



ATL Transformation

Catalogue of Model Transformations

Author

Eric Simon
eric.simon3 <at> gmail.com

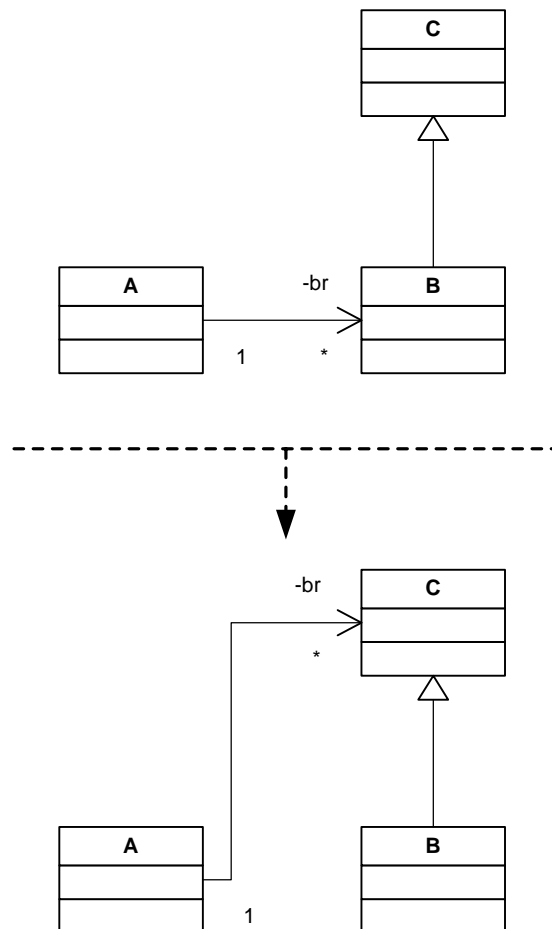
Documentation


Aug 10th 2006

| | | |
|------|---|---|
| 1. | ATL TRANSFORMATION EXAMPLE: DISAGGREGATION..... | 1 |
| 2. | ATL TRANSFORMATION OVERVIEW..... | 2 |
| 2.1. | DESCRIPTION | 2 |
| 2.2. | PURPOSE | 2 |
| 2.3. | RULES SPECIFICATION | 2 |
| 2.4. | ATL CODE..... | 4 |
| 3. | REFERENCES | 4 |

1. ATL Transformation Example: Raise supplier abstraction level

This example is extract from [Catalogue of Model Transformations](#) by K. Lano.
Section 2.13: Raise supplier abstraction level, page 27.



| | | |
|---|--|--|
|  | ATL Transformation Catalogue of Model Transformations | Author Eric Simon eric.simon3 <at> gmail.com |
| | Documentation | Aug 10th 2006 |

2. ATL Transformation overview

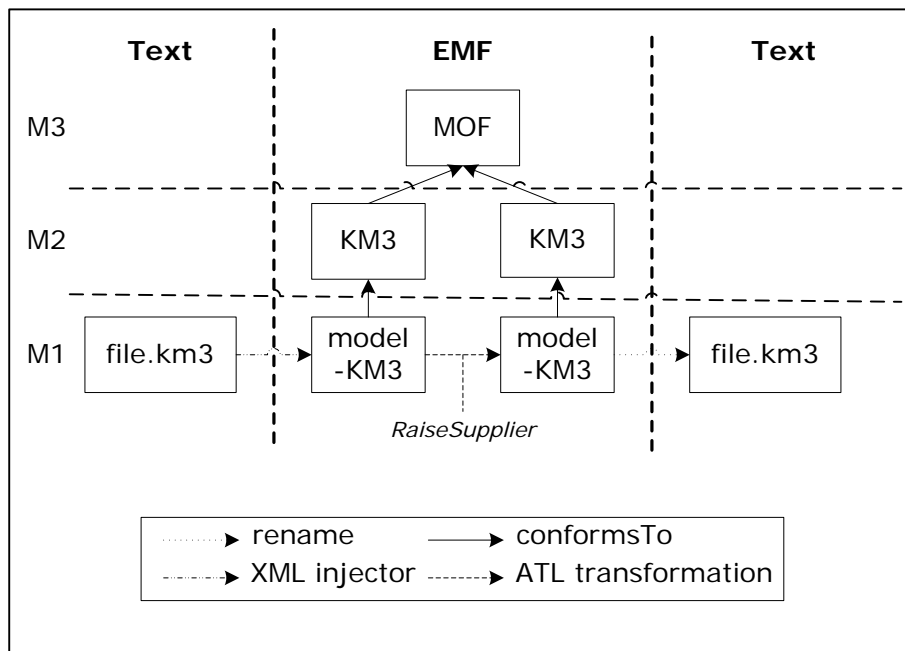


Fig 1. Overview of the transformation

2.1. Description

“A class is factored into component classes.”


