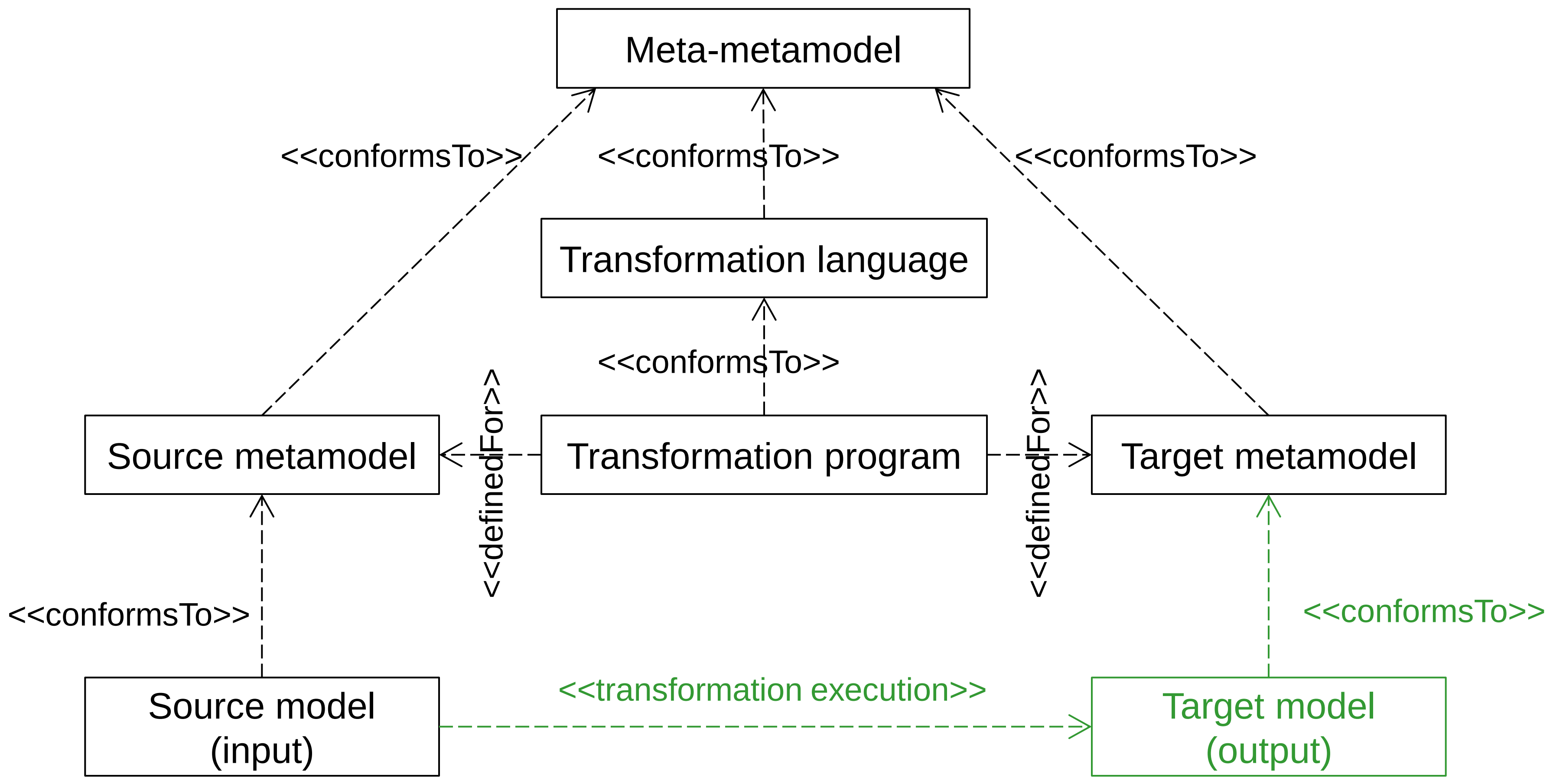
Model-to-model transformation with ATL (Atlas Transformation Language)

Juan de Lara, Elena Gómez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es
Computer Science Department
Universidad Autónoma de Madrid

Masters: I2TIC and Formal methods

Model-to-model transformation

Schema



ATL
Overview

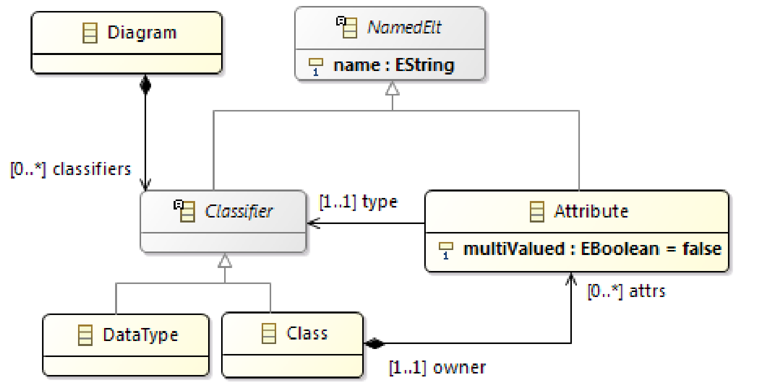
- Model-to-model transformation language
  - source models are read-only
  - target models are write-only

