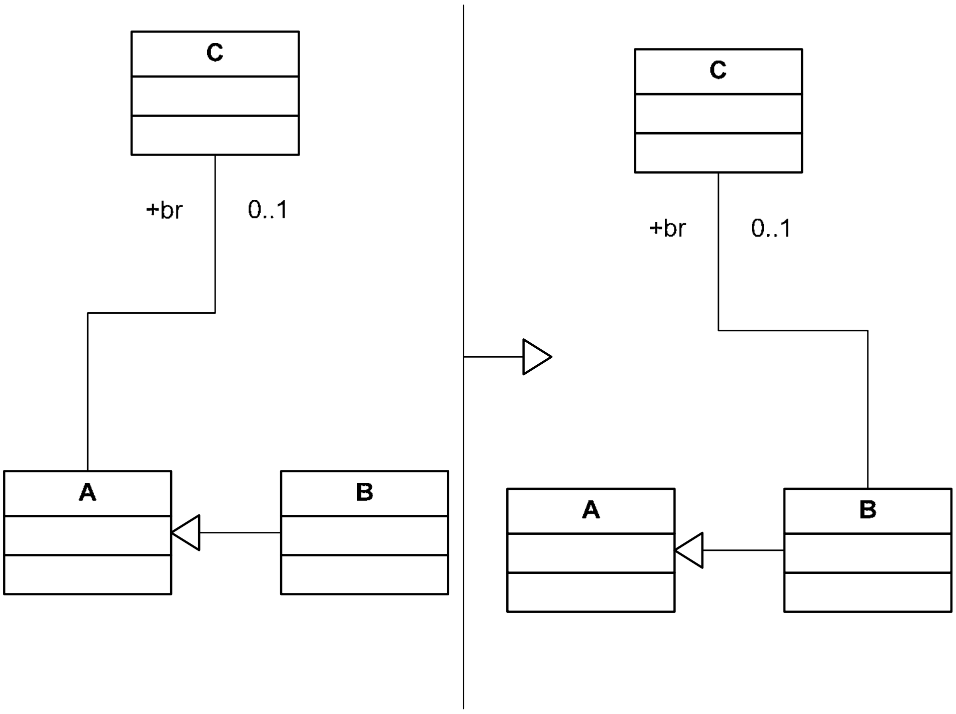
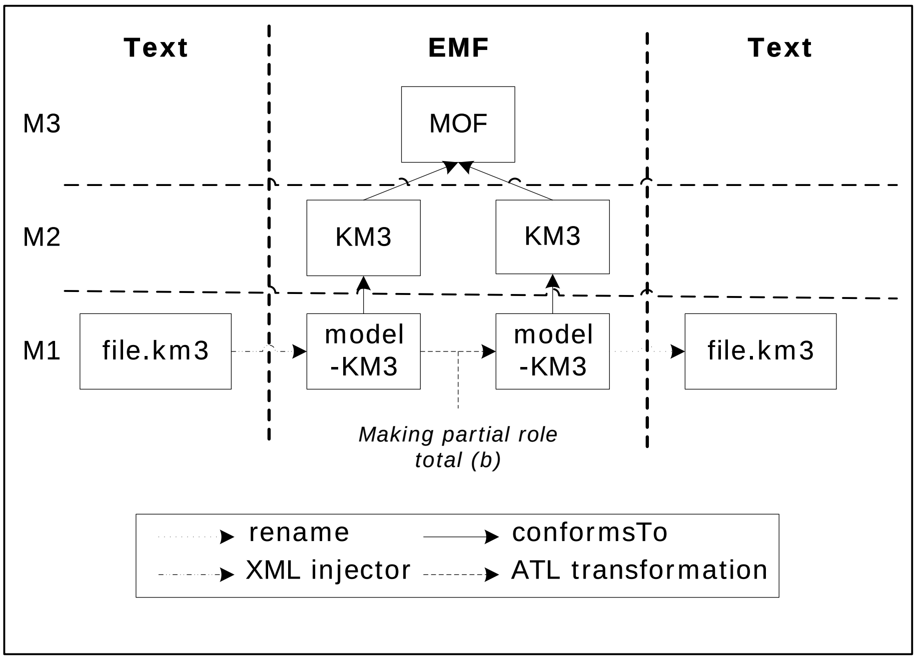
	ATL Transformation Catalogue of Model Transformations	Author Baudry Julien Jul.baudry <*at*> gmail.com
	Documentation	Aug 8th 2006

1. ATL Transformation Example: making partial role total (a)

This example is extract from Catalogue of Model Transformations by K. Lano.
Section 2.14: making partial role total (b), page 23.





2. ATL Transformation overview

2.1. Description

A 0..1 multiplicity role of a class A may be turned into a 1 multiplicity role by either moving the role to a superclass of its current target, or by moving the other end to a subclass of A on which the association is total.

2.2. Purpose

Total associations are generally easier to implement and manage than partial associations. The previous figure shows the 'generalise target' version of this transformation.

2.3. Rules specification

Our transformation has the same source and the target metamodel, KM3. We use 2 different names (KM3 and KM3target), but they refer to the same metamodel.

