

"Families to Persons"

A simple illustration of model-to-model transformation

Freddy Allilaire
Frédéric Jouault

ATLAS group, INRIA & University of Nantes, France





Context of this work

- The present courseware has been elaborated in the context of the "Usine Logicielle" project (www.usine-logicielle.org) of the cluster System@tic Paris-Région with the support of the Direction Générale des Entreprises, Conseil Régional d'Ile de France, Conseil Général des Yvelines, Conseil Général de l'Essonne, and Conseil Général des Hauts de Seine.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 license.



Overview

- This presentation describes a very simple model transformation example, some kind of ATL "hello world".
- It is intended to be extended later.
- The presentation is composed of the following parts:
 - Prerequisites.
 - Introduction.
 - Metamodeling.
 - Transformation.
 - Conclusion.

Prerequisites

- In the presentation we will not discuss the prerequisites.
- The interested reader may look in another presentation to these prerequisites on:
 - MDE (MOF, XMI, OCL).
 - Eclipse/EMF (ECORE).
 - AMMA/ATL.

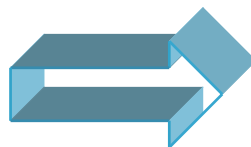
Introduction

- The goal is to present a use case of a model to model transformation written in ATL.
- This use case is named: "Families to Persons".
- Initially we have a text describing a list of families.
- We want to transform this into another text describing a list of persons.

Goal of the ATL transformation we are going to write

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

Let's suppose these are not texts, but models
(we'll discuss the correspondence
between models and texts later).

Input of the transformation is a model



Family March
Father: Jim
Mother: Cindy
Son: Brandon
Daughter: Brenda
Family Sailor
Father: Peter
Mother: Jackie
Son: David
Son: Dylan
Daughter: Kelly

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://
www.omg.org/XMI" xmlns="Families">
  <Family lastName="March">
    <father firstName="Jim"/>
    <mother firstName="Cindy"/>
    <sons firstName="Brandon"/>
    <daughters firstName="Brenda"/>
  </Family>
  <Family lastName="Sailor">
    <father firstName="Peter"/>
    <mother firstName="Jackie"/>
    <sons firstName="David"/>
    <sons firstName="Dylan"/>
    <daughters firstName="Kelly"/>
  </Family>
</xmi:XMI>
```

This is the text.

This is the corresponding model.
It is expressed in XMI,
a standard way to represent models.

Output of the transformation should be a model



Mr. Dylan Sailor
Mr. Peter Sailor
Mr. Brandon March
Mr. Jim March
Mr. David Sailor
Mrs. Jackie Sailor
Mrs. Brenda March
Mrs. Cindy March
Mrs. Kelly Sailor

This is the text.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns="Persons">
  <Male fullName="Dylan Sailor"/>
  <Male fullName="Peter Sailor"/>
  <Male fullName="Brandon March"/>
  <Male fullName="Jim March"/>
  <Male fullName="David Sailor"/>
  <Female fullName="Jackie Sailor"/>
  <Female fullName="Brenda March"/>
  <Female fullName="Cindy March"/>
  <Female fullName="Kelly Sailor"/>
</xmi:XMI>
```

This is the corresponding model
(The corresponding XMI file is named
"sample-Persons.ecore").