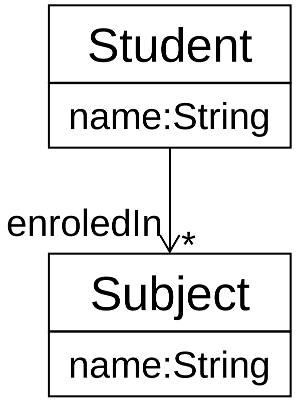
- Hybrid language
  - declarative part based on rules (**recommended**)
    - matching of source pattern
    - creation of target pattern
  - imperative part: action blocks (statements)

Model-to-model transformation

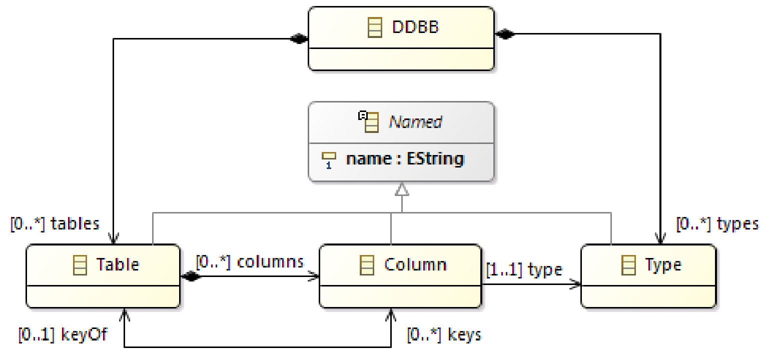
Example: Class to Relational

Class diagrams:





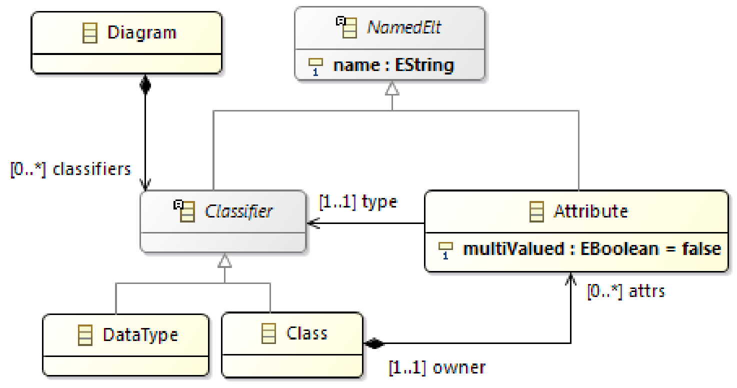
Relational ddbb:



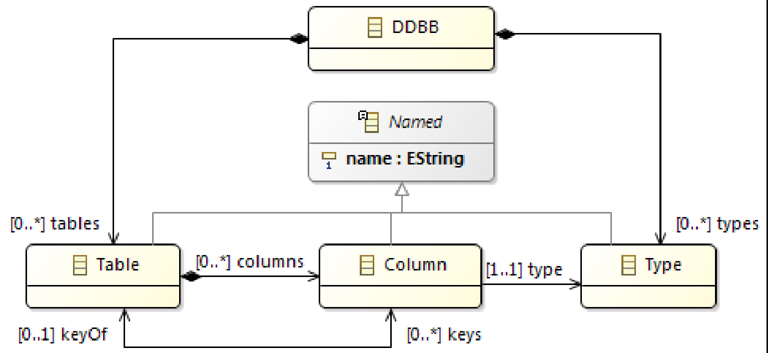
?

Model-to-model transformation
Example: Class to Relational

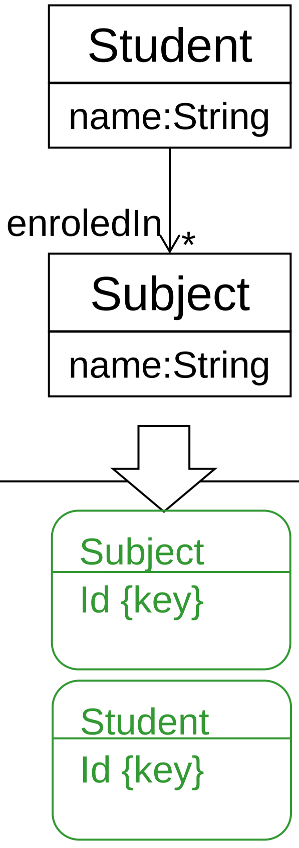
Class diagrams:



Relational ddbb:

