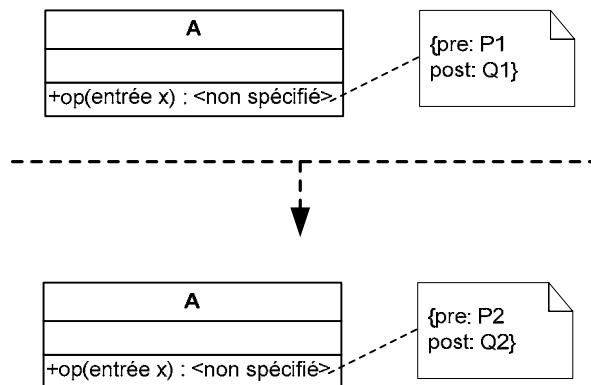
	ATL Transformation Catalogue of Model Transformations	Author Eric Simon eric.simon3 <at> gmail.com
	Documentation	Aug 7th 2006

1.	ATL TRANSFORMATION EXAMPLE: REMOVAL OF ASSOCIATION CLASSES	1
2.	ATL TRANSFORMATION OVERVIEW.....	1
2.1.	DESCRIPTION	1
2.2.	PURPOSE	1
2.3.	RULES SPECIFICATION	2
2.4.	ATL CODE.....	4
3.	REFERENCES	8

1. ATL Transformation Example: Assertion Modification

This example is extract from [Catalogue of Model Transformations](#) by K. Lano.
Section 1.2: Removal of many-many associations, page 2.




2. ATL Transformation overview

2.1. Description

“An operation precondition can be weakened (so that it is able to be applied in more situations without error) and/or its postcondition strengthened (so that its effect is determined more precisely). Both potentially move the method closer to implementation.”

2.2. Purpose

The purpose of this transformation is to weakening preconditions or strengthening postconditions.

	ATL Transformation Catalogue of Model Transformations	Author Eric Simon eric.simon3 <at> gmail.com
	Documentation	Aug 7th 2006

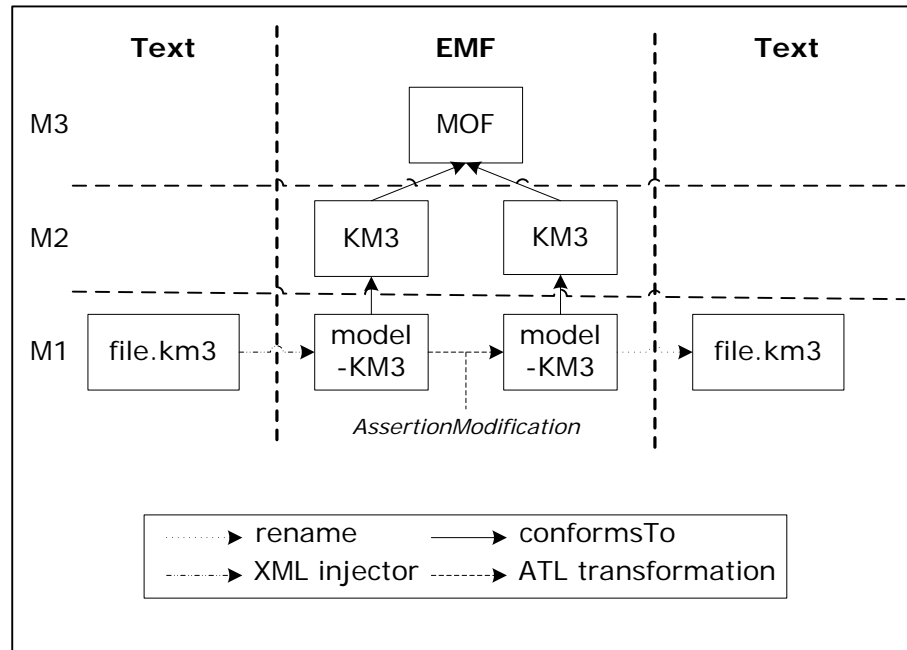



Fig 1. Overview of the transformation

2.3. Rules specification


The transformation has the same metamodel for the source and the target: UML2. However, we choose two different names: UML2 and UML2Target, indeed there is a confusion with the rule `ocl:UML2!<nameClass>->allInstances()` which returns all the class appertaining to the source **and** the target.

