visibility of the contextual ModelElement is undefined, the helper
-- returns 'vk_public', otherwise, it returns the value provided by
-- getVisibility().
-- WARNING: the contextual MOF!ModelElement must be of either a MOF!Feature, a
-- MOF!Import or a MOF!GeneralizableElement entity.

helper context MOF!ModelElement def: getUMLVisibility() : UML!Visibility =
if self.visibility.oclIsUndefined()
then
    #vk_public
else
    self.getVisibility()
endif;

-- This helper returns the UML!ChangeableKind that corresponds to the
-- non-defined MOF!ChangeableKind of the contextual MOF!ModelElement.
-- The helper returns the '#ck_changeable' or '#ck_frozen' value depending on
-- the value of the MOF 'isChangeable' boolean attribute.
-- WARNING: the contextual MOF!ModelElement must be of either a
-- MOF!StructuralFeature or a MOF!AssociationEnd element.

helper context MOF!ModelElement def: getChangeability() : UML!ChangeableKind =
    if self.isChangeable
        then
            #ck_changeable
        else
            #ck_frozen
        endif;

-- This helper returns the UML!ChangeableKind that corresponds to the
-- MOF!ChangeableKind of the contextual MOF!ModelElement.
-- If changeability of the contextual MOF!ModelElement is undefined, the helper
-- returns the '#ck_changeable' default value. Otherwise, it returns the value
-- computes by the getChangeability helper.
-- WARNING: the contextual MOF!ModelElement must be of either a
-- MOF!StructuralFeature or a MOF!AssociationEnd element.

helper context MOF!ModelElement def: getChangeability() : UML!ChangeableKind =
    if not self.isChangeable.oclIsUndefined()
        then
            self.getChangeability()
        else
            #ck_changeable
        endif;

-- This helper returns the UML!ScopeKind that corresponds to the MOF!ScopeKind
-- of the contextual MOF!Feature.

helper context MOF!Feature def: getUMLScope() : UML!ScopeKind =
    if self.scope = #instance_level
        then
            #sk_instance
        else
            #sk_classifier
        endif;

-- RULES
-- Rule 'Association'
-- This rule generates a UML!Association, along with its associated
-- Most properties of the generated association are copied from the input MOF
-- association properties. Its generalizations correspond to the Generalization
-- that are generated by the rule, whereas its specializations correspond to
-- the UML!Associations that are generated for the MOF!Associations that have
-- the input association as supertype.
-- A UML!Generalization is generated for each supertype of the input
-- MOF!Association. Its child corresponds to the generated UML association,
-- whereas its parent corresponds to the UML!Association generated for the
-- currently iterated supertype. Note that discriminator and powertype of the
-- generated Generalizations are set to default values since MOF defines no
-- corresponding properties.

rule Association {
    from
    ma : MOF!Association
to
    ua : UML!Association {
        -- Begin bindings inherited from ModelElement
        name <- ma.name,
        constraint <- ma.constraints,
        namespace <- ma.container,
        visibility <- ma.getUMLVisibility(),
        taggedValue <-,
        asArgument <-,
        clientDependency <-,
        implementationLocation <-,
        presentation <-,
        supplierDependency <-,
        templateParameter <-,
        stereotype<-
        -- End of bindings inherited from ModelElement

        -- Begin bindings inherited from GeneralizableElement
        isAbstract <- ma.isAbstract,
        isLeaf <- ma.isLeaf,
        isRoot <- ma.isRoot,
        generalization <- mr
        -- End of bindings inherited from GeneralizableElement

        child <- ua,
        parent <- e,
        discriminator <- '',
        powertype <- OclUndefined
    }
}

-- Rule 'AssociationEnd'
-- This rule generates a UML!AssociationEnd, along with its UML!Multiplicity,
-- and the MultiplicityRange of this last, from a MOF!AssociationEnd.
-- Most properties of the generated AssociationEnd are copied from those of
-- the input MOF AssociationEnd. Its multiplicity reference points to the
-- Multiplicity entity generated by the rule. The targetScope, qualifier and
-- specification properties are set to default values (MOF does not define
-- corresponding properties).

rule AssociationEnd {
    from
    ma : MOF!AssociationEnd
to
    ea : UML!AssociationEnd {
        -- Begin bindings inherited from ModelElement
        name <- ma.name,
        constraint <- ma.constraints->collect(e | e.constraints),
        namespace <- ma.container,
        visibility <- ma.getUMLVisibility(),
        taggedValue <-,
        asArgument <-,
        clientDependency <-,
        implementationLocation <-,
        presentation <-,
        supplierDependency <-,
        templateParameter <-,
        stereotype<-
        -- End of bindings inherited from ModelElement

        child <- ea,
        parent <- e,
        multiplicity <- OclUndefined
    }
}
ma : MOF!AssociationEnd
to
ua : UML!AssociationEnd