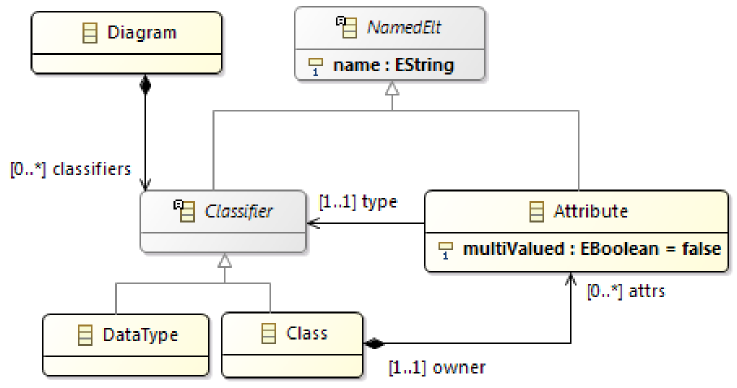


(1) A table is created for each class, with the same name as the table, and a column ID that is key of the table

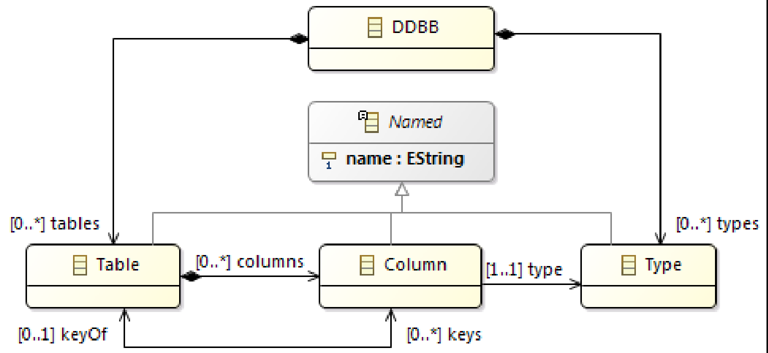


Model-to-model transformation
Example: Class to Relational

Class diagrams:

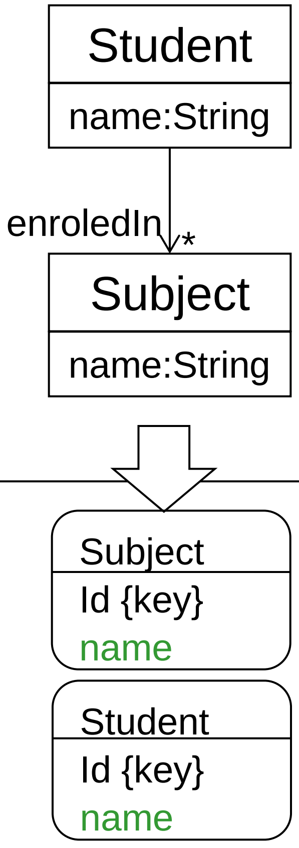


Relational ddbb:



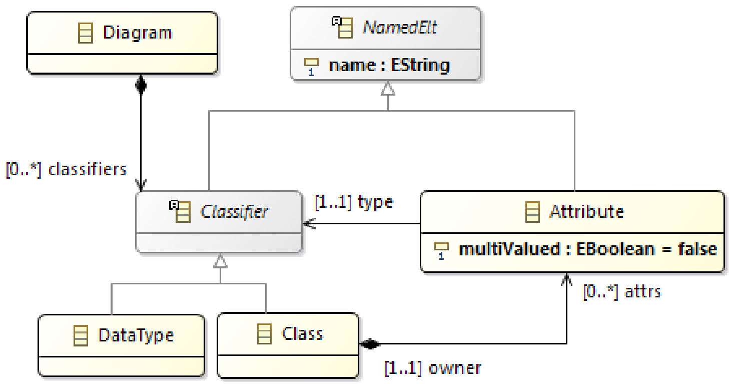
(1) A table is created for each class, with the same name as the table, and a column ID that is key of the table

