Advanced Applications Of Model-to-Model Transformation

Hugo Bruneliere & Frédéric Jouault

INRIA

Context of this work

- The present courseware has been elaborated in the context of the MODELPLEX European IST FP6 project (http://www.modelplex.org/).
- Co-funded by the European Commission, the MODELPLEX project involves 21 partners from 8 different countries.
- MODELPLEX aims at defining and developing a coherent infrastructure specifically for the application of MDE to the development and subsequent management of complex systems within a variety of industrial domains.
- To achieve the goal of large-scale adoption of MDE, MODELPLEX promotes the idea of a collaborative development of courseware dedicated to this domain.
- The MDE courseware provided here with the status of open-source software is produced under the EPL 1.0 license.

Outline

- Overview of advanced ATL features
  - Implicit resolution, ResolveTemp & lazy rules
  - Imperative "Do" section & Called Rules
  - Rule Inheritance
  - Refining mode
- Chaining ATL transformations using ANT scripts
  - ATL-specific ANT tasks
  - Concrete use
- Handling UML2 models & profiles with ATL
  - The UML24ATL model handler
  - Concrete use
- Handling weaving models with ATL & AMW
  - The AMW4ATL model handler
  - Concrete use
- High-Order Transformations: Handling of transformations by transformations
  - Principles
  - Concrete use

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Simple Copy

```
-- Source metamodel: MMA
class A1 {
    attribute v1 : String;
     attribute v2 : String;
}
```

```
-- Target metamodel: MMB
class B1 {
    attribute v1 : String;
     attribute v2 : String;
}
```

```
module MMAtoMMB;
create OUT : MMB from IN :
MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t : MMB!B1 (
      v1 <- s.v1,
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Structure creation

```
-- Source metamodel: MMA
class A1 {
    attribute v1 : String;
     attribute v2 : String;
}
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
             reference b3 : B3;    }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1andB2andB3 {
  from
    s : MMA!A1
  to
    t1 : MMB!B1 (
      b2 <- t2,
      b3 <- t3
    ),
    t2 : MMB!B2 (
      v1 <- s.v1
    ),
    t3 : MMB!B3 (
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Structure simplification

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2;
           reference a3 : A3;     }
class A2 { attribute v1 : String; }
class A3 { attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 {
      attribute v1 : String;
      attribute v2 : String;
}
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t : MMB!B1 (
      v1 <- s.a2.v1,
      v2 <- s.a3.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Structure simplification (needlessly more complex)

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2;
           reference a3 : A3;     }
class A2 { attribute v1 : String; }
class A3 { attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 {
    attribute v1 : String;
    attribute v2 : String;
}
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from s1 : MMA!A1, s2 : MMA!A2,
    s3 : MMA!A3 (s1.a2 = s2 and s1.a3 = s3)
  to
    t : MMB!B1 (
      v1 <- s.a2.v1,
      v2 <- s.a3.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Traceability: implicit resolution of default elements

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2;
           reference a3 : A3;     }
class A2 { attribute v1 : String; }
class A3 { attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
           reference b3 : B3;     }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t : MMB!B1 (
      b2 <- s.a2,    -- HERE
      b3 <- s.a3     -- HERE
    )
}
```

```
rule A2toB2 {
  from s : MMA!A2
  to     t : MMB!B2 (
    v1 <- s.v1       )
}
rule A3toB3 {
  from s : MMA!A3
  to     t : MMB!B3 (
    v2 <- s.v2       )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Remark: same result, less modular

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2;
           reference a3 : A3;    }
class A2 { attribute v1 : String; }
class A3 { attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
           reference b3 : B3;    }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t1 : MMB!B1 (
      b2 <- t2,
      b3 <- t3
    ),
    t2 : MMB!B2 (
      v1 <- s.a2.v1
    ),
    t3 : MMB!B3 (
      v2 <- s.a3.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Traceability: resolveTemp for additional elements

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2; }
class A2 { attribute v1 : String;
              attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
              reference b3 : B3;     }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from     s : MMA!A1
  to
    t : MMB!B1 (
      b2 <- s.a2,
      b3 <-
        thisModule.resolveTemp(s.a2, 't2')
    )
}
```

```
rule A2toB2andB3 {
  from
    s : MMA!A2
  to
    t1 : MMB!B2 (
      v1 <- s.v1
    ),
    t2 : MMB!B3 (
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Traceability: resolveTemp even for first element

```
-- Source metamodel: MMA
class A1 { reference a2 : A2; }
class A2 { attribute v1 : String;
           attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
           reference b3 : B3;    }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from s : MMA!A1
  to t : MMB!B1 (
      b2 <- -- possible but complex
         thisModule.resolveTemp(s.a2, 't1'),
      b3 <-
         thisModule.resolveTemp(s.a2, 't2')
    )
}
```

```
rule A2toB2andB3 {
  from
    s : MMA!A2
  to
    t1 : MMB!B2 (
      v1 <- s.v1
    ),
    t2 : MMB!B3 (
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - Structure creation revisited with resolveTemp

```
-- Source metamodel: MMA
class A1 {
    attribute v1 : String;
     attribute v2 : String;
}
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
           reference b3 : B3;    }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1andB2andB3 {
  from
    s : MMA!A1
  to
    t1 : MMB!B1 (-- possible but complex
      b2 <- thisModule.resolveTemp(s, 't2'),
      b3 <- thisModule.resolveTemp(s, 't3')
    ),
    t2 : MMB!B2 (
      v1 <- s.v1
    ),
    t3 : MMB!B3 (
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules
  - We have only seen standard rules in default mode so far.