-- Begin bindings inherited from ModelElement
name <- ma.name,
constraint <- ma.constraints,
namespace <- ma.container,
visibility <-,
taggedValue <-,
asArgument <-,
clientDependency <-,
implementationLocation <-,
presentation <-,
supplierDependency <-,
templateParameter <-,
stereotype<-,

-- End of bindings inherited from ModelElement

association <- ma.container,
aggregation <-
  if ma.aggregation = #none
  then
    #ak_none
  else
    if ma.aggregation = #shared
      then
        #ak_aggregate
    else
      -- ma.aggregation = #composite
        #ak_composite
  endif,
changeability <- ma.getUMLChangeability(),
ordering <- ma.getUMLOrdering(),
isNavigable <- ma.isNavigable,
multiplicity <- um,
targetScope <- #sk_instance,
qualifier <- Sequence{},
specification <- Set{},
participant <- ma.type,

},

um : UML!Multiplicity {
  range <- Set{ur}
},

ur : UML!MultiplicityRange {
  lower <- ma.multiplicity.lower,
  upper <- ma.multiplicity.upper,
  multiplicity <- um
}

-- Rule 'Parameter'
-- This rule generates a UML!Parameter from a MOF!Parameter.
-- Properties of the generated Parameter are copied from those of the input
-- Parameter, except the UML defaultValue attribute which has no MOF
-- equivalent. It is therefore set to 'oclUndefined'.
rule Parameter {
  from
  mp : MOF!Parameter
to
  up : UML!Parameter {
    -- Begin bindings inherited from ModelElement
    name <- mp.name,
    constraint <- mp.constraints,
    namespace <- mp.container,
    visibility <-,
taggedValue <-,
    implementationLocation <-,
ATL Transformation Example

MOF to UML

Date 03/11/2005

--- Rule 'Attribute'
--- This rule generates a UML!Attribute, along with its UML!Multiplicity, and
--- the UML!MultiplicityRange of this last, from a MOF!Attribute.
--- Most properties of the generated Attribute are copied from those of the
--- input MOF Attribute. Its multiplicity reference points to the Multiplicity
--- entity generated by the rule. The targetScope and initialValue properties
--- are set to default values (MOF does not define corresponding properties):
--- 'sk_instance' for targetScope and 'oclUndefined' for initialValue.
--- The range of the generated Multiplicity element is computed from the
--- multiplicity attribute of the input MOF!Attribute.

rule Attribute {
    from ma : MOF!Attribute
    to ua : UML!Attribute {
        -- Begin bindings inherited from ModelElement
        name <- ma.name,
        constraint <- ma.constraints,
        namespace <- ma.container,
        visibility <- ma.getUMLVisibility(),
        taggedValue <-,
        asArgument <-,
        clientDependency <-,
        implementationLocation <-,
        presentation <-,
        supplierDependency <-,
        stereotype<-
        -- End of bindings inherited from ModelElement
        changeability <- ma.getUMLChangeability(),
        ownerScope <- ma.getUMLScope(),
        owner <- ma.container,
        -- End of bindings inherited from Feature
        ordering <- ma.getUMLOrdering(),
        multiplicity <- um,
    }

    kind <-
    if mp.direction = #in_dir
        then #pdk_in
    else
        if mp.direction = #inout_dir
            then #pdk_inout
        else
            if mp.direction = #out_dir
                then #pdk_out
            else
                if mp.direction = #return_dir
                    then #pdk_return
                endif
            endif
        endif
    endif

    type <- mp.type,
    defaultValue <- OclUndefined
}

type <- ma.type,
targetScope <- #sk_instance,
-- End of bindings inherited from StructuralFeature
initialValue <- OclUndefined
},

um : UML!Multiplicity {
  range <- Set{ur}
},

ur : UML!MultiplicityRange {
  lower <- ma.multiplicity.lower,
  upper <- ma.multiplicity.upper,
  multiplicity <- um
}