(2) A column is created for each single-valued attribute

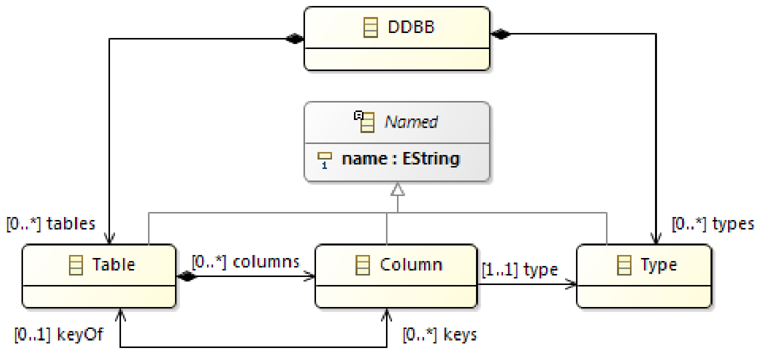


Model-to-model transformation
Example: Class to Relational

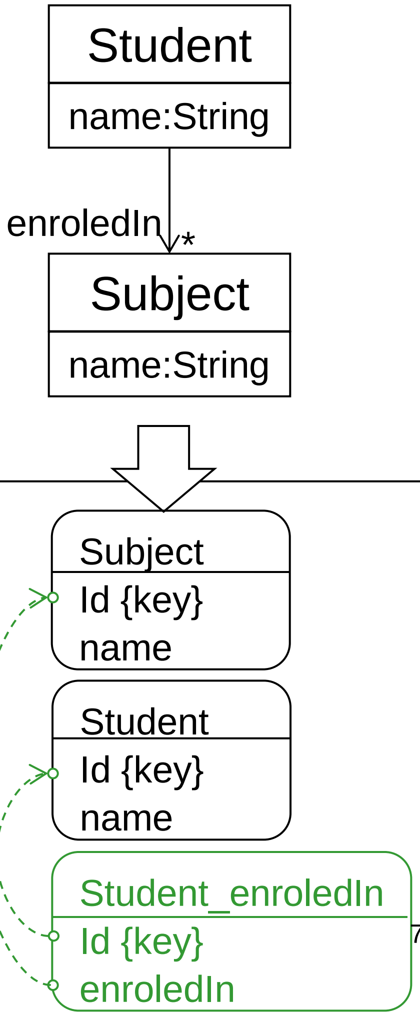
Class diagrams:



Relational ddbb:



(1) A table is created for each class, with the same name as the table, and a column ID that is key of the table
(2) A column is created for each single-valued attribute
(3) A table with two columns is created for each multi-valued attribute



ATL transformation
General transformation structure

```
-- header (mandatory)
module class2relational;
create OUT : Relational from IN : Class;


-- import section
...

-- helpers and transformation rules
...
```

logical name of
source model : source metamodel

logical name of
target model : target metamodel

ATL rules
Matched rules

```
-- a table is created for each class
rule Class2Table {
  from c : Class!Class
  to   t : Relational!Table
}
```

source pattern

target pattern

A **matched rule** is executed for each match of the source pattern in the source model.

The source/target patterns can have several objects.

ATL rules
Bindings

```
-- a table is created for each class
rule Class2Table {
  from c : Class!Class
  to   t : Relational!Table (
         -- the table name is equals to the class name
         name <- c.name
       )
}
```

A **binding** initializes the value of a target feature.

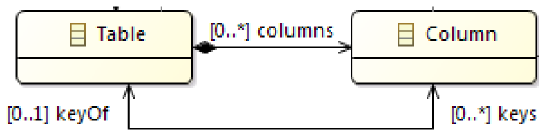
- Binding lhs: target feature
- Binding rhs: ocl expression over source model

ATL rules
Creating several target objects

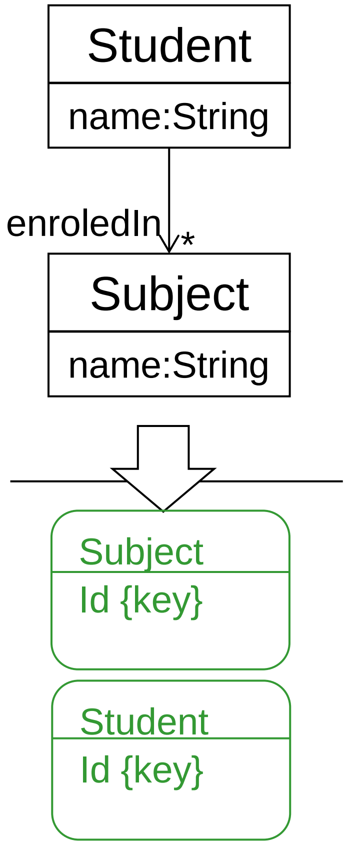
```
-- a table is created for each class
rule Class2Table {
  from c : Class!Class
  to   t : Relational!Table (
         -- the table name is equals to the class name
         name <- c.name,
         -- the table has a column ID, which is key
         columns <- Set{key},
         keys <- Set{key}
       ),
       key : Relational!Column (
          name <- 'Id'
       )
}
```

The binding can refer to other objects created by the rule

We will refine this rule later…



(excerpt of target meta-model)



(example)

ATL rules
Guards

```
-- a column is created for each single-valued attribute
rule SingleValuedAttribute2Column {
  from a : Class!Attribute (not a.multiValued)
  to   c : Relational!Column (
           name <- a.name
         )
}
```

The source pattern can define a **guard**, expressed in OCL. The rule will only be applied to the input objects that fulfil the guard.

At runtime, objects in the source model can be **matched by at most one rule**. Otherwise, there will be a runtime error.

If two rules have the same type, check carefully that they have disjoint guards.

