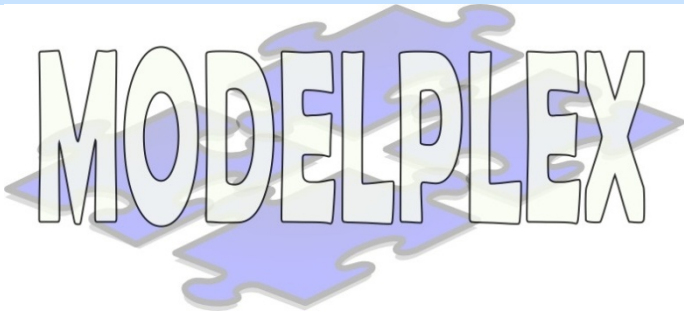
Introduction To Model-to-Model Transformation

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

- The Eclipse-M2M ATL Component
  - Overall presentation
  - How to get the ATL plugins
- M2M Transformation Principles
  - Main concepts & schema of principles
- M2M with ATL
  - Mapping of the M2M principles within the context of ATL
  - Overview of the language
- Writing a First Transformation with ATL
  - ATL Perspective
  - ATL module
  - Simple matched rules
  - Helpers
  - Running the transformation (launch configuration)

The Eclipse-M2M ATL Component

- ATL: a key part of the Eclipse-M2M project (Modeling)

The Eclipse-M2M ATL Component

- ATL homepage: http://www.eclipse.org/m2m/atl/

The Eclipse-M2M ATL Component

- Available resources:
  - Use cases → complete transformation scenarios covering many different domains of application
  - Basic examples → very first transformation examples which are interesting when starting with ATL (for beginners)
  - ATL Transformations → ATL Transformation Zoo which gathers a hundred of various and varied transformations implemented using ATL
  - Download → different binary builds of ATL available and also additional information for using the ATL update site

The Eclipse-M2M ATL Component

- Available resources:
  - Documentation → various kinds of ATL documents including a reference manual, a user manual, installation instructions, etc
  - Publications → non-exhaustive list of papers presenting different works involving or using (directly or indirectly) ATL
  - Wiki → an open section dedicated to ATL on the Eclipse Wiki which allows the community to consult or/and add information about ATL
  - Newsgroup → a link to the Eclipse newsgroup dedicated to the M2M project components (posts concerning ATL are prefixed with the [ATL] tag)

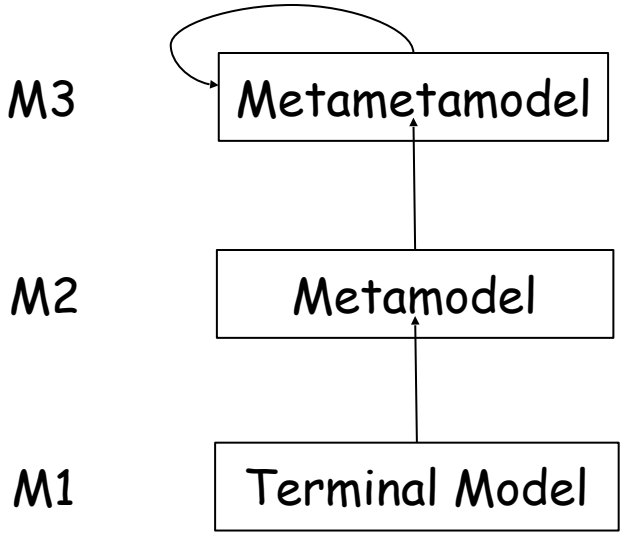
The Eclipse-M2M ATL Component

- How to get the plugins:
  - Download the latest binary builds (frequently updated): http://www.eclipse.org/modeling/m2m/downloads/?project=atl
  - Use the M2M update site (M2M ATL SDK): http://www.eclipse.org/modeling/m2m/updates/
  - Install ATL sources from CVS (stable HEAD): http://wiki.eclipse.org/ATL/How_Install_ATL_From_CVS/
  - Install ATL sources from CVS (development branch): http://wiki.eclipse.org/ATL/How_Install_ATL_(Dev)_From_CVS