-- Rule 'Class'
-- This rule generates a UML!Class, along with its associated
-- UML!Generalizations, the UML!Model, and the 'metamodel', 'import', and
-- 'clustering' UML!Stereotype from the reference MOF!Class provided by the
-- firstClass helper.
-- Most properties of the generated Class are copied from the input MOF!Class
-- properties. Its generalizations correspond to the Generalization that are
-- generated by the rule, whereas its specializations correspond to the
-- UML!Classes that are generated for the MOF!Classes that have the input Class
-- as supertype. The powertypeRange and isActive properties, which have no
-- equivalent in MOF, are set to default values.
-- A UML!Generalization is generated for each supertype of the input
-- MOF!Class. Its child corresponds to the generated UML class, whereas its
-- parent corresponds to the UML!Class generated for the currently iterated
-- supertype. Note that discriminator and powertype of the generated
-- Generalizations are set to default values since MOF defines no corresponding
-- properties.
-- 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 ('mo') generated by the rule.
rule FirstClass {
  from
  mc : MOF!Class {
    mc = thisModule.firstClass
  }
  to
  uc : UML!Class {
    -- Begin bindings inherited from ModelElement
    name <- mc.name,
    constraint <- mc.constraints,
    namespace <- mc.container,
    visibility <- mc.getUMLVisibility(),
    -- taggedValue <-,
    -- asArgument <-,
    -- clientDependency <-,
    -- implementationLocation <-,
    -- presentation <-,
    -- supplierDependency <-,
    -- templateParameter <-,
    -- stereotype-,
    -- End of bindings inherited from ModelElement
    isAbstract <- mc.isAbstract,
    isLeaf <- mc.isLeaf,
    isRoot <- mc.isRoot,
    generalization <- mr,
    -- End of bindings inherited from GeneralizableElement
   泛性 <- ma.pattern泛性,
    targetScope <- #sk_instance,
    -- End of bindings inherited from StructuralFeature
    初始化值 <- OclUndefined
  }
}
478  -- Begin bindings inherited from Namespace
479  -- ownedElement <- mc.contents,
480  -- End of bindings inherited from Namespace
481
482  -- Begin bindings inherited from Classifier
483  feature <- mc.contents,
484  powertypeRange <- Set{},
485  -- End of bindings inherited from Classifier
486
487  isActive <- false
488
489   
490  mr : distinct UML!Generalization foreach(e in mc.supertypes) (  
491   -- Begin bindings inherited from ModelElement
492   name <- mc.name,
493   constraint <- mc.supertypes->collect(e | e.constraints),
494   namespace <- mc.container,
495   visibility <- mc.getUMLVisibility(),
496   -- taggedValue <-,
497   -- asArgument <-,
498   -- clientDependency <-,
499   -- implementationLocation <-,
500   -- presentation <-,
501   -- supplierDependency <-,
502   -- templateParameter <-,
503   -- stereotype<-,
504   -- End of bindings inherited from ModelElement
505
506  child <- uc,
507  parent <- e,
508  discriminator <= '',
509  powertype <- OclUndefined
510
511 )
512
513  mo : UML!Model {
514   -- Begin bindings inherited from ModelElement
515   name <- "Model"--,
516   constraint <- Set{},
517   namespace <- mp.container,
518   visibility <- mp.getUMLVisibility(),
519   taggedValue <-,
520   -- asArgument <-,
521   -- implementationLocation <-,
522   -- presentation <-,
523   -- supplierDependency <-,
524   -- stereotype <- Set{},
525   -- End of bindings inherited from ModelElement
526
527      
528  mo : UML!Model {
529   -- Begin bindings inherited from GeneralizableElement
530   isAbstract <- mp.isAbstract,
531   isLeaf <- mp.isLeaf,
532   isRoot <- mp.isRoot,
533   -- generalization <- mr,
534   -- End of bindings inherited from GeneralizableElement
535
536  -- Begin bindings inherited from Namespace
537  ownedElement <- mp.contents,
538  -- End of bindings inherited from Namespace
539
540  -- Begin bindings inherited from Package
541  elementImport <- Set{}
542  -- End Of bindings inherited from Package
543
544 )
545
546  cl : distinct UML!Stereotype foreach(e in thisModule.firstClustered) (  
547   -- Begin bindings inherited from ModelElement
548   --
name <- 'clustering',
constraint <- Sequence{ Set{} },
namespace <- mo,
visibility <- mp.getUMLVisibility(),
taggedValue <-,
asArgument <-,
clientDependency <-,
implementationLocation <-,
presentation <-,
supplierDependency <-,
templateParameter <-,

stereotype <- Sequence{ Set{} },
-- End of bindings inherited from ModelElement

-- Begin bindings inherited from GeneralizableElement
isAbstract <- false,
isLeaf <- false,
isRoot <- false,
generalization <-,
-- End of bindings inherited from GeneralizableElement

stereotypeConstraint <- Sequence{ Set{} },
definedTag <- Sequence{ Set{} },
icon <- OclUndefined,
baseClass <- Sequence{ Set('Dependency') }