ATL rules
Binding resolution

```
-- a table is created for each class
rule Class2Table {
  from c : Class!Class
  to   t : Relational!Table (
         ...
         -- the table has a column ID, which is key AND
         -- single-valued attributes are columns of the table
         columns <- Set{key}
         ...
       ),
       key : Relational!Column ...
}

-- a column is created for each single-valued attribute
rule SingleValuedAttribute2Column {
  from a : Class!Attribute ...
  to   c : Relational!Column ...
}
```

We need to refine rule Class2Table, to store in the created table the columns created from single-valued attributes

ATL rules
Binding resolution

```
-- a table is created for each class
rule Class2Table {
  from c : Class!Class
  to   t : Relational!Table (
         ...
         -- the table has a column ID, which is key AND
         -- single-valued attributes are columns of the table
         columns <- Set{key}->union(c.attrs->select(a | not a.multiValued))
         ...
       ),
       key : Relational!Column ...
}

-- a column is created for each single-va
rule SingleValuedAttribute2Column {
  from a : Class!Attribute ...
  to   c : Relational!Column ...
}
```

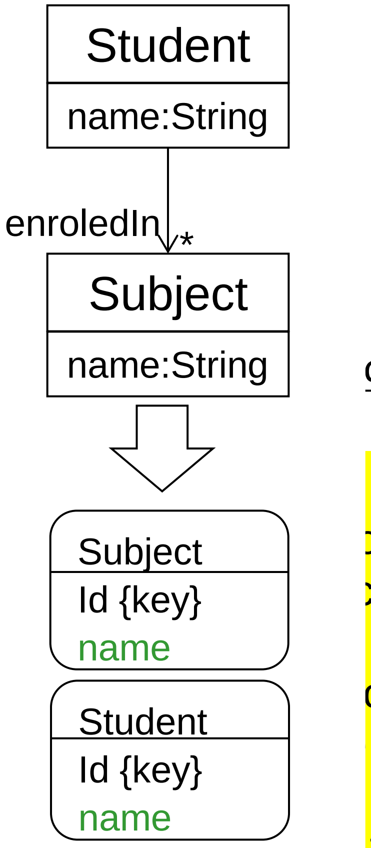
Binding resolution:
(1) The OCL expression is evaluated on the source model, and yields source objects.
(2) The engine obtains the target objects into which the source objects have been transformed.
(3) Those target objects are assigned to the target feature.

ATL rules

Binding resolution

```
-- a table is created for each class
rule Class2Table {
  from c : Class!Class
  to   t : Relational!Table (
         ...
         -- the table has a column ID, which is key AND
         -- single-valued attributes are columns of the
         columns <- Set{key}->union(c.attrs->select(a |                    d))
         ...
       ),
       key : Relational!Column ...
}

-- a column is created for each single-va
rule SingleValuedAttribute2Column {
  from a : Class!Attribute ...
  to   c : Relational!Column ...
}
```



Binding resoluti

(1) The OCL exp   on
the source m   ce
objects.

(2) The engine o   cts
into which the
been transfor

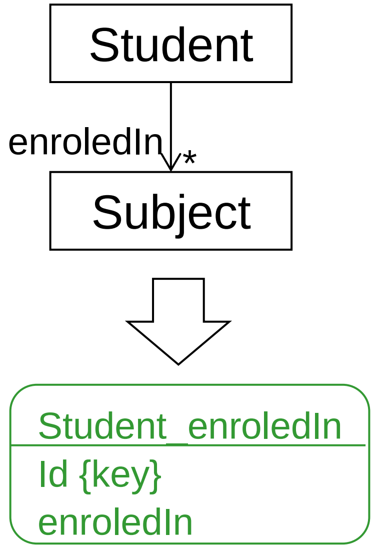
(3) Those target   to
the target feature.

ATL rules
Helpers

```
-- a table with two columns is created for each multi-valued attribute
rule MultiValuedAttribute2Table {
  from a : Class!Attribute (a.multiValued)
  to   t : Relational!Table (
         name <- a.composeColumnName(),
         columns <- Sequence{id, value}
       ),
       id : Relational!Column ( name <- 'Id' ),
       value : Relational!Column ( name <- a.name )
}

helper context Class!Attribute def :
  composeColumnName() : String =
  self.owner.name + '_' + self.name;
```

Helpers are auxiliary functions.
Their body is an OCL expression.



ATL rules

Lazy rules

```
-- a table with two columns is created for each multi-valued attribute
rule MultiValuedAttribute2Table {
  from a : Class!Attribute (a.multiValued)
  to   t : Relational!Table (
         name <- a.composeColumnName(),
         columns <- Set{thisModule.createColumnId(a), value}
       ),
       ~~id : Relational!Column ( name <- 'Id' ),~~
       value : Relational!Column ( name <- a.name )
}

lazy rule createColumnId {
  from a: Class!Attribute
  to   c: Relational!Column (
         name <- 'Id'
       )
}
```

Lazy rules must be explicitly invoked. Applied every time they are invoked.

Variant: **unique lazy rules** are applied at most once per match.