Kind of rule	Number of references to source pattern	Number of times the target pattern gets created	Kind of traceability link created
standard	0 1 n > 1	1 1 1	default or not (using keyword nodefault)
lazy	0 1 n > 1	0 1 n	Not default
unique lazy	0 1 n > 1	0 1 1	Not default

Overview of advanced ATL features

- Implicit resolution, ResolveTemp & lazy rules

Kind of rule	Definition	Reference

standard **default**	**rule** R1 {
	from s : MMA!A1
	to t : MMB!B1 }
		<value of type MMA!A1>
		or
		<collection of MMA!A1>

standard **nodefault**	**nodefault rule** R1 {
	from s : MMA!A1
	to t : MMB!B1 }
		not currently possible

lazy	**lazy rule** R1 {
	from s : MMA!A1
	to t : MMB!B1 }
		thisModule.R1(<value of type MMA!A1>)
		See (1)

unique lazy	**unique lazy rule** R1 {
	from s : MMA!A1
	to t : MMB!B1 }
		thisModule.R1(<value of type MMA!A1>)
		See (1)

(1) For collections: aCollection->collect(e \| thisModule.R1(e))

Overview of advanced ATL features

- Imperative “Do” Section
  - Example: use an incremental variable

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2;
           reference a3 : A3;     }
class A2 { attribute v1 : String; }
class A3 { attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 {
    attribute v1 : String;
    attribute v2 : String;
}
```

```
module MMAtoMMB;

create OUT : MMB from IN : MMA;

helper def: var: Integer = 1;

rule A1toB1 {
  from
    s : MMA!A1
    ...
```

```
  to
    t : MMB!B1 (
      v1 <- s.a2.v1 + thisModule.var.toString(),
      v2 <- s.a3.v2
    )
    do {
       if(thisModule.var <= 10) {
         thisModule.var <- thisModule.var + 1;
       }
    }
}
```

Overview of advanced ATL features

- Called Rules

```
-- Source metamodel: MMA
class A1 { reference  a2 : A2;
           reference a3 : A3;    }
class A2 { attribute v1 : String; }
class A3 { attribute v2 : String; }
```

```
-- Target metamodel: MMB
class B1 { reference b2 : B2;
           reference b3 : B3;    }
class B2 { attribute v1 : String; }
class B3 { attribute v2 : String; }
```

```
module  MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t : MMB!B1 (
      b2 <- thisModule.createB2(s.a2.v1),
      b3 <- thisModule.createB3()
    )
}
```

```
rule createB2(s :  String)  {
  to  t : MMB!B2 (
    v1 <- s           )
  do { t; }
}

rule createB3()  {
  to  t : MMB!B3 (
    v2 <- 'default'   )
  do { t; }
}
```

Overview of advanced ATL features

- Rule Inheritance
  - Help structure transformations and reuse rules and patterns:
    - A child rule matches a subset of what its parent rule matches,
      - ➔ All the bindings of the parent still make sense for the child
    - A child rule specializes target elements of its parent rule:
      - Initialization of existing elements may be improved or changed
      - New elements may be created
    - Syntax:

```
abstract rule R1 {
        -- ...
}
rule R2 extends R1 {
        -- ...
}
```

Overview of advanced ATL features

- Rule Inheritance
  - Copy class inheritance without rule inheritance

```
-- Source metamodel: MMA
class A1 { attribute v1 : String; }
class A2 extends A1 {
   attribute v2 : String;
}
```

```
-- Target metamodel: MMB
class B1 { attribute v1 : String; }
class B2 extends B1 {
     attribute v2 : String;
}
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t : MMB!B1 (
      v1 <- s.v1
    )
}
```

```
rule A2toB2 {
  from
    s : MMA!A2
  to
    t : MMB!B2 (
      v1 <- s.v1,
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Rule Inheritance
  - Copy class inheritance with rule inheritance

```
-- Source metamodel: MMA
class A1 { attribute v1 : String;  }
class A2 extends A1 {
   attribute v2 : String;
}
```

```
-- Target metamodel: MMB
class B1 { attribute v1 : String; }
class B2 extends B1 {
     attribute v2 : String;
}
```

```
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
  from
    s : MMA!A1
  to
    t : MMB!B1 (
      v1 <- s.v1
    )
}
```

```
rule A2toB2 extends A1toB1 {
  from
    s : MMA!A2
  to
    t : MMB!B2 (
      v2 <- s.v2
    )
}
```

Overview of advanced ATL features

- Refining Mode
  - For transformations that need to modify only a small part of a model:
    - Since source models are read-only target models must be created from scratch
    - This can be done by writing copy rules for each elements that are not transformed
      - ➔ This is not very elegant
    - In refining mode, the ATL engine automatically copies unmatched elements
  - The developer only specifies what changes
  - ATL semantics is respected: source models are still read-only.
    - ➔ An (optimized) engine may modify source models in-place but only commit the changes in the end
  - Syntax: replace `from` by `refining`