im : distinct UML!Stereotype foreach(e in thisModule.firstImport) {  
  -- Begin bindings inherited from ModelElement
  name <- 'import',
  constraint <- Sequence{ Set{} },
namespace <- mo,
visibility <- mp.getUMLVisibility(),
taggedValue <-,
asArgument <-,
clientDependency <-,
implementationLocation <-,
presentation <-,
supplierDependency <-,
templateParameter <-,

stereotype <- Sequence{ Set{} },
-- End of bindings inherited from ModelElement

-- Begin bindings inherited from GeneralizableElement
isAbstract <- false,
isLeaf <- false,
isRoot <- false,
generalization <-,
-- End of bindings inherited from GeneralizableElement

stereotypeConstraint <- Sequence{ Set{} },
definedTag <- Sequence{ Set{} },
icon <- OclUndefined,
baseClass <- Sequence{ Set('Dependency') }

}

mm : distinct UML!Stereotype foreach(e in thisModule.firstMetamodel) {  
  -- Begin bindings inherited from ModelElement
  name <- 'metamodel',
  constraint <- Sequence{ Set{} },
namespace <- mo,
visibility <- mp.getUMLVisibility(),
taggedValue <-,
asArgument <-,
clientDependency <-,
implementationLocation <-,
presentation <-,
supplierDependency <-,
templateParameter <-,
stereotype <- Sequence{ Set{} },

-- End of bindings inherited from ModelElement

-- Begin bindings inherited from GeneralizableElement
isAbstract <- false,
isLeaf <- false,
isRoot <- false,
generalization <-

-- End of bindings inherited from GeneralizableElement

stereotypeConstraint <- Sequence{ Set{} },
definedTag <- Sequence{ Set{} },
icon <- OclUndefined,
baseClass <- Sequence{ 'Package' } }

-- Rule 'OtherClass'
-- This rule generates a UML!Class, along with its associated
-- UML!Generalizations for each MOF!Class that is distinct from the reference
-- class computed by the firstClass helper.
-- Most properties of the generated Class are copied from the input MOF!Class
-- properties. Its generalizations correspond to the Generalization that are
-- generated by the rule, whereas its specializations correspond to the
-- UML!Classes that are generated for the MOF!Classes that have the input Class
-- as supertype. The powertypeRange and isActive properties, which have no
-- equivalent in MOF, are set to default values.
-- A UML!Generalization is generated for each supertype of the input
-- MOF!Class. Its child corresponds to the generated UML class, whereas its
-- parent corresponds to the UML!Class generated for the currently iterated
-- supertype. Note that discriminator and powertype of the generated
-- Generalizations are set to default values since MOF defines no corresponding
-- properties.

rule OtherClass {  
from
  mc : MOF!Class {
    mc <> thisModule.firstClass
  }

to
  uc : UML!Class {
    -- Begin bindings inherited from ModelElement
    name <- mc.name,
    constraint <- mc.constraints,
    namespace <- mc.container,
    visibility <- mc.getUMLVisibility(),
    taggedValue <-,
    -- asArgument <-,
    -- clientDependency <-,
    -- implementationLocation <-,
    -- presentation <-,
    -- supplierDependency <-,
    -- templateParameter <-,
    -- stereotype<-,
    -- End of bindings inherited from ModelElement
    -- Begin bindings inherited from GeneralizableElement
    isAbstract <- mc.isAbstract,
    isLeaf <- mc.isLeaf,
    isRoot <- mc.isRoot,
    generalization <- mc,
    -- End of bindings inherited from GeneralizableElement
    -- Begin bindings inherited from Namespace
    ownedElement <- mc.contents,
    -- End of bindings inherited from Namespace
    -- Begin bindings inherited from Classifier
    feature <- mc.contents,
    powertypeRange <- Set{},
}
-- End of bindings inherited from Classifier

isActive <- false

-- Begin bindings inherited from ModelElement

name <- mc.name,
constraint <- mc.supertypes->collect(e | e.constraints),
namespace <- mc.container,
visibility <- mc.getUMLVisibility(),
taggedValue <-,
asArgument <-,
clientDependency <-,
implementationLocation <-,
presentation <-,
supplierDependency <-,
templateParameter <-,
stereotype<-,
-- End of bindings inherited from ModelElement