2.2. Purpose

“A class may become large and unmanageable, with several loosely connected functionalities. It should be split into several classes, such as a master/controller class and helper classes, which have more coherent functionalities and data.”


2.3. Rules specification

The transformation has the same metamodel for the source and the target: KM3. . However, we choice two different name: KM3 and KM3Target, indeed there is a confusion with the rule `ocl: KM3!<nameElement>->allInstances()` which returns all the class appartain to the source **and** the target.

- For each [Metamodel](#) element, another *Metamodel* element is created with the following elements:
 - the attribute *location* is the same,

| | | |
|---|--|--|
|  | ATL Transformation Catalogue of Model Transformations | Author Eric Simon eric.simon3 <at> gmail.com |
| | Documentation | Aug 10th 2006 |

- the reference *contents* is the same.
- For each [Package](#) element, another *Package* element is created with the following elements:
 - the attribute *name* is the same,
 - the reference *contents* is the same.
- For each [DataType](#) element, another *DataType* element is created with the following elements:
 - the attributes *name* and *location* are the same,
- For each [EnumLiteral](#) element, another *EnumLiteral* element is created with the following elements:
 - the attributes *name* and *location* are the same,
 - the references *enum* and *package* are composed by the same source.
- For each [Enumeration](#) element, another *Enumeration* element is created with the following elements:
 - the attributes *name* and *location* are the same,
 - the reference *literals* and *package* are composed by the same source.
- For each [Class](#) element, another *Class* element is created with the following elements:
 - the attributes *name*, *location* and *isAbstract* are the same,
 - the references *supertypes* and *package* are the same one as the source,
 - the reference *structuralFeatures*, which are attributes, are the same,
 - the reference *structuralFeatures*, which are references with an opposite, are the same,
 - the reference *structuralFeatures*, which are references without an opposite, are modified: its are redirected to the root(s) SuperType(s).
- For each [Attribute](#) element, another *Attribute* element is created with the following elements:
 - the attributes *name*, *lower*, *upper*, *isOrdered* and *isUnique* are the same source value,
 - the references *package*, *owner* and *type*, are filled in with the same value respectively.
- For each *Reference* element
 - If this reference has an opposite
 - another [Reference](#) element is created with the following elements:
 - the attributes *name* and *isContainer* are the same,
 - the references *opposite*, *owner* and *package* are the same;
 - If this reference has not an opposite
 - For each root supertype, another [Reference](#) element is created with the following elements:
 - the attributes *name*, *location*, *isContainer* and *isAbstract* are the same,
 - the references *opposite*, *owner* and *package* are the same;
 - the reference *type* is the root supertype.

| | | |
|---|--|--|
|  | ATL Transformation Catalogue of Model Transformations | Author Eric Simon eric.simon3 <at> gmail.com |
| | Documentation | Aug 10th 2006 |

2.4. ATL Code

```

module RaiseSupplier; -- Module Template
create OUT : KM3 from IN : KM3;

-- @comment this helper returns the root SuperTypes of an element (it is a recursive helper)
helper context KM3!Class def: getRootSuperTypes : Sequence(KM3!Class) =
  if self.supertypes->isEmpty()
  then Sequence{}
  else self.supertypes->select(c | c.supertypes->notEmpty())
    ->iterate(a; acc : Sequence(KM3!Class)=Sequence{} | acc->including(a.getRootSuperTypes))
    ->union(
      self.supertypes->select(c | c.supertypes->isEmpty())
    ->iterate(a; acc : Sequence(KM3!Class)=Sequence{} | acc->including(a) )
    ).flatten()
  endif;


2.4.1. --@begin rule Metamodel
rule Metamodel {
  from
    inputMm:KM3!Metamodel
  to
    outputMm:KM3!Metamodel (
      location <- inputMm.location,
      contents <- inputMm.contents
    )
}
--@end rule Metamodel

2.4.2. --@begin rule Package
rule Package {
  from
    inputPkg:KM3!Package
  to
    outputPkg:KM3!Package (
      name <- inputPkg.name,
      contents <- inputPkg.contents
    )
}
--@end rule Package

2.4.3. --@begin rule DataType
rule DataType {
  from
    inputData:KM3!DataType
  to
    outputData:KM3!DataType(
      name <- inputData.name,
      location <- inputData.location
    )
}
--@end rule DataType

2.4.4. --@begin rule EnumLiteral
rule EnumLiteral {
  from
    inputL:KM3!EnumLiteral
  to
    outputL:KM3!EnumLiteral (
      name <- inputL.name,
      location <- inputL.location,