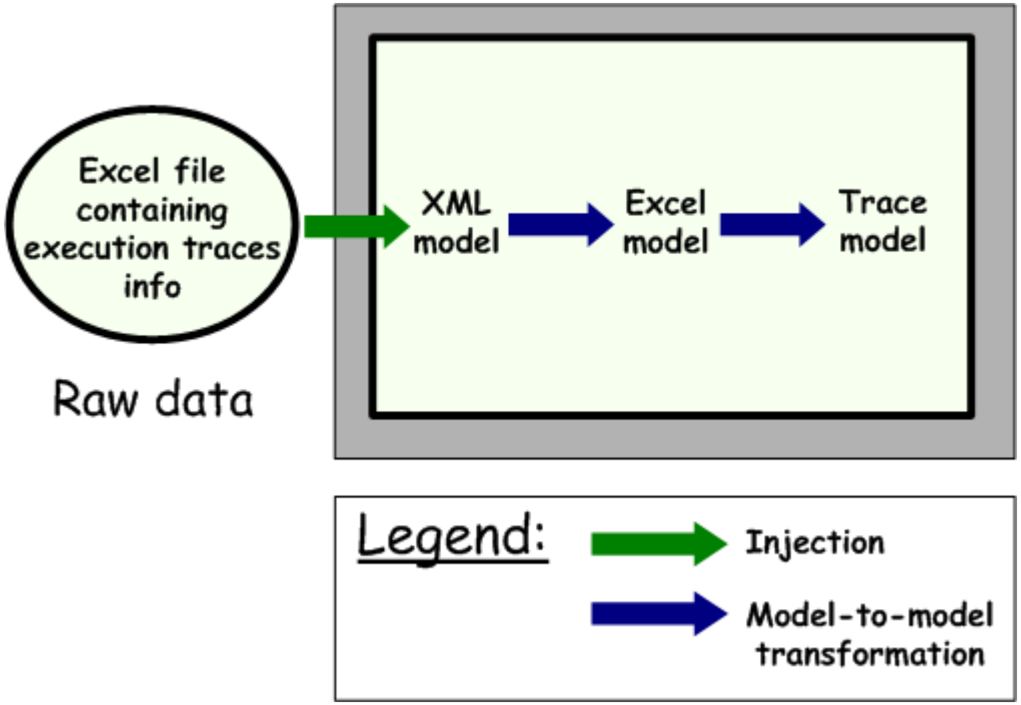
```
module A2A; create OUT : MMA refining IN : MMA;
```

Chaining ATL transformations using ANT scripts

- ATL-Specific ANT tasks
  - Task “am3.loadModel”
    - Loads a model either using the model handler facility or a specific injector
    - Works for terminal models & metamodels (the metametamodel is automatically loaded and can be referred to using “MOF”)
  - Task “am3.saveModel”
    - Saves a model either using the model handler facility or a specific extractor
    - Works for any models (including the metametamodel)
  - Task “am3.atl”
    - Launches the execution of an ATL transformation
      - Required models needs to be previously loaded
  - Detailed description available from the Eclipse Wiki: http://wiki.eclipse.org/AM3_Ant_Tasks

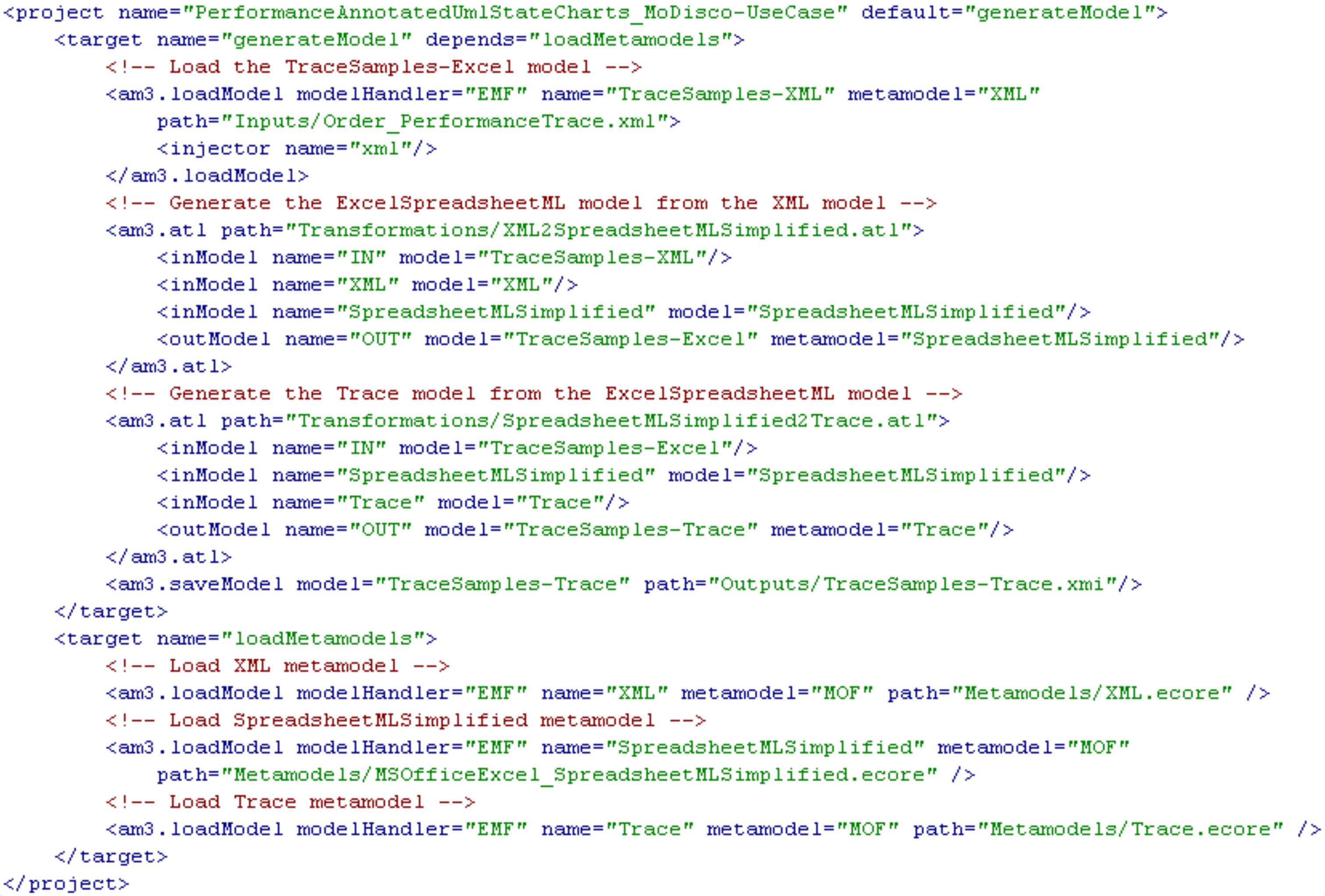
Chaining ATL transformations using ANT scripts