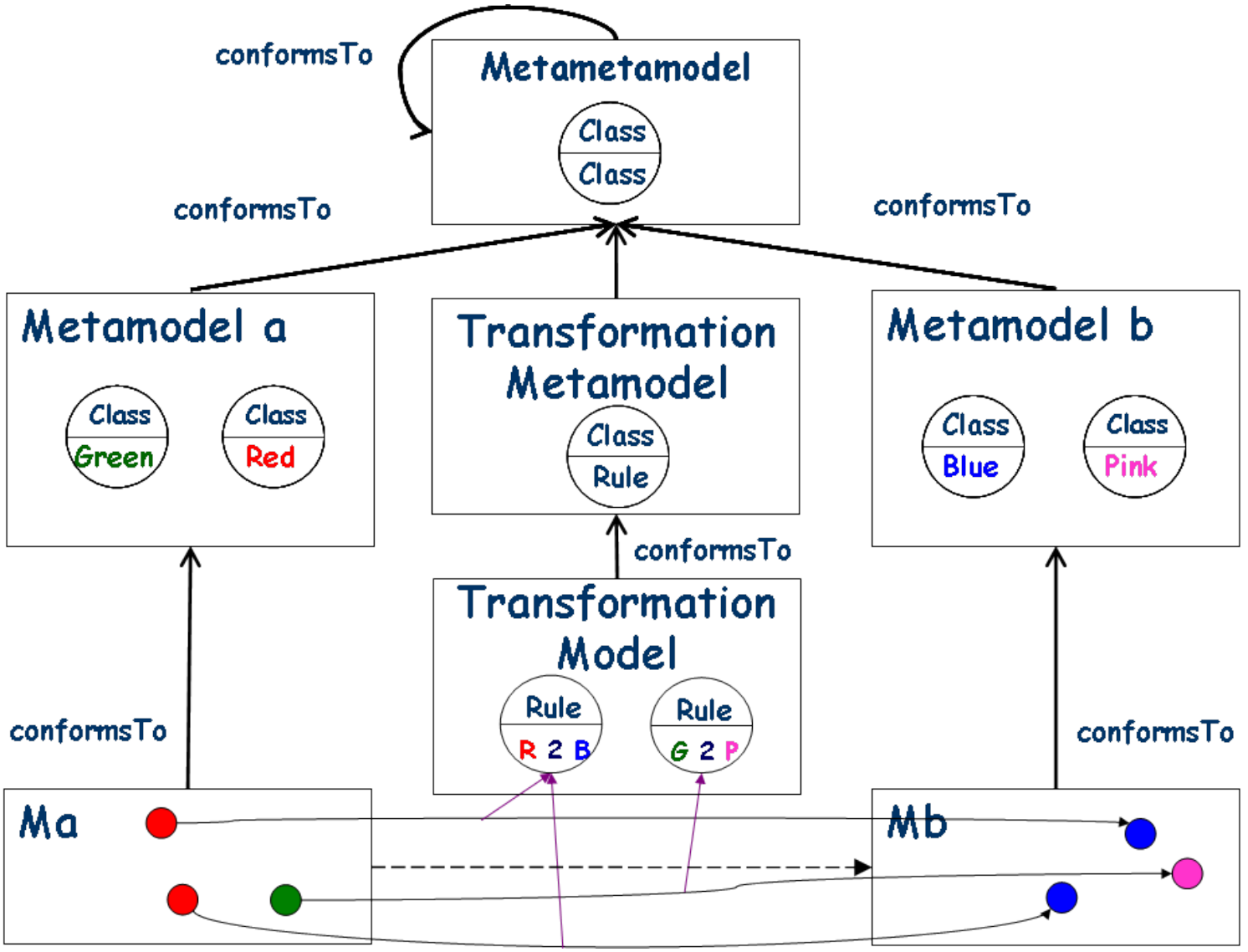
M2M Transformation Principles

- A M2M transformation is the automated creation of m target models from n source models
  - Each model conforms to a given reference model (which can be the same for several models)

- M2M transformation is not only about M1 to M1 transformations:
  - M1 to M2: promotion,
  - M2 to M1: demotion,
  - M2 to M2 or M3 to M3
  - M3 to M1, M3 to M2, etc.

