## 1. ATL Transformation Example: introducing an interface

This example is extract from *Catalogue of Model Transformations* by K. Lano. Section 2.6: Introduce interface(s) for supplier class(es), page 16.

![ATL Diagram]

1. ATL TRANSFORMATION EXAMPLE: INTRODUCING AN INTERFACE.......................... 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 .............................................................................................................. 7
2. ATL Transformation overview

2.1. Description

If class A is a client of class B, but only uses some of B's operations, introduce an interface B_I of B which has the subset of operations of B that are used by A. Make A a client B_I instead of B.

2.2. Purpose

This reduces the dependencies in the model and enables separate development of A and B, and permits them to be placed in different layers or tiers of the system.

2.3. Rules specification

Our transformation has the same source and the target metamodel, UML2. We use 2 different names (UML2 and UML2target), but they refer to the same metamodel. We use a second model, an XML file which describes the operation of a class uses by another class.

For example, the following model says that Class A use the b1 and b2 operation, both operation of the class B. And the Class B uses the a1 operation, both operation of the class A.
We use the helper `getOperations()` function. This helper returns in a sequence the operations of the class 'class' use by the class 'subClass'.

- For a Model element, another Model element is created:
  - with the same name, visibility and `packageableElement_visibility`,
  - Linked to the same `ownedMember`.

- For a Class element, another Class element is created:
  - with the same name, visibility and `packageableElement_visibility`,
  - with the same properties, `isAbstract`, `isLeaf` and `isActive`,
  - Linked to the same `ownedAttribute` and `ownedOperation`.

- For a Association element, another Association element is created:
  - with the same name, visibility and `packageableElement_visibility`,
  - with the same properties, `isAbstract`, `isLeaf` and `isDerived`,
  - Linked to the same `ownedEnd` and `memberEnd`.

- For a LiteralNull/ LiteralInteger/ LiteralUnlimitedNatural element, another LiteralNull/ LiteralInteger/ LiteralUnlimitedNatural element is created:
  - With the same name, and value.

- For a Operation element, another Operation element is created:
  - with the same name and visibility,
  - With the same properties, `isAbstract`, `isLeaf`, `isOrdered`, `isQuery`, `isStatic`, `isUnique`.

- For a XML element, another Operation element is created:
  - With the same name.

- For a Property element, 3 elements are created:
  - A property element:
    - With the same properties `isDerived`, `isDerivedUnion`, `isLeaf`, `isOrdered`, `isReadOnly`, `isStatic`, `isUnique`,
2.4. ATL Code

``` ATL
module UML2Transformations; -- Module Template
create OUT : UML2target from IN : UML2, MODEL : XML;

-- helper getOperations
-- IN : UML2!Class, UML2!Class
-- OUT : Sequence(UML2!Operation)
-- this helper returns in a sequence the operations of the class 'class' use by the class 'sub class'
helper context UML2!Property def: getOperations(class : UML2!Class, subClass : UML2!Class) : Sequence(UML2!Operation) =
XML!Root.allInstances()->asSequence()->first().children
->select(a|a.name = class.name)->first().children
->select(a|(a.name = subClass.name)and((a.children->select(a|a.name='model')
->first().value)=subClass.package.name))
->first().children
->select(a|a.oclIsTypeOf(XML!Element));

--@begin rule model
rule model {
  from
  inputModel : UML2!Model
to
  outputModel : UML2target!Model {
    name <- inputModel.name,
    visibility <- inputModel.visibility,
    packageableElement_visibility <- inputModel.packageableElement_visibility,
    ownedMember <- inputModel.ownedMember
  }
}
--@end rule model
```
```plaintext
---@begin rule class
rule class { 
  from 
  inputClass : UML2!Class 
  to 
  outputClass : UML2target!Class ( 
    name <- inputClass.name, 
    visibility <- inputClass.visibility, 
    packageableElement_visibility <- inputClass.packageableElement_visibility, 
    isAbstract <- inputClass.isAbstract, 
    isLeaf <- inputClass.isLeaf, 
    isActive <- inputClass.isActive, 
    ownedAttribute <- inputClass.ownedAttribute, 
    ownedOperation <- inputClass.ownedOperation 
  )
}
---@end rule class

---@begin association
rule association { 
  from 
  inputAssoc : UML2!Association 
  to 
  outputAssoc : UML2target!Association ( 
    isAbstract <- inputAssoc.isAbstract, 
    isDerived <- inputAssoc.isDerived, 
    isLeaf <- inputAssoc.isLeaf, 
    ownedEnd <- inputAssoc.ownedEnd, 
    memberEnd <- inputAssoc.memberEnd, 
    name <- inputAssoc.name, 
    packageableElement_visibility <- inputAssoc.packageableElement_visibility, 
    visibility <- inputAssoc.visibility 
  )
}
---@end association

---@begin rule property
rule property { 
  from 
  inputProperty : UML2!Property 
  to 
  outputProperty : UML2target!Property ( 
    isDerived <- inputProperty.isDerived, 
    isDerivedUnion <- inputProperty.isDerivedUnion, 
    isLeaf <- inputProperty.isLeaf, 
    isOrdered <- inputProperty.isOrdered, 
    isReadOnly <- inputProperty.isReadOnly, 
    isStatic <- inputProperty.isStatic, 
    isUnique <- inputProperty.isUnique, 
    name <- inputProperty.name, 
    visibility <- inputProperty.visibility, 
    lowerValue <- inputProperty.lowerValue, 
    upperValue <- inputProperty.upperValue, 
    type <- outputInterface 
  ), 
  outputInterface : UML2target!Interface ( 
    name <- inputProperty.name+'_interface_'+inputProperty.association.name, 
    ownedOperation <- inputProperty.getOperations(inputProperty.association.endType->excluding(inputProperty.type)->first(),
      inputProperty.type)
  ) 
  Sequence{UML2target!Operation} = Sequence{} 
}
```

>includin(thisModule.operationXML(a))

inputImplementation : UML2target!Implementation {
    name <- inputProperty.name+"_implementation_"+inputProperty.association.name,
    contract <- outputInterface,
    implementingClassifier <- inputProperty.type
}

--@end rule property

--@begin literal null
rule literalNull {
    from
        inputLiteral : UML2!LiteralNull
    to
        outputLiteral : UML2target!LiteralNull {
            name <- inputLiteral.name,
            value <- inputLiteral.value
        }
}

--@end literal null

--@begin literal integer
rule literalInteger {
    from
        inputLiteral : UML2!LiteralInteger
    to
        outputLiteral : UML2target!LiteralInteger {
            name <- inputLiteral.name,
            value <- inputLiteral.value
        }
}

--@end literal integer

--@begin literal unlimited natural
rule literalUnlimitedNatural {
    from
        inputLiteral : UML2!LiteralUnlimitedNatural
    to
        outputLiteral : UML2target!LiteralUnlimitedNatural {
            name <- inputLiteral.name,
            value <- inputLiteral.value
        }
}

--@end literal unlimited natural

--@begin operation
rule operation {
    from
        inputOperation : UML2!Operation
    to
        outputOperation : UML2target!Operation {
            isAbstract <- inputOperation.isAbstract,
            isLeaf <- inputOperation.isLeaf,
            isOrdered <- inputOperation.isOrdered,
            isQuery <- inputOperation.isQuery,
            isStatic <- inputOperation.isStatic,
            isUnique <- inputOperation.isUnique,
            name <- inputOperation.name,
            visibility <- inputOperation.visibility
        }
}

--@end operation
3. References

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