It is necessary for that, to have two entries: standard entry (model) and a "library" (metadata) which allows replacing the oldest assertion by a new. However, we do not verify if the replaced assertion is conforming.

- Rule [Model](#): for each *Model* element, another *Model* element is created with the following elements:
 - the attribute *name* is the same,
 - the reference *ownedMember* is the same for the classes, but for the associations only those, which are not association classes.
- Rule [DataType](#): for each *DataType* element, another *DataType* element is created with the following element:
 - the attribute *name* is the same.
- Rule [LiteralNull](#): for each *LiteralNull* element, another *LiteralNull* element is created;
- Rule [LiteralInteger](#): for each *LiteralInteger* element, another *LiteralInteger* element is created with the following element:
 - the attribute *value* is the same.

	ATL Transformation Catalogue of Model Transformations	Author Eric Simon eric.simon3 <at> gmail.com
	Documentation	Aug 7th 2006

- Rule [LiteralUnlimitedNatural](#): for each *LiteralUnlimitedNatural* element, another *LiteralUnlimitedNatural* element is created with the following element:
 - the attribute *value* is the same.
- Rule [LiteralString](#): for each *LiteralUnlimitedNatural* element
 - if a new rule does not exists for the assertion
 - another *LiteralString* element is created with the following element:
 - the attribute *value* is the same.
- Rule [Association](#): for each *Association* element, another *Association* element with the following elements:
 - the attribute *name* is the same,
 - the reference *memberEnd* is the same one as source.
- Rule [Property](#): for each *Property* element, another *Property* element is created with the following elements:
 - the attribute *name* is the same,
 - the reference *type* is the same one as the source.
- Rule [Constraint](#): for each *Constraint* element, another *Constraint* element is created with the following elements:
 - the attribute *name* is the same,
 - the reference *namespace* is
 - if it exists in the “library” a new rule for the constraint
 - the *literalString* is replaced by the new rule thanks to the [newRule](#) rule.
 - if it does not exist in the “library” a new rule for the constraint
 - the *namespace* is the same one as the source.
- Rule [Class](#): for each *Class* element, another *Class* element is created with the following elements:
 - the attributes *name* and *isActive* are the same,
 - the references *ownedOperation*, *nestedClassifier*, *ownedReception* and *ownedAttribute* are the same one as the source.
- Rule [Operation](#): for each *Operation* element, another *Operation* element is created with the following elements:
 - the attribute *name* is the same,
 - the references *class_*, *ownedRule* and *ownedParameter* are the same one as the source.
- Rule [Parameter](#): for each *Parameter* element, another *Parameter* element is created with the following elements:
 - the attribute *name* is the same,
 - the references *operation* and *type* are the same one as the source.

	ATL Transformation Catalogue of Model Transformations	Author Eric Simon eric.simon3 <at> gmail.com
	Documentation	Aug 7th 2006