ATL
Execution semantics

- First, **matched rules** are executed
  - applied for each object of the type in “`from`”
  - a guard expression permits filtering
  - create trace between source and target objects
- Next, **lazy rules** are executed if invoked explicitly
  - `thisModule.ruleName(srcObjects)`
- Finally, **binding resolution** (`feat <- srcObject`)
  - lookup a trace t such that t.source = srcObject
  - assign t.target to feat

ATL
Additional resources

- User guide of ATL (quite complete):
  https://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language
- ATL zoo (repository of transformations):
  https://www.eclipse.org/atl/atlTransformations
- Analizador estático de ATL (anATLyzer):
  https://anatlyzer.github.io/

ATL development environment

Creating an ATL project

File / New / Project... / ATL / ATL Project

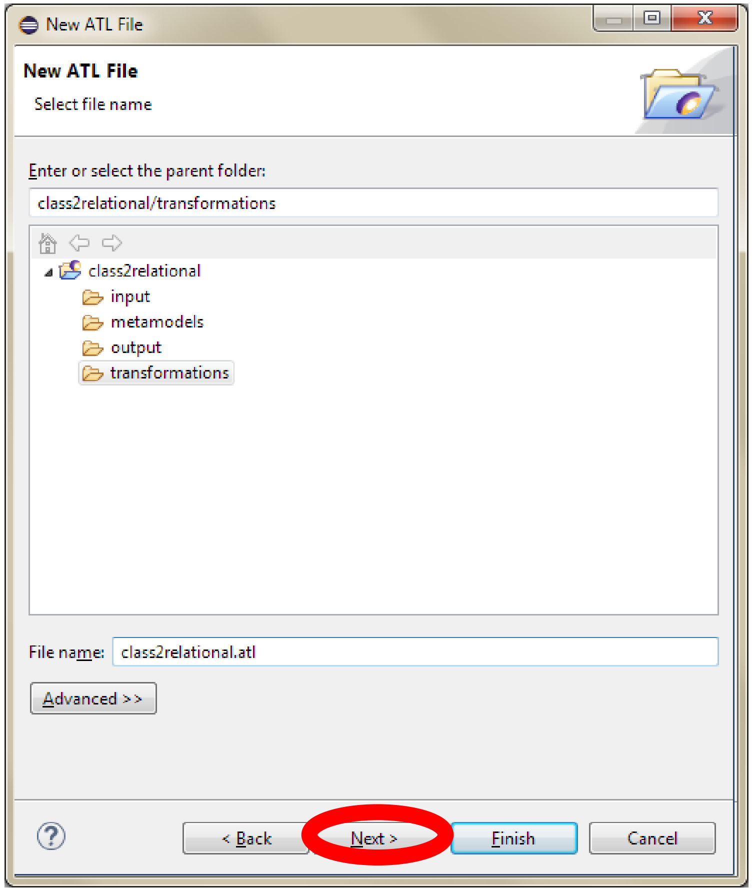
- This creates an empty project
- We recommend creating some folders like:
  - **transformations**: to store .atl files
  - **metamodels**: to store .ecore files not available from other projects
  - **input**: to store input test models
  - **output**: to store generated output models

ATL development environment

Creating an ATL transformation (i)

File / New / Other... / ATL / ATL File

- Transformation file name



Enter or select the parent folder:
class2relational/transformations

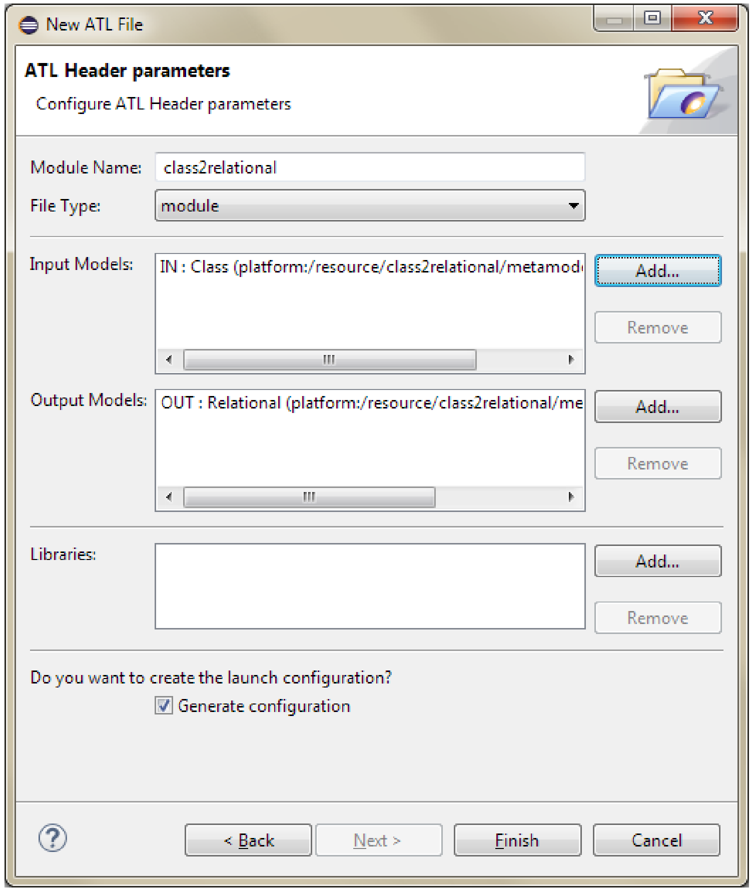
File name: class2relational.atl

ATL development environment

Creating an ATL transformation (ii)