```

| | | |
|---|--|--|
|  | ATL Transformation Catalogue of Model Transformations | Author Eric Simon eric.simon3 <at> gmail.com |
| | Documentation | Aug 10th 2006 |

```


        enum <- inputL.enum,
        package <- inputL.package
    )
}
--@end rule EnumLiteral

2.4.5. --@begin rule Enumeration
rule Enumeration {
    from
        inputEnum:KM3!Enumeration
    to
        outputEnum:KM3!Enumeration (
            name <- inputEnum.name,
            location <- inputEnum.location,
            package <- inputEnum.package,
            literals <- inputEnum.literals
        )
}
--@end rule Enumeration

2.4.6. --@begin rule Class
rule Class {
    from
        inputC:KM3!Class
    to
        outputC:KM3!Class (
            isAbstract <- inputC.isAbstract,
            supertypes <- inputC.supertypes,
            name <- inputC.name,
            location <- inputC.location,
            package <- inputC.package,
            structuralFeatures <- inputC.structuralFeatures->
                select(r | r.oclIsTypeOf(KM3!Reference))->select(r | r.opposite.oclIsUndefined())-
>
            iterate(a;acc : Sequence(KM3!Reference) = Sequence{} |
                if a.type.oclIsTypeOf(KM3!Class)
                    then acc->append(a.type.getRootSuperTypes->iterate(b;
acc1:Sequence(KM3!Reference) = Sequence{}|
                    acc1->append(thisModule.InheritAndAssociation(b,a)))->flatten()
                    else acc
                endif),
            structuralFeatures <- inputC.structuralFeatures
                ->select(r | r.oclIsTypeOf(KM3!Reference))
                ->select(r | not r.opposite.oclIsUndefined()),
            structuralFeatures <- inputC.structuralFeatures
                ->select(r | not r.oclIsTypeOf(KM3!Reference))
        )
}
--@end rule Class

2.4.7. --@begin rule Attribute
rule Attribute {
    from
        inputAttr : KM3!Attribute
    to
        outputAttr : KM3!Attribute (
            package <- inputAttr.package,
            name <- inputAttr.name,
            lower <- inputAttr.lower,
            upper <- inputAttr.upper,
            isOrdered <- inputAttr.isOrdered,
            isUnique <- inputAttr.isUnique,
            owner <- inputAttr.owner,
            type <- inputAttr.type

```

| | | |
|---|--|--|
|  | ATL Transformation Catalogue of Model Transformations | Author Eric Simon eric.simon3 <at> gmail.com |
| | Documentation | Aug 10th 2006 |

```

    )
}
--@end rule Attribute

2.4.8. --@begin rule Reference
rule ReferenceWithOpposite {
  from
    inputRef : KM3!Reference
    (not inputRef.opposite.oclIsUndefined())
  to
    outputRef : KM3!Reference (
      package <- inputRef.package,
      name <- inputRef.name,
      lower <- inputRef.lower,
      upper <- inputRef.upper,
      isOrdered <- inputRef.isOrdered,
      isUnique <- inputRef.isUnique,
      owner <- inputRef.owner,
      type <- inputRef.type,
      isContainer <- inputRef.isContainer,
      opposite <- inputRef.opposite
    )
}
--@end rule Reference

2.4.9. -- @comment this lazy rule creates a reference for a given supertypes end another
reference
lazy rule InheritAndAssociation{
  from
    supertype:KM3!Class,
    reference:KM3!Reference

  to
    refChildren : KM3!Reference (
      package <- reference.package,
      name <- reference.name,
      lower <- reference.lower,
      upper <- reference.upper,
      isOrdered <- reference.isOrdered,
      isUnique <- reference.isUnique,
      owner <- reference.owner,
      type <- supertype,
      isContainer <- reference.isContainer
    )
}

```

3. References

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