cap <- uc,
child <- e,
parent <- e,

-- Begin bindings inherited from ModelElement

discriminator <- '',
powertype <- OclUndefined

-- Begin bindings inherited from Feature

ownerScope <- mc.getUMLScope(),

-- End of bindings inherited from Feature

-- Begin bindings inherited from BehavioralFeature

isQuery <- mo.isQuery,
parameter <- mo.contents,

-- End of bindings inherited from BehavioralFeature

concurrency <- #cck_guarded,
isAbstract <- false,
isLeaf <- false,
isRoot <- false
-- Rule 'Constraint'
-- This rule generates a UML!Constraint from a MOF!Constraint.
-- Properties of the generated constraint are copied from the input constraint,
-- except body which is set by default to the 'oclUndefined' value.
rule Constraint {
    from mc : MOF!Constraint
to uc : UML!Constraint {
    -- Begin bindings inherited from ModelElement
    name <- mc.name,
    constraint <- mc.constraints,
    namespace <- mc.container,
    -- visibility <-,
    -- taggedValue <-,
    asArgument <-,
    -- clientDependency <-,
    -- implementationLocation <-,
    -- presentation <-,
    -- supplierDependency <-,
    -- templateParameter <-,
    -- stereotype<-
    -- End of bindings inherited from ModelElement
    constrainedElement <- mc.constrainedElements,
    body <- OclUndefined
}
}

-- Rule 'Tag'
-- This rule generates a UML!TaggedValue from a MOF!Tag.
-- Note that the type of the generated Tag is copied from the MOF!Tag tagId
-- attribute. The model element the generated TaggedValue is attached to
-- corresponds to the first element of the elements collection of the input
-- MOF!Tag entity. Finally, as MOF only provides support for dataValues, the
-- referenceValue of the generated UML!TaggedValue element is initialized
-- with an empty set.
rule TaggedValue {
    from mt : MOF!Tag
to ut : UML!TaggedValue {
    -- Begin bindings inherited from ModelElement
    name <- mt.name,
    constraint <- mt.constraints,
    namespace <- mt.container,
    -- visibility <-,
    -- taggedValue <-,
    asArgument <-,
    -- clientDependency <-,
    -- implementationLocation <-,
    -- presentation <-,
    -- supplierDependency <-,
    -- templateParameter <-,
    -- stereotype<-
    -- End of bindings inherited from ModelElement
    dataValue <- mt.values,
    type <- mt.tagId,
    modelElement <- mt.elements->asSequence()->first(),
    referenceValue <- Set()
}
}

-- Rule 'Import'
-- This rule generates a UML!Dependency from a MOF!Import entity.
-- The client of the generated Dependency corresponds to the container of the
-- input Import, whereas its supplier corresponds to the importedNamespace of
-- the Import.
-- The namespace of the generated package corresponds to the model ('mo')
-- generated by the FirstClass rule, whereas, according to the value of the
-- isClustered attribute, its stereotype corresponds either to the clustering
-- ('cl') or import ('im') stereotype generated by FirstClass.
rule Import {
  from mi : MOF!Import
to ud : UML!Dependency {
    -- Begin bindings inherited from ModelElement
    name <- mi.name,
    constraint <- mi.constraints,
    implementationLocation <-,
    presentation <-,
    supplierDependency <-,
    templateParameter <-,
    namespace <- thisModule.resolveTemp(thisModule.firstClass, 'mo'),
    visibility <-,
    taggedValue <-,
    stereotype <-
    Set{
      if mi.isClustered
      then 'cl'
      else 'im'
    }
    -- End of bindings inherited from ModelElement
    client <- Sequence(mi.container),
    supplier <- Sequence(mi.importedNamespace)
}

-- Rule 'Package'
-- This rule generates a UML Package with its associated Generalizations from a
-- MOF Package. Its child corresponds to the generated MOF!Package that have the input
-- Package as supertype. The powertypeRange and isActive properties, which have
-- no equivalent in MOF, are set to default values. The namespace of the
-- generated package corresponds to the model ('mo') generated by the
-- FirstClass rule, whereas its stereotype corresponds to the metamodel ('mm')
-- stereotype generated by this rule.
-- A UML!Generalization is generated for each supertype of the input
-- MOF!Package. Its child corresponds to the generated UML Package, whereas its
-- parent corresponds to the UML!Package generated for the currently iterated
-- supertype. Note that discriminator and powertype of the generated
-- Generalizations are set to default values since MOF defines no corresponding
rule Package {
  from mp : MOF!Package
to up : UML!Package {
    -- Begin bindings inherited from ModelElement
    name <- mp.name,
    constraint <- mp.constraints,
    namespace <- thisModule.resolveTemp(thisModule.firstClass, 'mo'),
    visibility <- mp.getUMLVisibility(),
    taggedValue <-,
    asArgument <-,
    clientDependency <-,
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 )