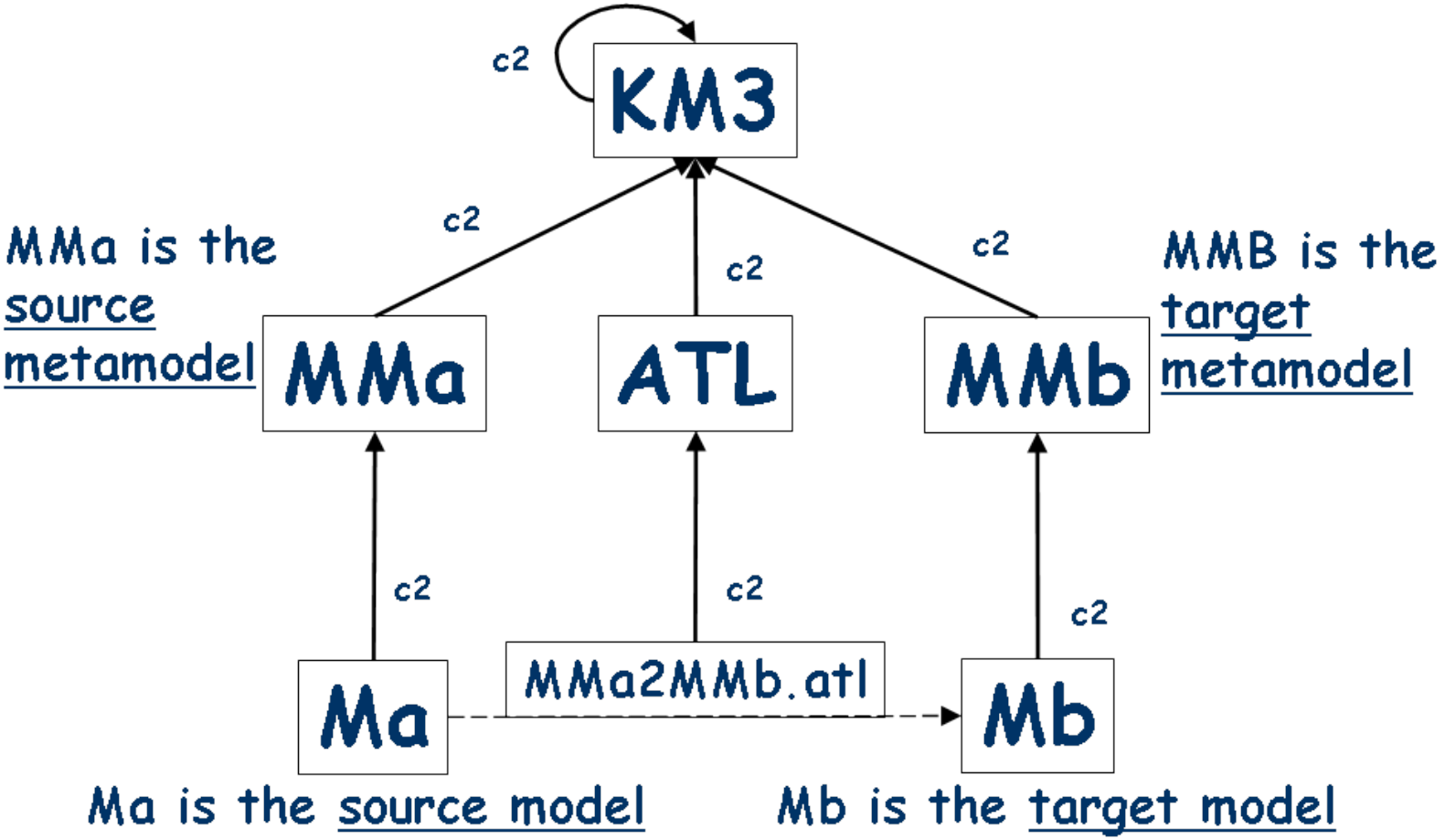


M2M Transformation Principles



M2M with ATL

- Application of the principles within the context of ATL



M2M with ATL

- Overview of the language (1/6)
  - Source models and target models are distinct:
    - Source models are read-only (they can only be navigated, not modified),
    - Target models are write-only (they cannot be navigated).
  - The language is a declarative-imperative hybrid:
    - Declarative part:
      - Matched rules with automatic traceability support,
      - Side-effect free navigation (and query) language: OCL 2.0
    - Imperative part:
      - Called rules,
      - Action blocks.
  - Recommended programming style: declarative

M2M with ATL

- Overview of the language (2/6)
  - A declarative rule specifies:
    - a source pattern to be matched in the source models,
    - a target pattern to be created in the target models for each match during rule application.
  - An imperative rule is basically a procedure:
    - It is called by its name,
    - It may take arguments,
    - It can contain:
      - A declarative target pattern,
      - An action block (i.e. a sequence of statements),
      - Both.

M2M with ATL

- Overview of the language (3/6)
  - Applying a declarative rule means:
    - Creating the specified target elements,
    - Initializing the properties of the newly created elements.
  - There are three types of declarative rules:
    - Standard rules that are applied once for each match,
      - A given set of elements may only be matched by one standard rule,
    - Lazy rules that are applied as many times for each match as it is referred to from other rules (possibly never for some matches),
    - Unique lazy rules that are applied at most once for each match and only if it is referred to from other rules.

M2M with ATL

- Overview of the language (4/6)
  - Declarative rules: source pattern
    - The source pattern is composed of:
      - A labeled set of types coming from the source metamodels
      - A guard (Boolean expression) used to filter matches
    - A match corresponds to a set of elements coming from the source models that:
      - Are of the types specified in the source pattern (one element for each type)
      - Satisfy the guard

M2M with ATL

- Overview of the language (5/6)
  - Declarative rules: target pattern
    - The target pattern is composed of:
      - A labeled set of types coming from the target metamodels
      - For each element of this set, a set of bindings
      - A binding specifies the initialization of a property of a target element using an expression
    - For each match, the target pattern is applied:
      - Elements are created in the target models (one for each type of the target pattern)
      - Target elements are initialized by executing the bindings:
        - First evaluating their value
        - Then assigning this value to the corresponding property

M2M with ATL

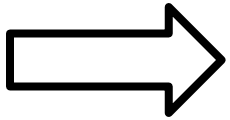
- Overview of the language (6/6)
  - Execution order of declarative rules
    - Declarative ATL frees the developer from specifying execution order:
      - The order in which rules are matched and applied is not specified (remark: the match of a lazy or unique lazy rules must be referred to before the rule is applied)
      - The order in which bindings are applied is not specified
    - The execution of declarative rules can however be kept deterministic:
      - The execution of a rule cannot change source models
        - ➔ It cannot change a match
      - Target elements are not navigable
        - ➔ The execution of a binding cannot change the value of another

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example

Transforming this …

…
Family March
Father: Jim
Mother: Cindy
Son: Brandon
Daughter: Brenda
… other Families



… into this.

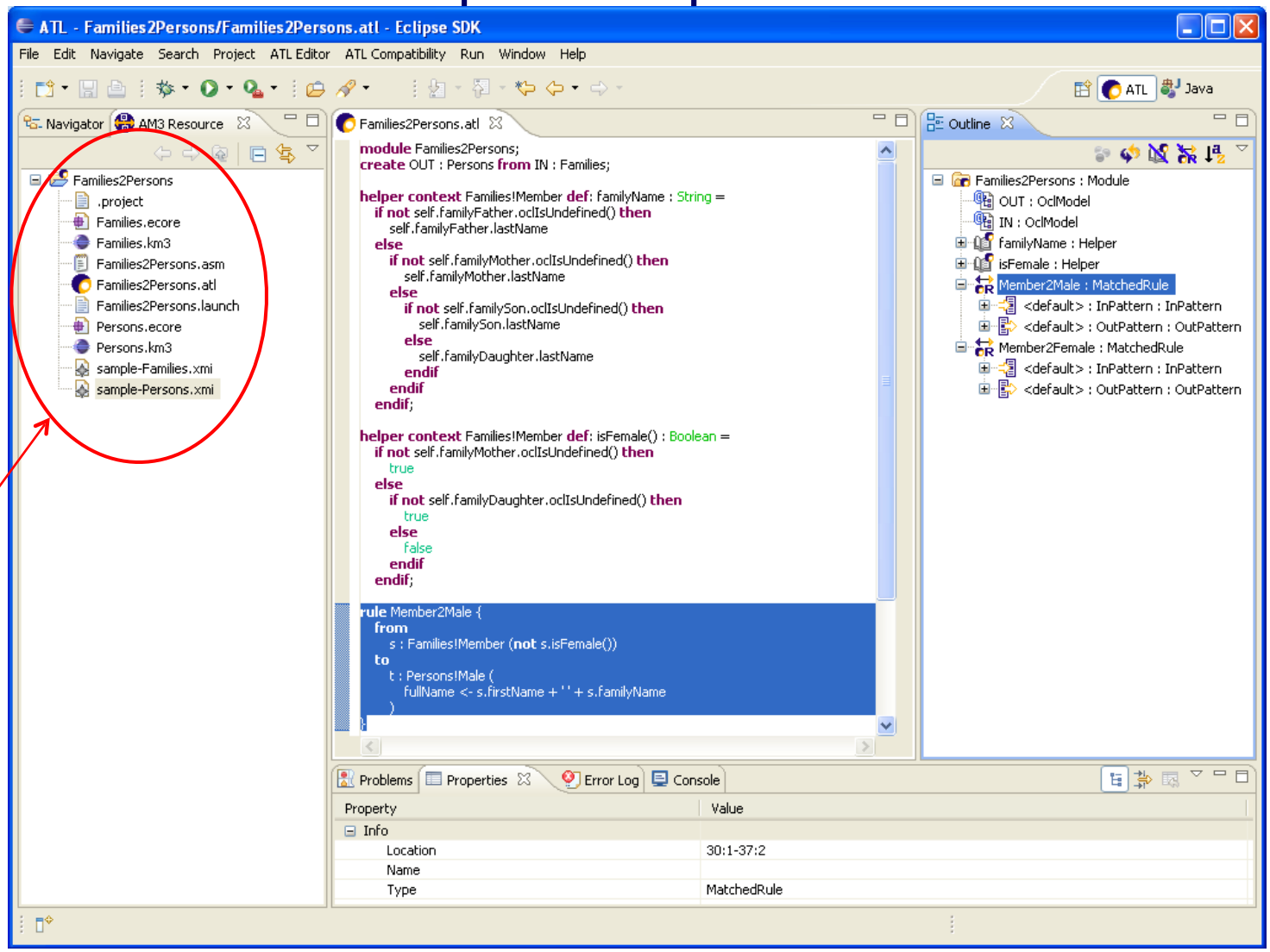
…
Mr. Jim March
Mrs. Cindy March
Mr. Brandon March
Mrs. Brenda March
… other Persons