File / New / Other... / ATL / ATL File

- Configure transformation:
  - File type: “module” to select a transformation module (instead of a library of helpers)
  - Input models / output models of the transformation
  - Libraries of helpers used by the transformation



Module Name: class2relational

File Type: module

Input Models: IN : Class (platform:/resource/class2relational/metamod

Output Models: OUT : Relational (platform:/resource/class2relational/me

Libraries:

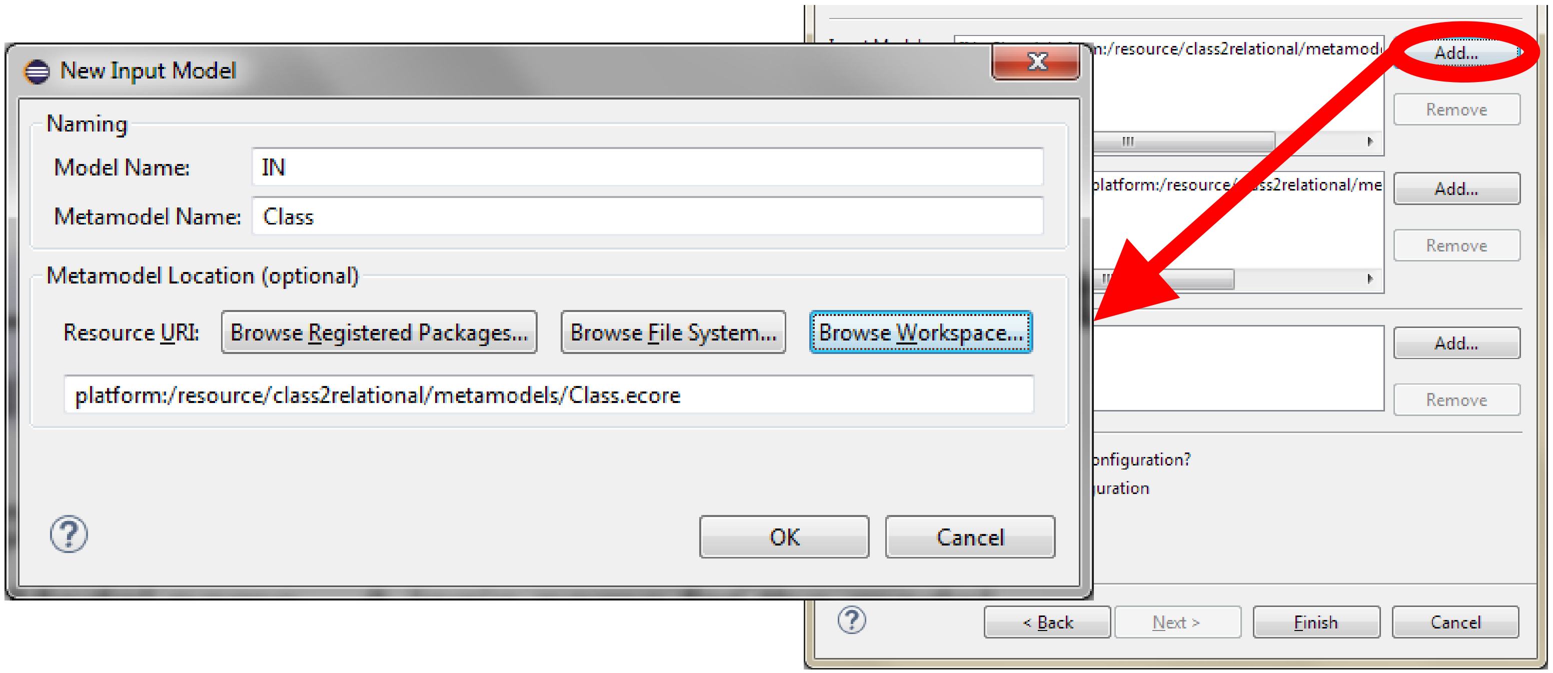
Do you want to create the launch configuration?