We use the helper *hasChild ()*, who return true if a class has the current class as children, referring to inheritance.

- For a Metamodel element, another Metamodel element is created :
  - with the same name and location,
  - Linked to the same contents.

- For a Package element, another Package element is created :
  - with the same name,
  - Linked to the same contents.

- For a DataType element, another DataType element is created :
  - With the same name and location.

- For a Enumeration element, another Enumeration element is created :
  - with the same name, package and location,
  - Linked to the same literals.

- For a EnumLiteral element, another EnumLiteral element is created :
  - With the same name, package and location.

- For a Class element, we create another Class element if the Input Class doesn't match with these 3 cases:
  - First case:
    - *hasChild* () return true,
    - The class as a reference with a cardinality 1..0 and its opposite 1..1;
  - Second case:
    - The helper *hasChild* () using with the type of the reference return true,
    - The class as a reference with a cardinality 1..1 and its opposite 1..0;
  - Third case:
    - The class has supertypes,
    - The class as a reference with a cardinality 1..0 and its opposite 1..1.

  If the class doesn't match one of these cases, we create a class:
  - With the same name, location and package,
  - With the same property isAbstract,
  - Link to the same structuralFeatures and supertypes.

- For a Attribute element , another Attribute element is created :
  - With the same name, package, owner and type,
  - With the same properties isOrdered and isUnique,
  - With the same upper and lower values.

- For a Reference element, we create another Reference element if the Input Reference :
  - First case:
    - *hasChild* () return true,
    - The class as a reference with a cardinality 1..0 and its opposite 1..1;
  - Second case:
    - The helper *hasChild* () using with the type of the reference return true,
    - The class as a reference with a cardinality 1..1 and its opposite 1..0;

  If the class doesn't match one of these cases, we create a class:
  - With the same name, location, package, owner, type and opposite,
  - With the same property isOrdered, isUnique and isContainer,
  - Link to the same upper and lower values.

- The last rule has 5 input elements, a Class **inputSuperType**, a Class **inputChild**, a Class **inputClass**, a Reference **inputRef**, a Reference **inputRef2**:
    - InputRef has a cardinality 1..0,
    - InputRef2 has a cardinality 1..1,
    - inputRef is owned by inputSuperType,
    - inputRef2 is owned by inputClass,
    - InputRef2 is the opposite of inputRef,
    - InputRef and InputRef2 is not container.

  If the class matches these conditions, we create:
    - A Class outputSupertype from inputSuperType,
        - With the same property isAbstract,
        - With the same, location and package,
        - Linked to the same supertypes,
        - Linked to the same structuralFeatures, except InputRef;
    - A Class outputClass from inputClass,
        - With the same property isAbstract,
        - With the same, location and package,
        - Linked to the same supertypes,
        - Linked to the same structuralFeatures, except InputRef2;
    - A Class outputChild from inputChild,
        - With the same property isAbstract,
        - With the same, location and package,
        - Linked to the same supertypes,
        - outputRef in his structuralFeatures;
    - A Reference OuputRef from inputRef,
        - With the same name, location, package,
        - With the same property isOrdered, isUnique and isContainer,
        - With 1 as upper and lower value,
        - With outputClass as type,
        - With outputChild as owner;
    - A Reference OuputRef2 from inputRef2.
        - With the same name, location, package and owner,
        - With the same property isOrdered, isUnique and isContainer,
        - With 1 as upper and lower value,
        - With outputChild as type.