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example

Work with the ATL Perspective

Required artefacts (models, metamodels & transformation)

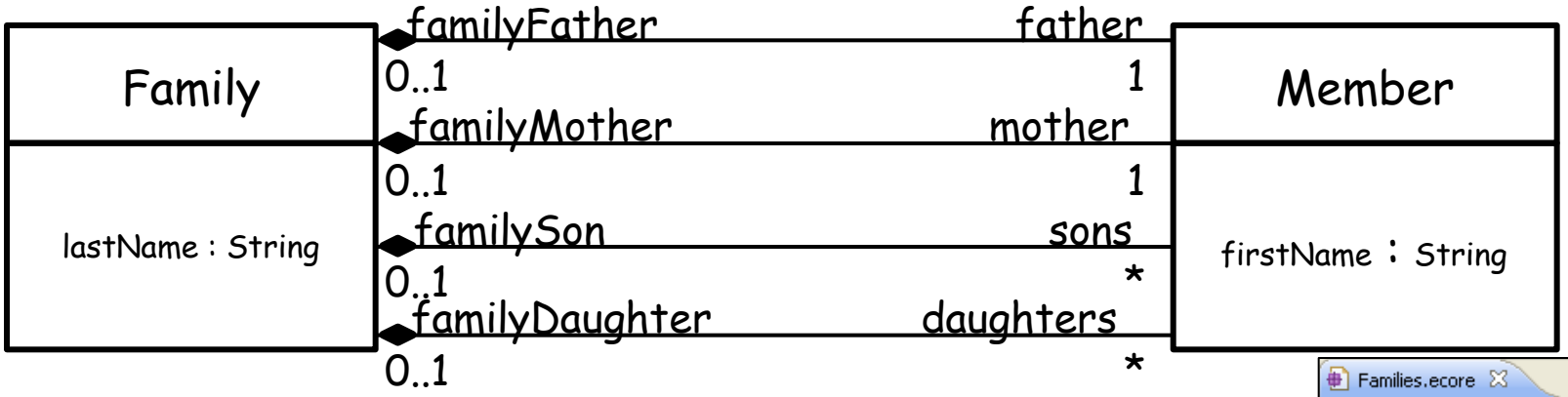


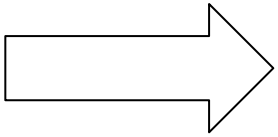
Writing a First Transformation with ATL

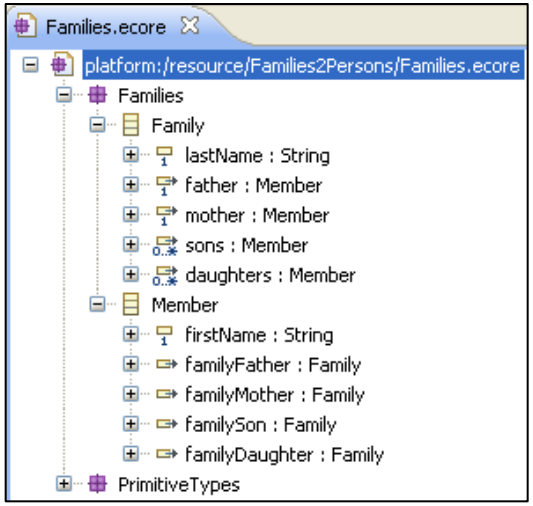
- "Families-to-Persons" Simple Example
  - In order to achieve the transformation, we need to provide:
    1. A "Families " source metamodel in Ecore (generated from its KM3 version).
    2. A source model (in XMI) conforming to "Families".
    3. A "Persons " target metamodel in Ecore (generated from its KM3 version).
    4. A "Families2Persons " transformation model in ATL.
  - When the ATL transformation is executed, we obtain:
    - A target model (in XMI) conforming to "Persons".

