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:
  o with the same name,
  o Linked to the same contents.

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

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

• For a EnumLiteral element, another EnumLiteral element is created:
  o 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:
  o First case:
    ▪ `hasChild () return true`,
    ▪ The class as a reference with a cardinality 1..0 and its opposite 1..1;
  o 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;
  o 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:
  o With the same name, location and package,
  o With the same property isAbstract,
  o Link to the same structuralFeatures and supertypes.

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

• For a Reference element, we create another Reference element if the Input Reference:
  o First case:
    ▪ `hasChild () return true`,
    ▪ The class as a reference with a cardinality 1..0 and its opposite 1..1;
  o 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:
  o With the same name, location, package, owner, type and opposite,
  o With the same property isOrdered, isUnique and isContainer,
  o 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`,
```plaintext
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
```
``` ATL
--@begin rule EnumLiteral

--@end rule EnumLiteral

--@begin rule Class

```
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/kel/tcat.pdf