2.4. ATL Code

```

module AssertionModification; -- Module Template
create OUT : UML2Target from IN : UML2, Lib : XML;

-- @comment this helper returns the first attribute named "name"
helper context XML!Element def : getAttr(name : String) : XML!Attribute =
  self.children->
    select (c|c.oclIsTypeOf(XML!Attribute))->
      select(c|c.name = name)->first();

-- @comment this helper returns the value of the first attribute named "name"
helper context XML!Element def : getAttrVal(name : String) : String =
  self.getAttr(name).value;

-- @comment this helper returns the set of childs for a given type
helper context XML!Element
def : getChildren(type : OclType, name : String) : Sequence(XML!Node) =
  self.children->
    select(e|e.oclIsKindOf(type))->select(e|e.name = name);

-- @comment this helper returns the new assertion contained by the library
helper context UML2!Constraint def: searchInLib : String =
  if self.owner.oclIsTypeOf(UML2!Class)
  then self.ClassExistInLib()
  else if self.owner.oclIsTypeOf(UML2!Association)
  then self.AssociationExistInLib()
  else if self.owner.oclIsTypeOf(UML2!Operation)
  then self.OperationExistInLib()
  else 'Erreur entry Type: either class or association or operation'
  endif
  endif
endif
;

-- @comment this helper is called by the helper searchInLib for the constraint is on a class
helper context UML2!Constraint def: ClassExistInLib() : String =
  if XML!Element.allInstances()->select(c|c.name='class')->
    select(c|c.getAttr('package').value = self.owner.package.name
      and c.getAttr('name').value = self.owner.name
      and c.getChildren(XML!Element, 'assertion')->
        exists(const|const.getAttr('name') = self.name)
    ).first() <> OclUndefined
  then
    XML!Element.allInstances()->select(c|c.name='class')->
      select(c|c.getAttr('package').value = self.owner.package.name
        and c.getAttr('name').value = self.owner.name
        and c.getChildren(XML!Element, 'assertion')->
          exists(const|const.getAttr('name') = self.name)
        ).first().getChildren(XML!Element, 'assertion')->
          select(const|const.getAttr('name') = self.name).first().value
  else 'noRule'
  endif
;

-- @comment this helper is called by the helper searchInLib for the constraint is on an
association
helper context UML2!Constraint def: AssociationExistInLib() : String =
  if XML!Element.allInstances()->select(c|c.name='association')->
    select(c|c.getAttr('package').value = self.owner.package.name
      and c.getAttr('name').value = self.owner.name

```



ATL Transformation

Catalogue of Model Transformations

Author

Eric Simon
eric.simon3 <at> gmail.com

Documentation

Aug 7th 2006

```
        and c.getChildren(XML!Element, 'assertion')->
            exists(const|const.getAttr('name').value = self.name)
    ).first() <> OclUndefined
then
XML!Element.allInstances()->select(c|c.name='association')->
select(c|c.getAttr('package').value = self.owner.package.name
and c.getAttr('name').value = self.owner.name
and c.getChildren(XML!Element, 'assertion')->
exists(const|const.getAttr('name').value = self.name)
).first().getChildren(XML!Element, 'assertion')->
select(const|const.getAttr('name').value = self.name).first().getAttr('value').value
else 'noRule'
endif
endif
;

-- @comment this helper is called by the helper searchInLib for the constraint is on an
operation
helper context UML2!Constraint def: OperationExistInLib() : String =
if self.owner.precondition->includes(self)
then
if XML!Element.allInstances()->select(c|c.name='operation')->
select(c|c.getAttr('package').value = self.owner.owner.package.name
and c.getAttr('class').value = self.owner.owner.name
and c.getChildren(XML!Element, 'precondition')->
exists(const|const.getAttr('name').value = self.name)
).first() <> OclUndefined
then
XML!Element.allInstances()->select(c|c.name='operation')->
select(c|c.getAttr('package').value = self.owner.owner.package.name
and c.getAttr('class').value = self.owner.owner.name
and c.getChildren(XML!Element, 'precondition')->
exists(const|const.getAttr('name').value = self.name)
).first().getChildren(XML!Element, 'precondition')->
select(const|const.getAttr('name').value = self.name).first().getAttr('value').value
else 'noRule'
endif
endif
else if self.owner.postcondition->includes(self)
then
if XML!Element.allInstances()->select(c|c.name='operation')->
select(c|c.getAttr('package').value = self.owner.owner.package.name
and c.getAttr('class').value = self.owner.owner.name
and c.getChildren(XML!Element, 'postcondition')->
exists(const|const.getAttr('name').value = self.name)
).first() <> OclUndefined
then XML!Element.allInstances()->select(c|c.name='operation')->
select(c|c.getAttr('package').value = self.owner.owner.package.name
and c.getAttr('class').value = self.owner.owner.name
and c.getChildren(XML!Element, 'postcondition')->
exists(const|const.getAttr('name').value = self.name)
).first().getChildren(XML!Element, 'postcondition')->
select(const|const.getAttr('name').value =
self.name).first().getAttr('value').value
else 'noRule'
endif
else 'noRule'
endif
endif
endif
;

2.4.1. -- @begin Model
rule Model {
from
inputM : UML2!Model
to
```