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - The "Family" metamodel
    - Source metamodel of the transformation

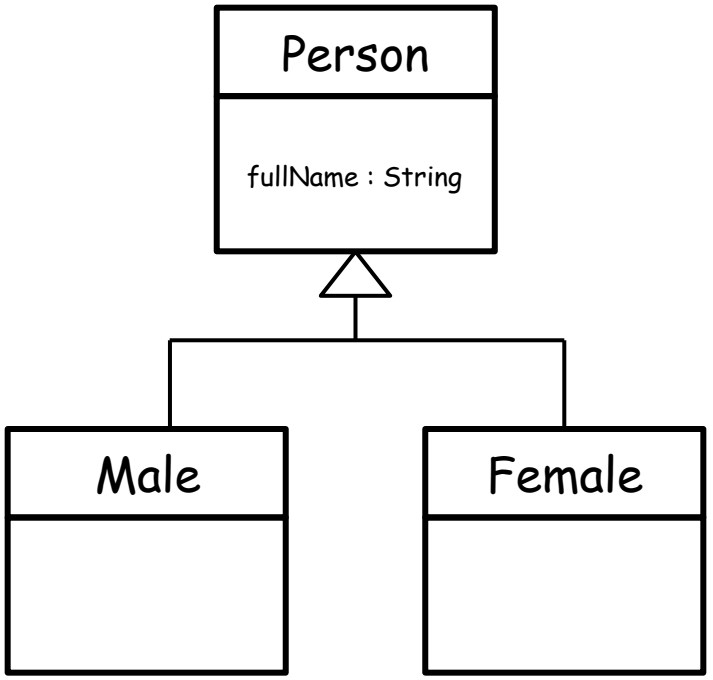


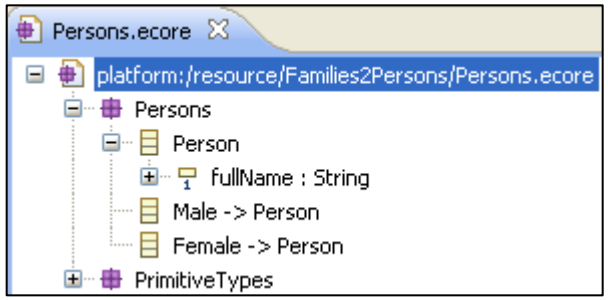




Writing a First Transformation with ATL

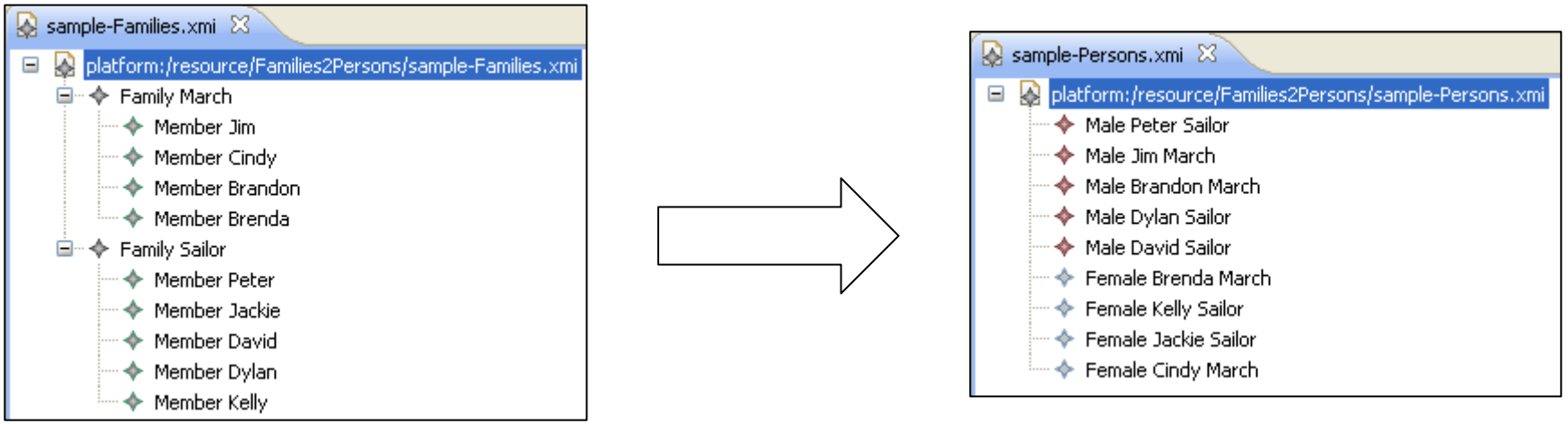
- "Families-to-Persons" Simple Example
  - The "Person" metamodel
    - Target metamodel of the transformation





Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - The "sample-Families" input and "sample-Persons" output models
    - The "sample-Families" model conforms to the "Families" metamodel
    - The "sample-Persons" model conforms to the "Persons" metamodel
    - The "sample-Persons" model is the result of the execution of the transformation on the "sample-Families" model