- Concrete Use
  - Part of the Performance-Annotated UML2 State Charts scenario
    http://www.eclipse.org/gmt/modisco/useCases/PerformanceAnnotatedUmlStateCharts/
    - An XML injection (XML model loading)
    - Two ATL transformations (XML-to-Excel and Excel-to-Trace)
    - A model serialisation (Trace model saving)


Excel file containing execution traces info
Raw data
XML model
Excel model
Trace model
Legend:
Injection
Model-to-model transformation

Chaining ATL transformations using ANT scripts

- Concrete Use
  - Corresponding ANT script



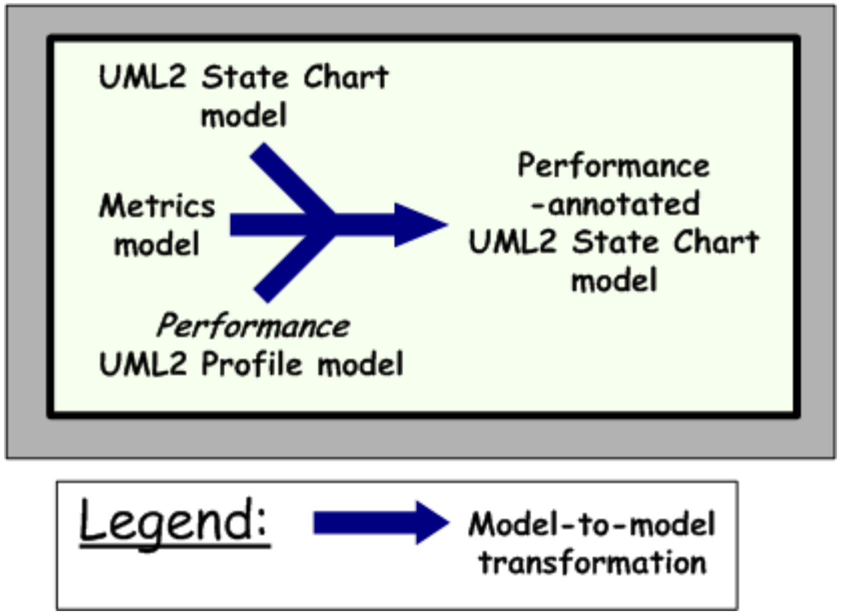
```
<project name="PerformanceAnnotatedUmlStateCharts_MoDisco-UseCase" default="generateModel">
    <target name="generateModel" depends="loadMetamodels">
        <!-- Load the TraceSamples-Excel model -->
        <am3.loadModel modelHandler="EMF" name="TraceSamples-XML" metamodel="XML"
            path="Inputs/Order_PerformanceTrace.xml">
            <injector name="xml"/>
        </am3.loadModel>
        <!-- Generate the ExcelSpreadsheetML model from the XML model -->
        <am3.atl path="Transformations/XML2SpreadsheetMLSimplified.atl">
            <inModel name="IN" model="TraceSamples-XML"/>
            <inModel name="XML" model="XML"/>
            <inModel name="SpreadsheetMLSimplified" model="SpreadsheetMLSimplified"/>
            <outModel name="OUT" model="TraceSamples-Excel" metamodel="SpreadsheetMLSimplified"/>
        </am3.atl>
        <!-- Generate the Trace model from the ExcelSpreadsheetML model -->
        <am3.atl path="Transformations/SpreadsheetMLSimplified2Trace.atl">
            <inModel name="IN" model="TraceSamples-Excel"/>
            <inModel name="SpreadsheetMLSimplified" model="SpreadsheetMLSimplified"/>
            <inModel name="Trace" model="Trace"/>
            <outModel name="OUT" model="TraceSamples-Trace" metamodel="Trace"/>
        </am3.atl>
        <am3.saveModel model="TraceSamples-Trace" path="Outputs/TraceSamples-Trace.xmi"/>
    </target>
    <target name="loadMetamodels">
        <!-- Load XML metamodel -->
        <am3.loadModel modelHandler="EMF" name="XML" metamodel="MOF" path="Metamodels/XML.ecore" />
        <!-- Load SpreadsheetMLSimplified metamodel -->
        <am3.loadModel modelHandler="EMF" name="SpreadsheetMLSimplified" metamodel="MOF"
            path="Metamodels/MSOfficeExcel_SpreadsheetMLSimplified.ecore" />
        <!-- Load Trace metamodel -->
        <am3.loadModel modelHandler="EMF" name="Trace" metamodel="MOF" path="Metamodels/Trace.ecore" />
    </target>
</project>
```

Handling UML2 models & profiles with ATL

- The UML24ATL model handler
  - Extends the standard EMF model handler in order to support Eclipse-MDT UML2 (http://www.eclipse.org/modeling/mdt/?project=uml2)
  - Provides a set of UML2 specific primitives (cf. the Eclipse-MDT UML2 project documentation)
  - Some useful primitives for profile application
    - "applyProfile" → applies the profile given as parameter on the selected UML2 model
    - "applyStereotype" → applies the stereotype given as parameter on the selected UML2 model element
    - "setValue" → sets the value given as parameter to the given property of the specified stereotype