2.4. ATL Code

```
-- @name  Making partial role total (a)
-- @version    1.0
-- @domains    Catalogue of Model Transformations
-- @authors    Baudry Julien (jul.baudry<at>gmail.com)
-- @date    2006/08/02
-- @description   The purpose of this transformation is to making a patial role total
-- @see http://www.dcs.kcl.ac.uk/staff/kcl/tcat.pdf
-- @see section 2.14, page 23
-- @see author of article : K. Lano

module PartialRolesTotalB; -- Module Template
create OUT : KM3Target from IN : KM3;

helper context KM3!Class def: hasChild : Boolean =
   KM3!Class.allInstances()->select(c|c.supertypes->notEmpty())->exists(r|r.supertypes.first()
= self)
   ;

--@begin rule Metamodel
rule Metamodel {
   from
      inputMm:KM3!Metamodel
   to
      outputMm:KM3Target!Metamodel (
         location <- inputMm.location,
         contents <- inputMm.contents
      )
}
--@end rule Metamodel

--@begin rule Package
rule Package {
   from
      inputPkg:KM3!Package
   to
      outputPkg:KM3Target!Package (
         name <- inputPkg.name,
         contents <- inputPkg.contents
      )
}
--@end rule Package

--@begin rule DataType
rule DataType {
   from
      inputData:KM3!DataType
   to
      outputData:KM3Target!DataType(
         name <- inputData.name,
         location <- inputData.location
      )
}
--@end rule DataType

--@begin rule Enumeration
rule Enumeration {
   from
      inputEnum:KM3!Enumeration
   to
      outputEnum:KM3Target!Enumeration (
         name <- inputEnum.name,
         location <- inputEnum.location,
         package <- inputEnum.package,
         literals <- inputEnum.literals
      )
}
--@end rule Enumeration


--@begin rule EnumLiteral
rule DataType {
   from
      inputL:KM3!EnumLiteral
   to
      outputL:KM3Target!EnumLiteral (
         name <- inputL.name,
         location <- inputL.location,
         package <- inputL.package
      )
}
--@end rule EnumLiteral

--@begin rule Class
rule Class {
   from
      inputC:KM3!Class
      (     not( inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.upper=1)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.lower=0)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.opposite.upper=1)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.opposite.lower=1)
               and inputC.hasChild
            )
            and
            not( inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.upper=1)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.lower=1)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.opposite.upper=1)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(r|r.opposite.lower=0)
               and inputC.structuralFeatures->select(r|r.oclIsTypeOf(KM3!Reference))-
>exists(c|c.type.hasChild)
            )
            and
            not( inputC.supertypes->exists(r|r.structuralFeatures-
>select(r|r.oclIsTypeOf(KM3!Reference))->exists(r|r.upper=1))
               and inputC.supertypes->exists(r|r.structuralFeatures-
>select(r|r.oclIsTypeOf(KM3!Reference))->exists(r|r.lower=0))
               and inputC.supertypes->exists(r|r.structuralFeatures-
>select(r|r.oclIsTypeOf(KM3!Reference))->exists(r|r.opposite.upper=1))
               and inputC.supertypes->exists(r|r.structuralFeatures-
>select(r|r.oclIsTypeOf(KM3!Reference))->exists(r|r.opposite.lower=1))
               and inputC.supertypes->notEmpty()
            )
         )
         to
      outputC:KM3Target!Class (
         isAbstract <- inputC.isAbstract,
         supertypes <- inputC.supertypes,
         name <- inputC.name,
         location <- inputC.location,
         package <- inputC.package,
         structuralFeatures <- inputC.structuralFeatures
      )
}
--@end rule Class

--@begin rule Attribute
rule Attribute {
   from
      inputAttr : KM3!Attribute
   to
      outputAttr : KM3Target!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
      )
}
--@end rule Attribute

--@begin rule Reference
rule Reference {
   from
      inputRef : KM3!Reference
         (  not( inputRef.upper=1
               and inputRef.lower=0
               and inputRef.opposite.upper=1
               and inputRef.opposite.lower=1
               and inputRef.owner.hasChild
            )
            and
            not( inputRef.upper=1
               and inputRef.lower=1
               and inputRef.opposite.upper=1
               and inputRef.opposite.lower=0
               and inputRef.type.hasChild
            )
         )
   to
      outputRef : KM3Target!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 Attribute

--@begin rule Merging
rule PartialRoles {
   from
      inputSuperType : KM3!Class,
      inputChild : KM3!Class,
      inputClass : KM3!Class,
      inputRef : KM3!Reference,
      inputRef2 : KM3!Reference
   ( (inputChild.supertypes->includes(inputSuperType)
      and inputRef.owner = inputSuperType
      and inputRef2.owner = inputClass
      and inputRef.upper = 1
      and inputRef.lower = 0
      and not inputRef.isContainer
      and inputRef2.upper = 1
      and inputRef2.lower = 1
      and not inputRef2.isContainer
      and inputRef.opposite=inputRef2
      )
   )
   to
      outputSuperType: KM3Target!Class(
         isAbstract <- inputSuperType.isAbstract,
         supertypes <- inputSuperType.supertypes,
         name <- inputSuperType.name,
         location <- inputSuperType.location,
         package <- inputSuperType.package,
         structuralFeatures <- inputSuperType.structuralFeatures->select(r| r<>inputRef)
         ),
      outputClass: KM3Target!Class(
         isAbstract <- inputClass.isAbstract,
         supertypes <- inputClass.supertypes,
         name <- inputClass.name,
         location <- inputClass.location,
         package <- inputClass.package,
         structuralFeatures <- inputClass.structuralFeatures->select(r|r<>inputRef2),
         structuralFeatures <- outputRef2
         ),
      outputRef: KM3Target!Reference(
         package <- inputRef.package,
         name <- inputRef.name,
         lower <- 1,
         upper <- 1,
         isOrdered <- inputRef.isOrdered,
         isUnique <- inputRef.isUnique,
         owner <- outputChild,
         type <- outputClass,
         isContainer <- false,
         opposite <- outputRef2
         ),
      outputChild: KM3Target!Class(
         isAbstract <- inputChild.isAbstract,
         supertypes <- inputChild.supertypes,
         name <- inputChild.name,
         location <- inputChild.location,
         package <- inputChild.package,
         structuralFeatures <- inputChild.structuralFeatures,
         structuralFeatures <- outputRef
         ),
      outputRef2: KM3Target!Reference(
         package <- inputRef2.package,
         name <- inputRef2.name,
         lower <- 1,
         upper <- 1,
         isOrdered <- inputRef2.isOrdered,
         isUnique <- inputRef2.isUnique,
         owner <- inputRef2.owner,
         type <- outputChild,
         isContainer <- false,
         opposite <- outputRef
         )

}
```

3. References

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