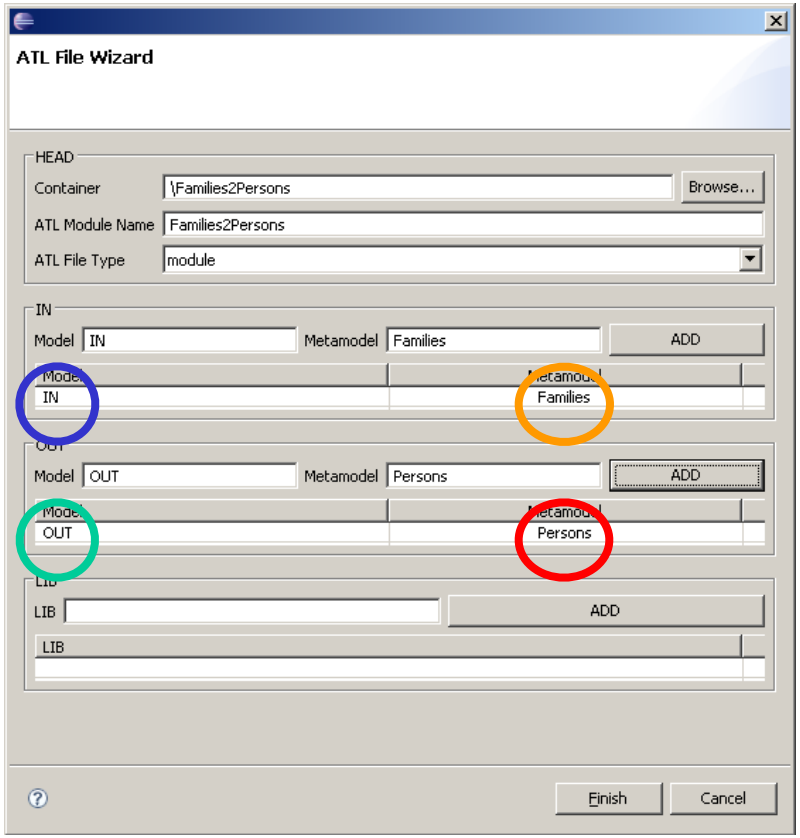


Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - To create the ATL transformation, we use the ATL File Wizard. This will generate automatically the header section.

IN:
Name of the source model in the transformation

OUT:
Name of the target model in the transformation



Families:
Name of the source metamodel in the transformation

Persons:
Name of the target metamodel in the transformation

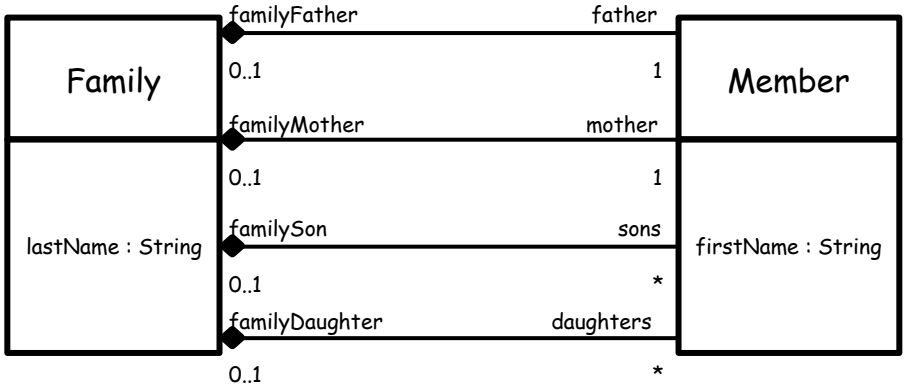
Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - The header section names the transformation module and names the variables corresponding to the source and target models ("IN" and "OUT") together with their metamodels ("Persons" and "Families") acting as types. The header section of "Families2Persons" is:

```
module Families2Persons;
create OUT : Persons from IN : Families;
```

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - A helper is an auxiliary function that computes a result needed in a rule.
  - The following helper "isFemale()" computes the gender of the current member:

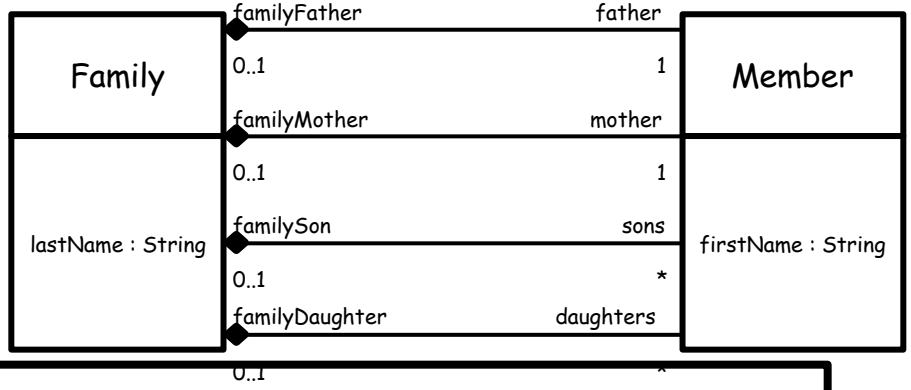



```
helper context Families!Member def: isFemale() : Boolean =
      if not self.familyMother.oclIsUndefined() then
            true
      else
            if not self.familyDaughter.oclIsUndefined() then
                  true
            else
                  false
            endif
      endif;
```

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example

- The family name is not directly contained in class "Member". The following helper returns the family name by navigating the relation between "Family" and "Member":




```
helper context Families!Member def: familyName : String =
	if not self.familyFather.oclIsUndefined() then
		self.familyFather.lastName
	else
		if not self.familyMother.oclIsUndefined() then
			self.familyMother.lastName
		else
			if not self.familySon.oclIsUndefined() then
				self.familySon.lastName
			else
				self.familyDaughter.lastName
			endif
		endif
	endif;
```

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - After the helpers we now write the rules:
    - Member to Male

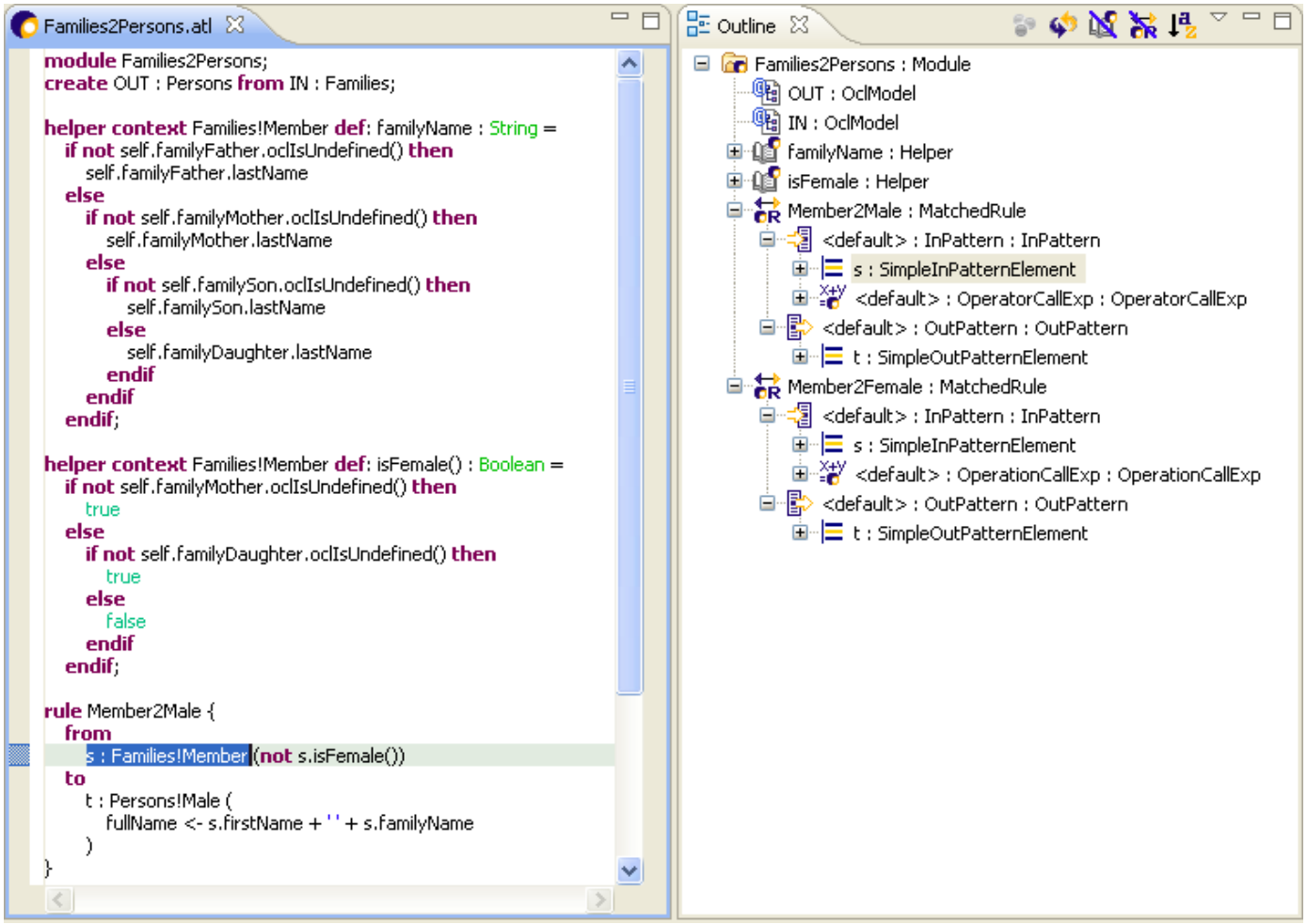
```
rule Member2Male {
        from
                s : Families!Member (not s.isFemale())
        to
                t : Persons!Male (
                        fullName <- s.firstName + ' ' + s.familyName
                )
}
```

    - Member to Female

```
rule Member2Female {
        from
                s : Families!Member (s.isFemale())
        to
                t : Persons!Female (
                        fullName <- s.firstName + ' ' + s.familyName
                )
}
```

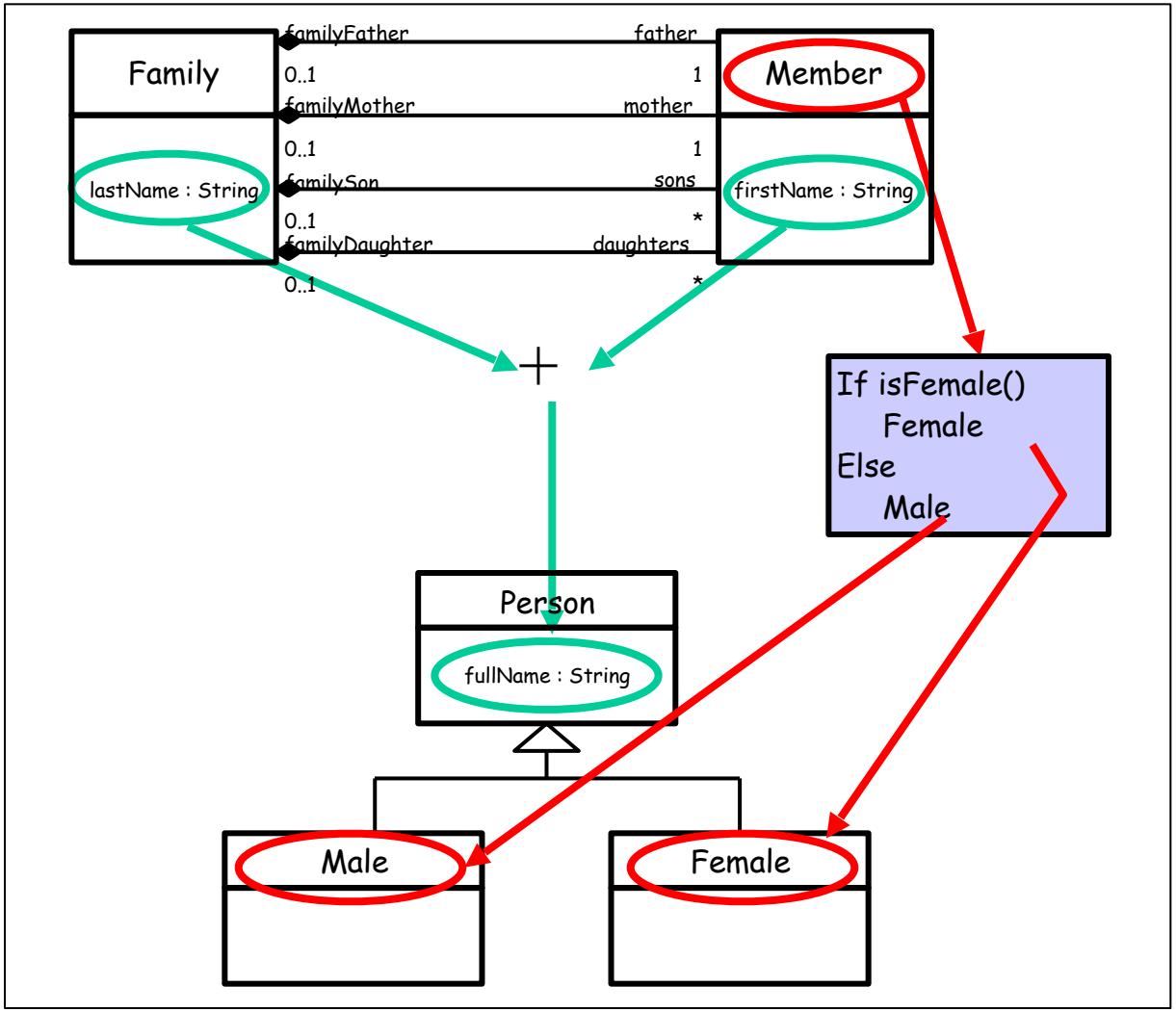
Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - The created transformation



Writing a First Transformation with ATL

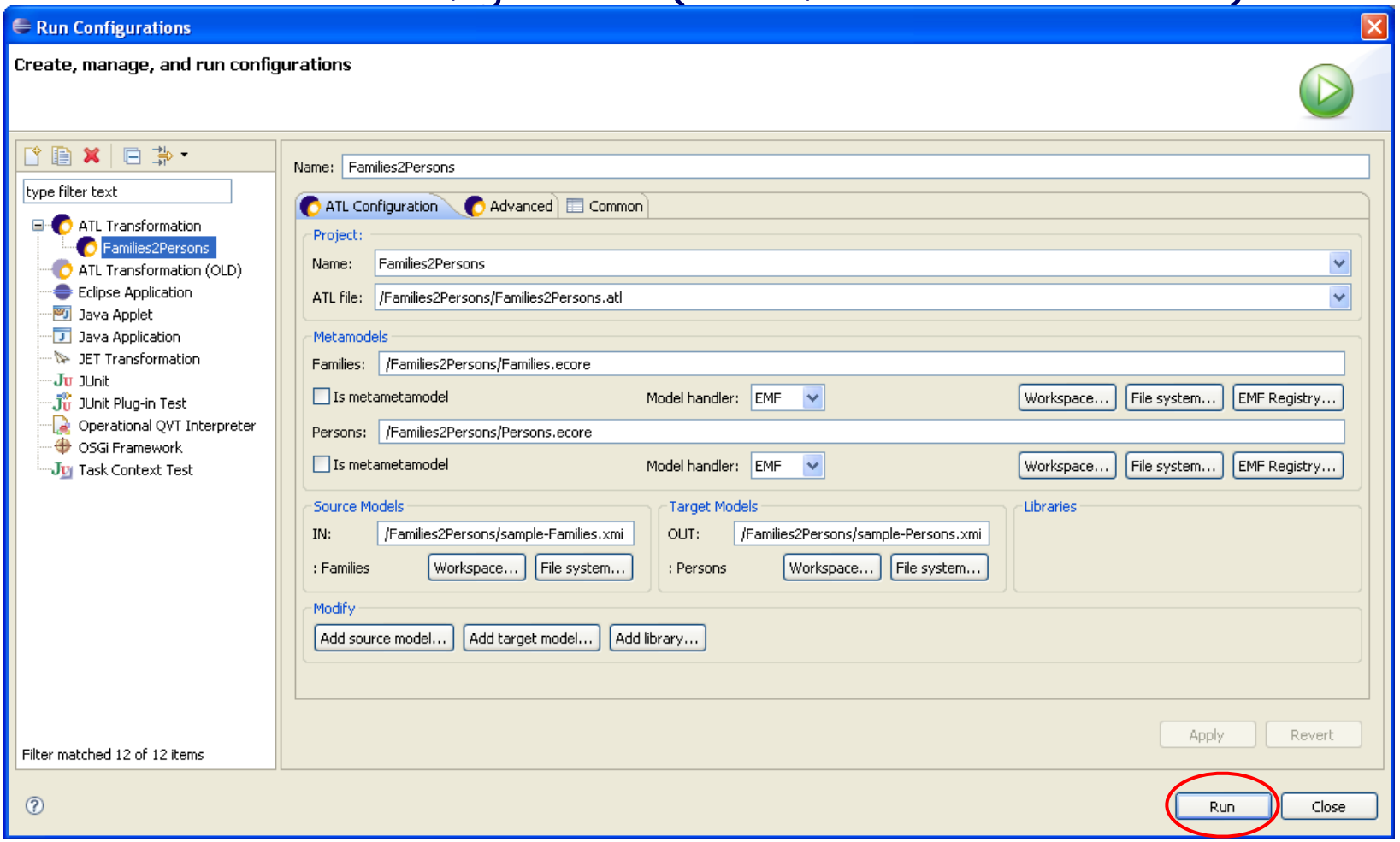
- "Families-to-Persons" Simple Example



1. For each instance of the class "Member" in the IN model, create an instance in the OUT model.
2. If the original "Member" instance is a "mother" or one of the "daughters" of a given "Family", then we create an instance of the "Female" class in the OUT model.
3. If the original "Member" instance is a "father" or one of the "sons" of a given "Family", then we create an instance of the "Male" class in the OUT model.
4. In both cases, the "fullname" of the created instance is the concatenation of the Member "firstName" and of the Family "lastName", separated by a blank.

Writing a First Transformation with ATL

- "Families-to-Persons" Simple Example
  - ATL launch configuration (transformation execution)



References

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