[✓] Generate configuration

ATL development environment

Creating an ATL transformation (iii)

- Specify input models/output models
  - Model name: logic name of the model
  - Metamodel name: logic name of the meta-model
  - Resource URI: actual meta-model (registered or ecore)



New Input Model

Naming

Model Name:	IN
Metamodel Name:	Class

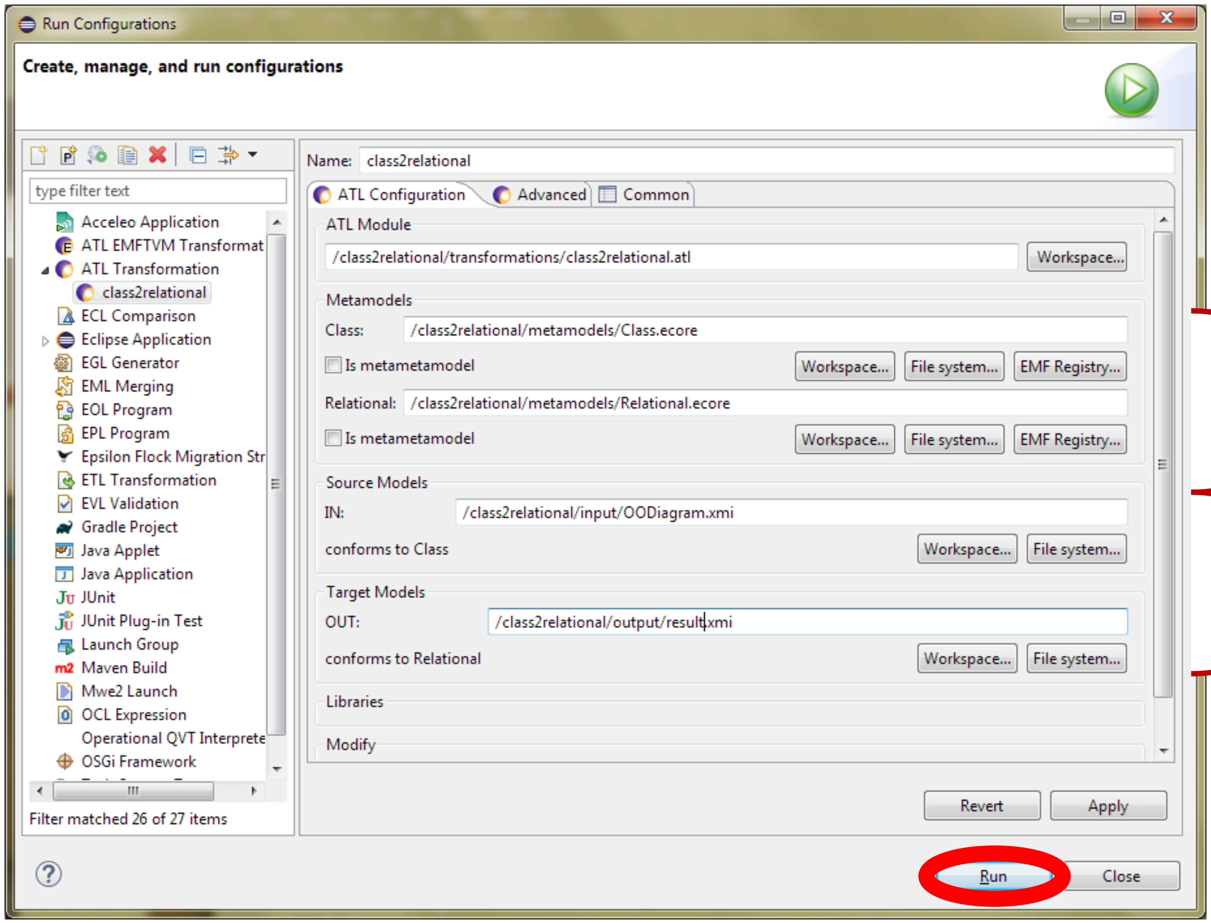
Metamodel Location (optional)

Resource URI: platform:/resource/class2relational/metamodels/Class.ecore

ATL development environment

Running an ATL transformation

Run / Run configurations... / ATL transformation



Run Configurations

Create, manage, and run configurations

type filter text

- Acceleo Application
- ATL EMFTVM Transformat
- ATL Transformation
  - class2relational
- ECL Comparison
- Eclipse Application
- EGL Generator
- EML Merging
- EOL Program
- EPL Program
- Epsilon Flock Migration Str
- ETL Transformation
- EVL Validation
- Gradle Project
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Launch Group
- Maven Build
- Mwe2 Launch
- OCL Expression
- Operational QVT Interprete
- OSGi Framework

Filter matched 26 of 27 items

Name: class2relational

ATL Configuration | Advanced | Common

ATL Module

/class2relational/transformations/class2relational.atl

Metamodels

Class: /class2relational/metamodels/Class.ecore

☐ Is metametamodel

Relational: /class2relational/metamodels/Relational.ecore

☐ Is metametamodel

Source Models

IN: /class2relational/input/OODiagram.xmi

conforms to Class

Target Models

OUT: /class2relational/output/result.xmi

conforms to Relational

Libraries

Modify

make sure metamodels are properly set

set source and target models

execution errors are reported in the console

Others things that may be useful
Epsilon development tools

- First, install Epsilon Core + Epsilon Development Tools for EMF
    - **To register ecore metamodel**: right-click on ecore, and select Register Metamodel
    - **To create metamodel instance** without starting a new Eclipse app: select New File / Epsilon / EMF Model, and provide model file name, metamodel URI, and root instance type