Handling UML2 models & profiles with ATL

- Concrete Use
  - Part of the Performance-Annotated UML2 State Charts scenario
    http://www.eclipse.org/gmt/modisco/useCases/PerformanceAnnotatedUmlStateCharts/
    - Application of a UML2 profile (about Performance) to a UML2 model (state chart) using data coming from an additional model (previously computed metrics)



UML2 State Chart model

Metrics model

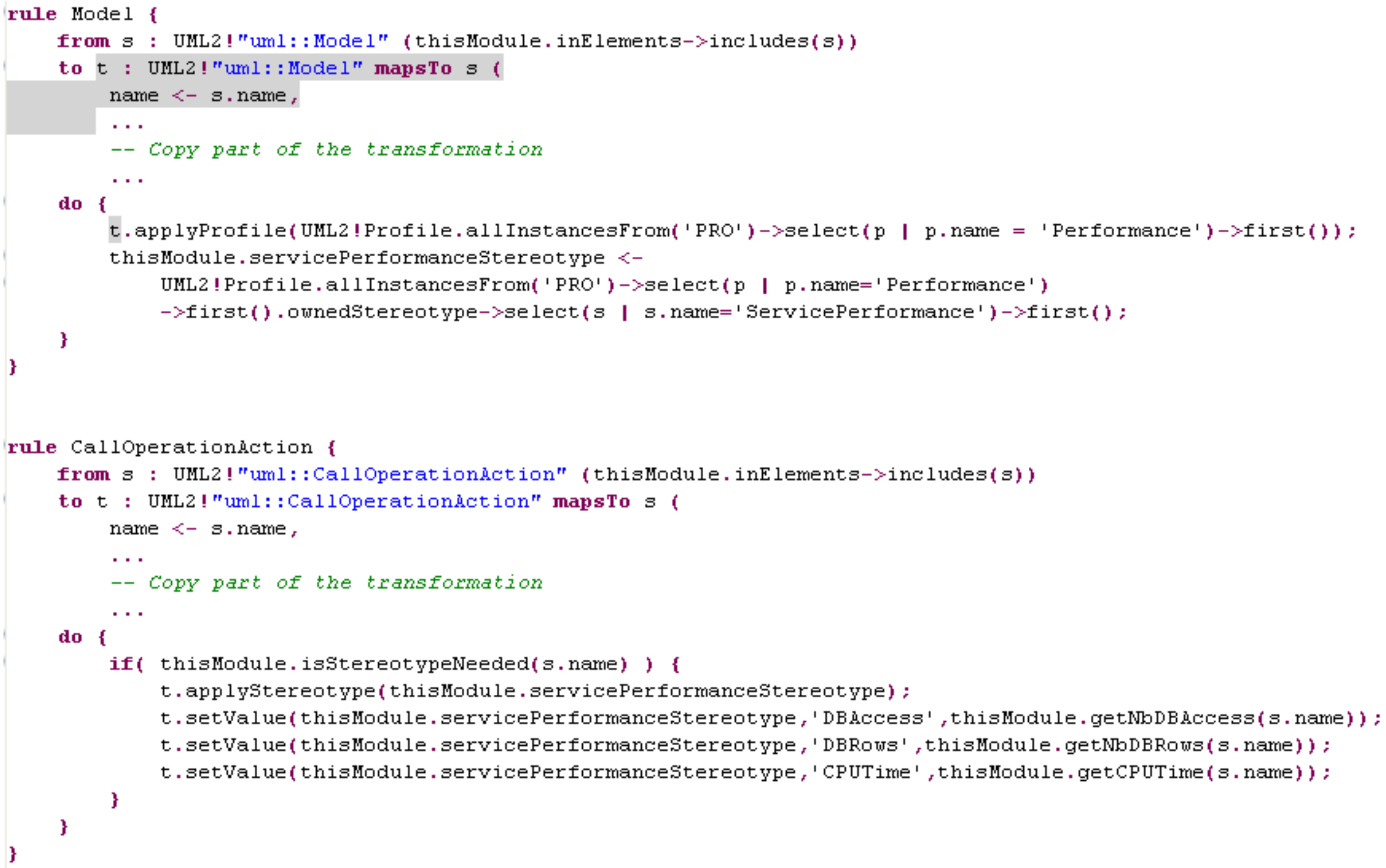
Performance UML2 Profile model

Performance-annotated UML2 State Chart model

Legend: → Model-to-model transformation

Handling UML2 models & profiles with ATL

- Concrete Use
  - Corresponding ATL transformation (excerpt)