ATL Transformation

Catalogue of Model Transformations

Author

Eric Simon
eric.simon3 <at> gmail.com

Documentation

Aug 7th 2006

```
        outputM : UML2Target!Model (
            name <- inputM.name,
            ownedMember <- inputM.ownedMember
        )
    }
-- @end Model

2.4.2. -- @begin DataType
rule DataType {
    from
        inputC : UML2!DataType
    to
        outputC : UML2Target!DataType (
            name <- inputC.name
        )
}
-- @end DataType

2.4.3. -- @begin LiteralNull
rule LiteralNull {
    from
        inputLN : UML2!LiteralNull
    to
        outputLN : UML2Target!LiteralNull
}
-- @end LiteralNull

2.4.4. -- @begin LiteralInteger
rule LiteralInteger {
    from
        inputLI : UML2!LiteralInteger
    to
        outputLI : UML2Target!LiteralInteger (
            value <- inputLI.value
        )
}
-- @end LiteralInteger

2.4.5. -- @begin LiteralUnlimitedNatural
rule LiteralUnlimitedNatural {
    from
        inputLUN : UML2!LiteralUnlimitedNatural
    to
        outputLUN : UML2Target!LiteralUnlimitedNatural (
            value <- inputLUN.value
        )
}
-- @end LiteralUnlimitedNatural

2.4.6. -- @begin LiteralString
rule LiteralString {
    from
        inputLS : UML2!LiteralString
        (inputLS.owner.OperationExistInLib()='noRule')
    to
        outputLS : UML2Target!LiteralString (
            value <- inputLS.value
        )
}
-- @end LiteralString

2.4.7. -- @begin Association
rule Association {
    from
```



ATL Transformation

Catalogue of Model Transformations

Author

Eric Simon
eric.simon3 <at> gmail.com

Documentation

Aug 7th 2006


```
    inputA : UML2!Association
  to
    outputA : UML2Target!Association (
      name <- inputA.name,
      memberEnd <- inputA.memberEnd
    )
  }
-- @end Association

2.4.8. -- @begin Property
rule Property {
  from
    inputP : UML2!Property
  to
    outputP : UML2Target!Property (
      owningAssociation <- inputP.owningAssociation,
      name <- inputP.name,
      type <- inputP.type,
      upperValue <- inputP.upperValue,
      lowerValue <- inputP.lowerValue,
      defaultValue <-inputP.defaultValue
    )
  }
-- @end Property

2.4.9. -- @begin Constraint
rule Constraint {
  from
    inputC : UML2!Constraint
  to
    outputC : UML2Target!Constraint (
      name <- inputC.name,
      namespace <- inputC.namespace,
      specification <- if inputC.OperationExistInLib()='noRule'
        then inputC.specification
        else thisModule.newRule(inputC)
      endif
    )
  }
-- @end Constraint

2.4.10. -- @begin Class
rule Class {
  from
    inputC : UML2!Class
  to
    outputC : UML2Target!Class (
      name <- inputC.name,
      ownedOperation <- inputC.ownedOperation,
      nestedClassifier <- inputC.nestedClassifier,
      isActive <- inputC.isActive,
      ownedReception <- inputC.ownedReception,
      ownedAttribute <- inputC.ownedAttribute
    )
  }
-- @end Class

2.4.11. -- @begin Operation
rule Operation {
  from
    inputO : UML2!Operation
  to
    outputO : UML2Target!Operation (
      name <- inputO.name,
```

	ATL Transformation Catalogue of Model Transformations	Author Eric Simon eric.simon3 <at> gmail.com
	Documentation	Aug 7th 2006

```

class_ <- inputO.class_,
ownedRule <- inputO.ownedRule,
ownedParameter <- inputO.ownedParameter
)
}
-- @end Operation

2.4.12. -- @begin Parameter
rule Parameter {
  from
    inputP : UML2!Parameter
  to
    outputP : UML2Target!Parameter (
      name <- inputP.name,
      operation <- inputP.operation,
      type <- inputP.type
    )
}
-- @end Parameter

-- @comment this lazy rule replace the oldest assertion by a new
2.4.13. -- @begin newRule
lazy rule newRule {
  from
    inputC : UML2!Constraint
  to
    outputLS : UML2Target!LiteralString (
      value <- inputC.OperationExistInLib()
    )
}
-- @end newRule

```

3. References

- [1] Catalogue of Model Transformations
<http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf>