1. ATL Transformation Example: Removal a many-many association

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

The purpose of this transformation is to substitute a many-many association by an introduction of class with two many-one associations.

2.2. Purpose

“Explicit many-many associations cannot be implemented using foreign keys in a relational database – an intermediary table would need to be used instead. This transformation is the object-oriented equivalent of introducing such a table.”

2.3. Rules specification

- Rule Metamodel: for each Metamodel element, another Metamodel element is created with the following elements:
  - the attribute location is the same,
  - the reference contents is the same.
- Rule Package: 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.
• Rule **DataType**: for each `DataType` element, another `DataType` element is created with the following elements:
  o the attributes `name` and `location` are the same.

• Rule **EnumLiteral**: for each `EnumLiteral` element, another `EnumLiteral` element is created with the following elements:
  o the attributes `name` and `location` are the same,
  o the references `enum` and `package` are composed by the same source.

• Rule **Enumeration**: for each `Enumeration` element, another `Enumeration` element is created with the following elements:
  o the attributes `name` and `location` are the same,
  o the reference `literals` and `package` are composed by the same source.

• Rule **Class**: for each `Class` element
  o If the `Class` element contained a reference which is not contained by a many-many association
    ▪ another `Class` element is created with the following elements:
      • the attributes `name`, `location` and `isAbstract` are the same,
      • the references `structuralFeatures`, `supertypes` and `package` are the same.

• Rule **Attribute**: for each `Attribute` element, another `Attribute` element is created with the following elements:
  o the attributes `name`, `lower`, `upper`, `isOrdered` and `isUnique` are the same source value,
  o the references `package`, `owner` and `type`, are filled in with the same value respectively.

• Rule **Reference**: for each `Reference` element
  o If the `Reference` element is not contained by a many-many association
    ▪ another `Reference` element is created with the following elements:
      • the attributes `name` and `isContainer` are the same,
      • the references `owner`, `opposite`, `type` and `package` are the same;

• Rule **Association**: for each pair of `Reference` element which is considered like many-many association
  o a `Class` element is created with the following elements:
    ▪ the elements of both `Class`, which are linked by this pair of `Reference`, composed this new `Class` element
2.4. ATL Code

module Removing; // Module Template
create OUT : KM3Target from IN : KM3;

-- @comment this helper allows to know if a reference own the properties necessary for the
rule association
helper context KM3!Reference def: isManyToManyNotContainer : Boolean =
  self.lower = 0 and self.upper < 0 and not self.isContainer
;
-- @comment this helper create a Map which uses in the rule isAlreadyConsidered.
helper def: assoMap : Map(KM3!Reference, Sequence(KM3!Reference)) = Map{};
-- @comment this rule allows to know if a pair of element is already considered. E.g.: (A,B)
and [B,A] => (A,B).
rule isAlreadyConsidered(ref1 : KM3!Reference, ref2 : KM3!Reference) {
  do {
    if (not thisModule.assoMap.get(ref2).oclIsUndefined()) {
      if (thisModule.assoMap.get(ref2)->includes(ref1)) {
        true;
      }
    } else {
      if (not thisModule.assoMap.get(ref1).oclIsUndefined()) {
        thisModule.assoMap <-
        thisModule.assoMap.including(ref1,thisModule.assoMap.get(ref1)->including(ref2));
        false;
      } else {
        thisModule.assoMap <- thisModule.assoMap.including(ref1, Sequence{ref2});
        false;
      }
    } else {
      if (not thisModule.assoMap.get(ref1).oclIsUndefined()) {
        thisModule.assoMap <-
        thisModule.assoMap.including(ref1,thisModule.assoMap.get(ref1)->including(ref2));
        false;
      } else {
        thisModule.assoMap <- thisModule.assoMap.including(ref1, Sequence{ref2});
        false;
      }
    }
  }
}

2.4.1. @begin rule Metamodel
rule Metamodel {
  from
    inputMm:KM3!Metamodel
  to
    outputMm:KM3Target!Metamodel {
      location <- inputMm.location,
      contents <- inputMm.contents
    }
-- @end rule Metamodel
2.4.2. -- @begin rule Package
rule Package {
  from
    inputPkg:KM3!Package
  to
    outputPkg:KM3Target!Package {
      name <- inputPkg.name,
      contents <- inputPkg.contents
    }
} -- @end rule Package

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

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

2.4.5. -- @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

2.4.6. -- @begin rule Class
rule Class {
from
   inputC:KM3!Class
   (not inputC.structuralFeatures->select(a|aoclIsTypeOf(KM3!Reference))->exists(r|r.isManyToManyNotContainer and r.opposite.isManyToManyNotContainer))
   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

2.4.7. -- @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

2.4.8. -- @begin rule Reference
rule Reference {
   from
      inputRef : KM3!Reference
      (not (inputRef.isManyToManyNotContainer and
      inputRef.opposite.isManyToManyNotContainer))
   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 Reference

-- @comment This rule takes a pair of Reference and, if these are not already considered,
creates a class with two many-one association.
2.4.9. -- @begin rule Association
rule Association {
  from
  inputA : KM3!Reference, 
  inputB : KM3!Reference 
  {
    inputA.opposite = inputB 
    and inputA.isManyToManyNotContainer 
    and inputB.isManyToManyNotContainer 
    -- and inputA <> inputB 
    and not thisModule.isAlreadyConsidered(inputA, inputB) 
  } 
  to
  outputA : KM3Target!Class ( 
    package <- inputA.owner.package, 
    name <- inputA.owner.name, 
    isAbstract <- inputA.owner.isAbstract, 
    structuralFeatures <- inputA.owner.structuralFeatures- >select(b|b.oclIsTypeOf(KM3!Reference))->select(a|not a.isManyToManyNotContainer), 
    structuralFeatures <- inputA.owner.structuralFeatures- >select(b|b.oclIsTypeOf(KM3!Attribute)), 
    structuralFeatures <- referenceAC 
  ), 
  outputB : KM3Target!Class ( 
    package <- inputB.owner.package, 
    name <- inputB.owner.name, 
    isAbstract <- inputB.owner.isAbstract, 
    structuralFeatures <- inputB.owner.structuralFeatures- >select(b|b.oclIsTypeOf(KM3!Reference))->select(a|not a.isManyToManyNotContainer), 
    structuralFeatures <- inputB.owner.structuralFeatures- >select(b|b.oclIsTypeOf(KM3!Attribute)), 
    structuralFeatures <- referenceBC 
  ), 
  outputC : KM3Target!Class ( 
    package <- inputA.owner.package, 
    name <- inputA.owner.name->concat(inputB.owner.name), 
    isAbstract <- false, 
    structuralFeatures <- referenceCA, 
    structuralFeatures <- referenceCB 
  ), 
  referenceAC : KM3Target!Reference ( 
    name <- inputA.name, 
    lower <= 1, 
    upper <= 1, 
    isOrdered <- false, 
    isUnique <- false, 
    owner <- outputA, 
    isContainer <- false, 
    opposite <- referenceCA 
  ), 
  referenceCA : KM3Target!Reference ( 
    name <- outputC.name->concat(‘1’), 
    lower <= 0, 
    upper <= 0-1, 
    isOrdered <- false, 
    isUnique <- false, 
    owner <- outputC, 
    isContainer <- false, 
    opposite <- referenceAC 
  ), 
  referenceBC : KM3Target!Reference ( 

}
name <- inputB.name,
lower <- 1,
upper <- 1,
isOrdered <- false,
isUnique <- false,
owner <- outputB,
isContainer <- false,
opposite <- referenceCB
),
referenceCB : KM3Target!Reference {
    name <- outputC.name->concat('2'),
lower <- 0,
upper <- 0-1,
isOrdered <- false,
isUnique <- false,
owner <- outputC,
isContainer <- false,
opposite <- referenceBC
}

--@end rule Association

3. References

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