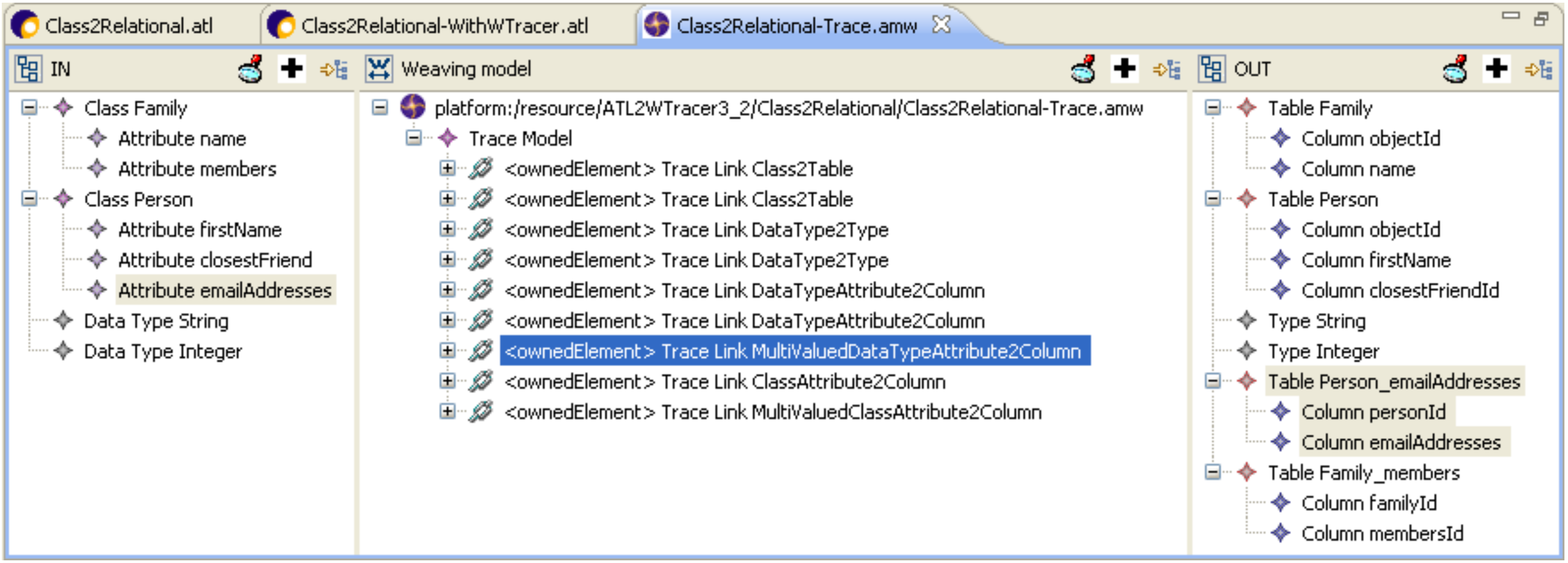
```
rule Model {
    from s : UML2!"uml::Model" (thisModule.inElements->includes(s))
    to t : UML2!"uml::Model" mapsTo s (
        name <- s.name,
        ...
        -- Copy part of the transformation
        ...
    do {
        t.applyProfile(UML2!Profile.allInstancesFrom('PRO')->select(p | p.name = 'Performance')->first());
        thisModule.servicePerformanceStereotype <-
            UML2!Profile.allInstancesFrom('PRO')->select(p | p.name='Performance')
            ->first().ownedStereotype->select(s | s.name='ServicePerformance')->first();
    }
}


rule CallOperationAction {
    from s : UML2!"uml::CallOperationAction" (thisModule.inElements->includes(s))
    to t : UML2!"uml::CallOperationAction" mapsTo s (
        name <- s.name,
        ...
        -- Copy part of the transformation
        ...
    do {
        if( thisModule.isStereotypeNeeded(s.name) ) {
            t.applyStereotype(thisModule.servicePerformanceStereotype);
            t.setValue(thisModule.servicePerformanceStereotype,'DBAccess',thisModule.getNbDBAccess(s.name));
            t.setValue(thisModule.servicePerformanceStereotype,'DBRows',thisModule.getNbDBRows(s.name));
            t.setValue(thisModule.servicePerformanceStereotype,'CPUTime',thisModule.getCPUTime(s.name));
        }
    }
}
```

Handling weaving models with ATL & AMW

- The ATL4AMW model handler
  - Extends the standard EMF model handler in order to support Eclipse-GMT AMW (http://www.eclipse.org/gmt/amw/)
  - Provides a set of model weaving-specific primitives (details provided here: http://wiki.eclipse.org/AMW_Model_Handler)
  - Generating weaving models (from woven models):
    - "generateModelRef" → produces a reference to a given woven model
    - "getElementID" & "getElementIDbyRefType" → provides a reference to a given woven model element
  - Use existing weaving models:
    - "getReferredElement" → allows retrieving a given model element (from another model) linked to a model element thanks to a weaving link

Handling weaving models with ATL & AMW

- Concrete Use
  - Application of the Traceability AMW use case http://www.eclipse.org/gmt/amw/usecases/traceability/ on the "Class To Relational" basic transformation example
  - Objective: generate a traceability model between the source and target models of the transformation by "augmented" it



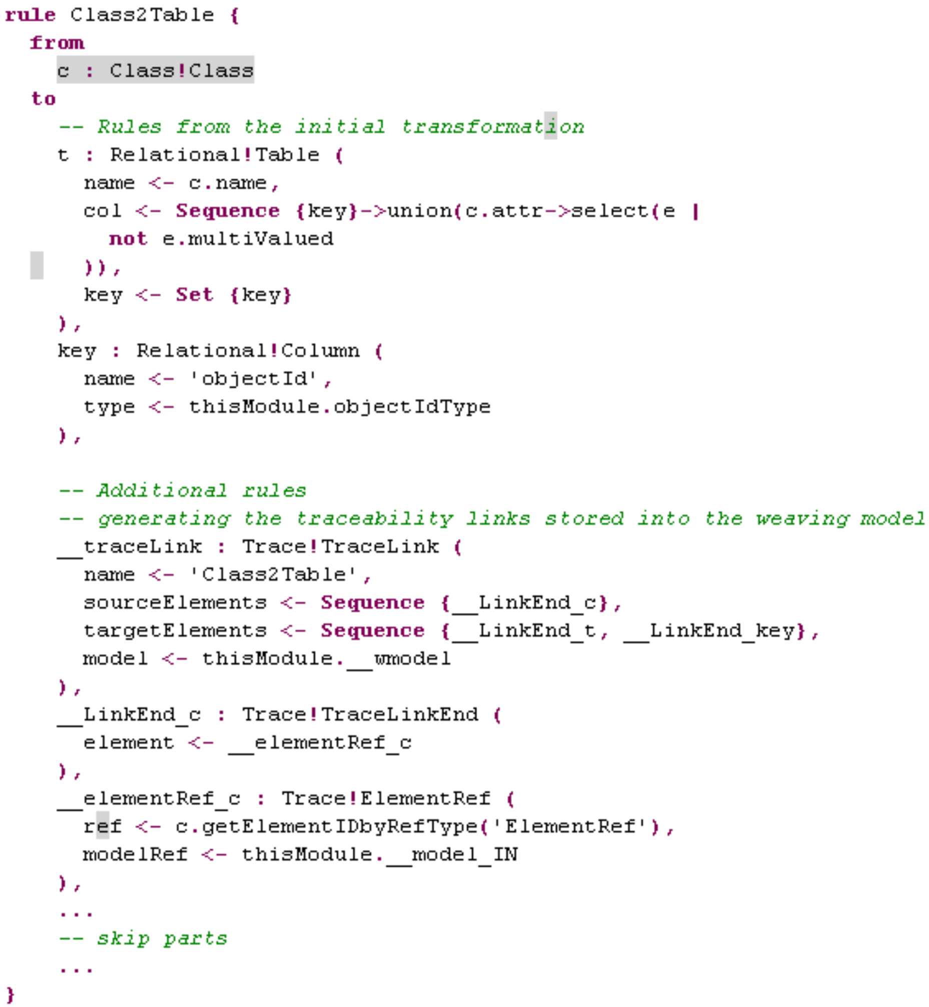
IN:
- Class Family
  - Attribute name
  - Attribute members
- Class Person
  - Attribute firstName
  - Attribute closestFriend
  - Attribute emailAddresses
- Data Type String
- Data Type Integer

Weaving model:
- platform:/resource/ATL2WTracer3_2/Class2Relational/Class2Relational-Trace.amw
  - Trace Model
    - <ownedElement> Trace Link Class2Table
    - <ownedElement> Trace Link Class2Table
    - <ownedElement> Trace Link DataType2Type
    - <ownedElement> Trace Link DataType2Type
    - <ownedElement> Trace Link DataTypeAttribute2Column
    - <ownedElement> Trace Link DataTypeAttribute2Column
    - <ownedElement> Trace Link MultiValuedDataTypeAttribute2Column
    - <ownedElement> Trace Link ClassAttribute2Column
    - <ownedElement> Trace Link MultiValuedClassAttribute2Column

OUT:
- Table Family
  - Column objectId
  - Column name
- Table Person
  - Column objectId
  - Column firstName
  - Column closestFriendId
- Type String
- Type Integer
- Table Person_emailAddresses
  - Column personId
  - Column emailAddresses
- Table Family_members
  - Column familyId
  - Column membersId

Handling weaving models with ATL & AMW

- Concrete Use
  - Corresponding “augmented” ATL transformation (excerpt)



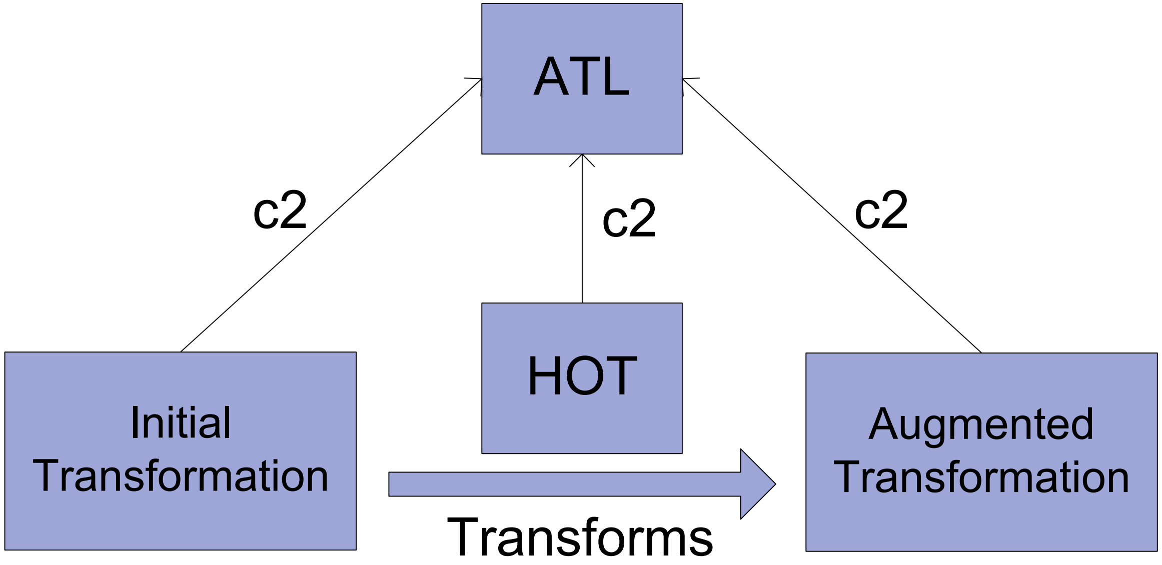
```
rule Class2Table {
  from
    c : Class!Class
  to
    -- Rules from the initial transformation
    t : Relational!Table (
      name <- c.name,
      col <- Sequence {key}->union(c.attr->select(e |
        not e.multiValued
      )),
      key <- Set {key}
    ),
    key : Relational!Column (
      name <- 'objectId',
      type <- thisModule.objectIdType
    ),

    -- Additional rules
    -- generating the traceability links stored into the weaving model
    __traceLink : Trace!TraceLink (
      name <- 'Class2Table',
      sourceElements <- Sequence {__LinkEnd_c},
      targetElements <- Sequence {__LinkEnd_t, __LinkEnd_key},
      model <- thisModule.__wmodel
    ),
    __LinkEnd_c : Trace!TraceLinkEnd (
      element <- __elementRef_c
    ),
    __elementRef_c : Trace!ElementRef (
      ref <- c.getElementIDbyRefType('ElementRef'),
      modelRef <- thisModule.__model_IN
    ),
    ...
    -- skip parts
    ...
}
```

High-Order Transformations (HOT): Handling of transformations by transformations

- Principles
  - A transformation is a model
    - An ATL transformation is a model which conforms to the ATL metamodel
  - A transformation can be taken as input of another transformation
  - A transformation can be generated as output of another transformation
  - Such a transformation is called a High-Order Transformation or HOT
  - A lot of applications of such a technique for automation of transformations (often used in combination with model weaving techniques):
    - Traceability, Model evolution, Interoperability, etc

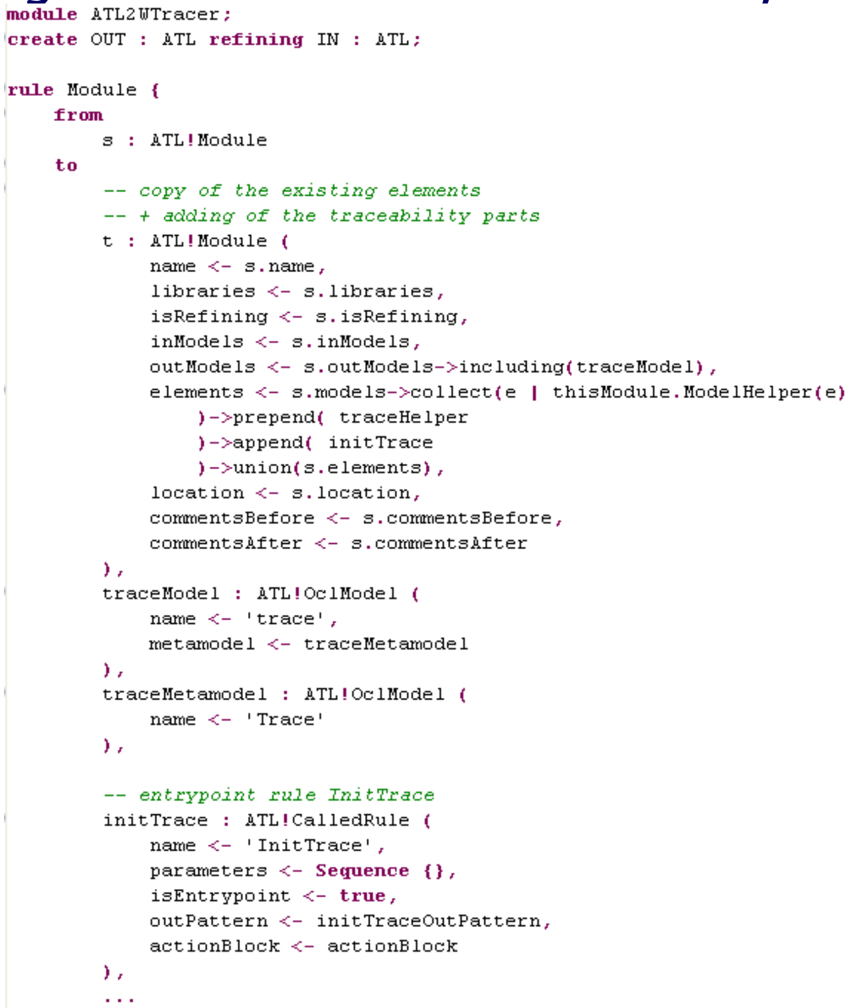
High-Order Transformations (HOT): Handling of transformations by transformations

- Concrete Use
  - An extension of the Traceability AMW use case http://www.eclipse.org/gmt/amw/usecases/traceability/
  - Objective: automated generation using a HOT of an "augmented" transformation (traceability) from the initial one



High-Order Transformations (HOT): Handling of transformations by transformations

- Concrete Use
  - Corresponding HOT which adds traceability support (excerpt)



```
module ATL2WTracer;
create OUT : ATL refining IN : ATL;

rule Module {
    from
        s : ATL!Module
    to
        -- copy of the existing elements
        -- + adding of the traceability parts
        t : ATL!Module (
            name <- s.name,
            libraries <- s.libraries,
            isRefining <- s.isRefining,
            inModels <- s.inModels,
            outModels <- s.outModels->including(traceModel),
            elements <- s.models->collect(e | thisModule.ModelHelper(e)
                )->prepend( traceHelper
                )->append( initTrace
                )->union(s.elements),
            location <- s.location,
            commentsBefore <- s.commentsBefore,
            commentsAfter <- s.commentsAfter
        ),
        traceModel : ATL!OclModel (
            name <- 'trace',
            metamodel <- traceMetamodel
        ),
        traceMetamodel : ATL!OclModel (
            name <- 'Trace'
        ),

        -- entrypoint rule InitTrace
        initTrace : ATL!CalledRule (
            name <- 'InitTrace',
            parameters <- Sequence {},
            isEntrypoint <- true,
            outPattern <- initTraceOutPattern,
            actionBlock <- actionBlock
        ),
        ...
```

References

- ATL Home page
  - http://www.eclipse.org/m2m/atl/
- ATL Documentation page
  - http://www.eclipse.org/m2m/atl/doc/
- ATL Use Cases page
  - http://www.eclipse.org/m2m/atl/usecases/
- ATL Newsgroup (use [ATL] tag)
  - news://news.eclipse.org/eclipse.modeling.m2m
- ATL on Eclipse Wiki
  - http://wiki.eclipse.org/index.php/ATL