Each model conforms to a metamodel

Source metamodel

conformsTo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://
www.omg.org/XMI" xmlns="Families">
  <Family lastName="March">
    <father firstName="Jim"/>
    <mother firstName="Cindy"/>
    <sons firstName="Brandon"/>
    <daughters firstName="Brenda"/>
  </Family>
  <Family lastName="Sailor">
    <father firstName="Peter"/>
    <mother firstName="Jackie"/>
    <sons firstName="David"/>
    <sons firstName="Dylan"/>
    <daughters firstName="Kelly"/>
  </Family>
</xmi:XMI>
```

Source model

"sample-Families.ecore"

Target metamodel

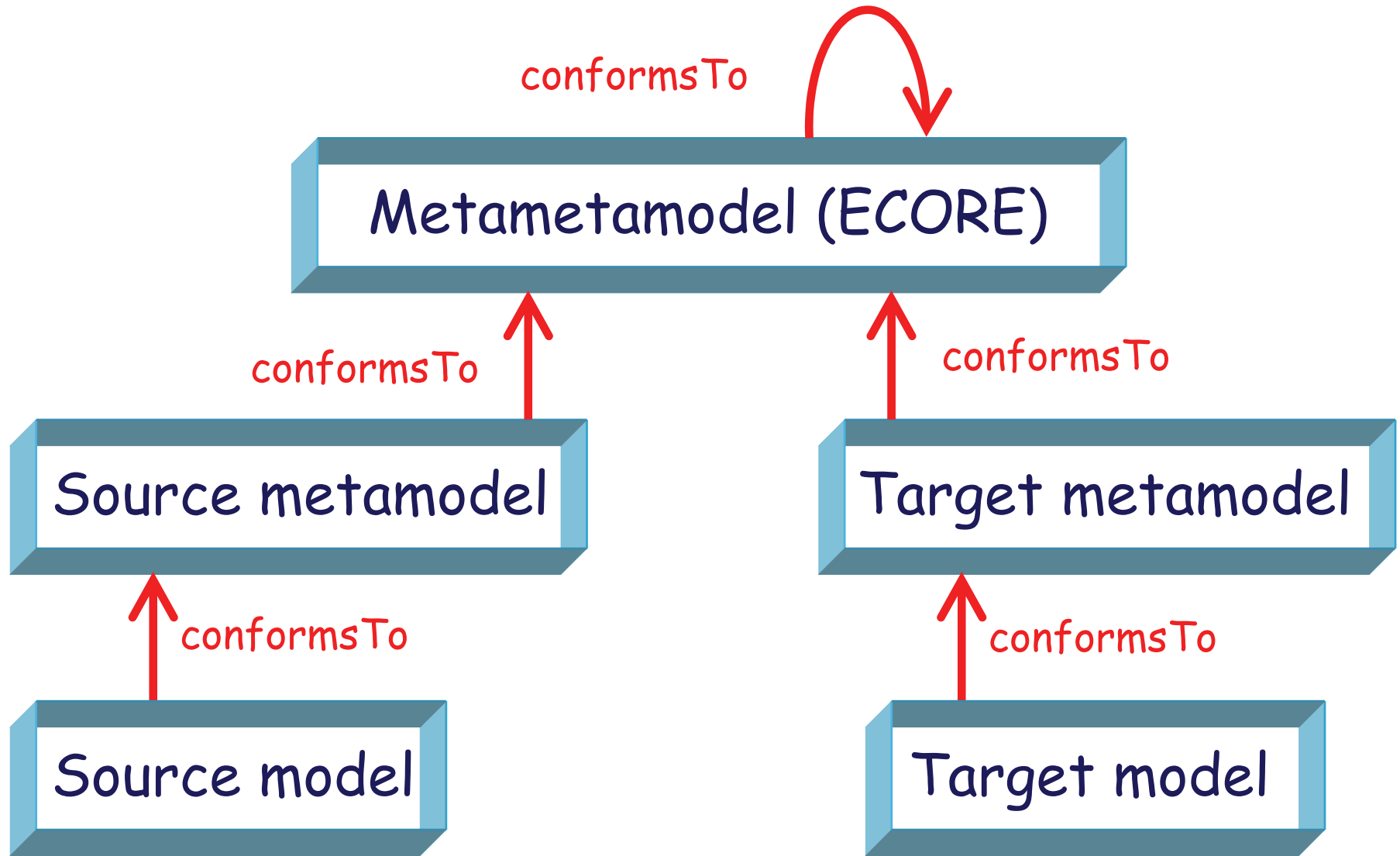
conformsTo

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns="Persons">
  <Male fullName="Dylan Sailor"/>
  <Male fullName="Peter Sailor"/>
  <Male fullName="Brandon March"/>
  <Male fullName="Jim March"/>
  <Male fullName="David Sailor"/>
  <Female fullName="Jackie Sailor"/>
  <Female fullName="Brenda March"/>
  <Female fullName="Cindy March"/>
  <Female fullName="Kelly Sailor"/>
</xmi:XMI>
```

Target model

"sample-Persons.ecore"

The general picture



What we need to provide

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

Definition of the source metamodel "Families"

What is "Families":

A collection of families.

Each family has a name and is composed of members:

A father

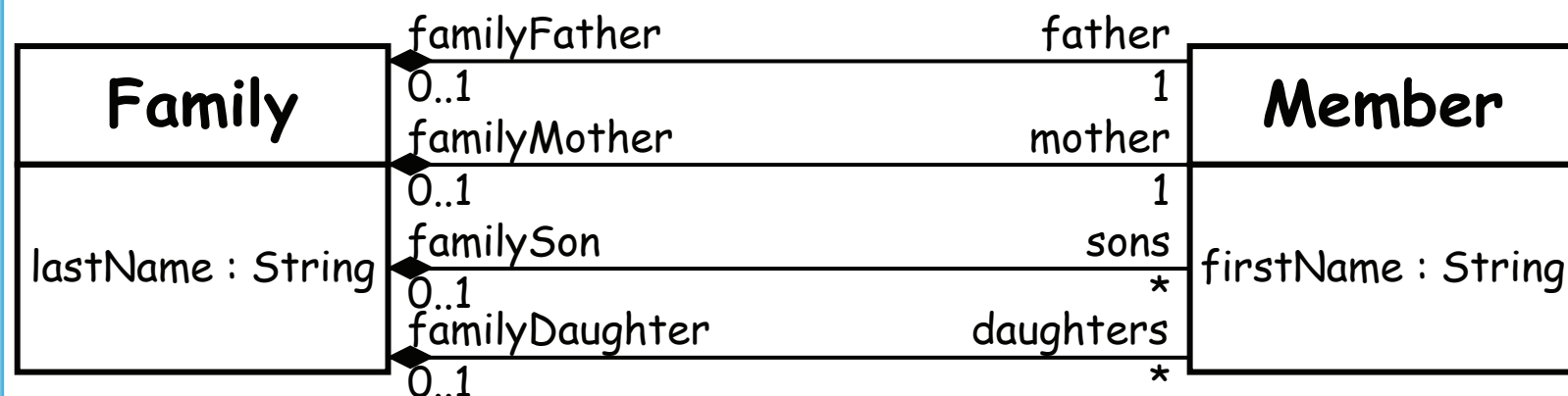
A mother

Several sons

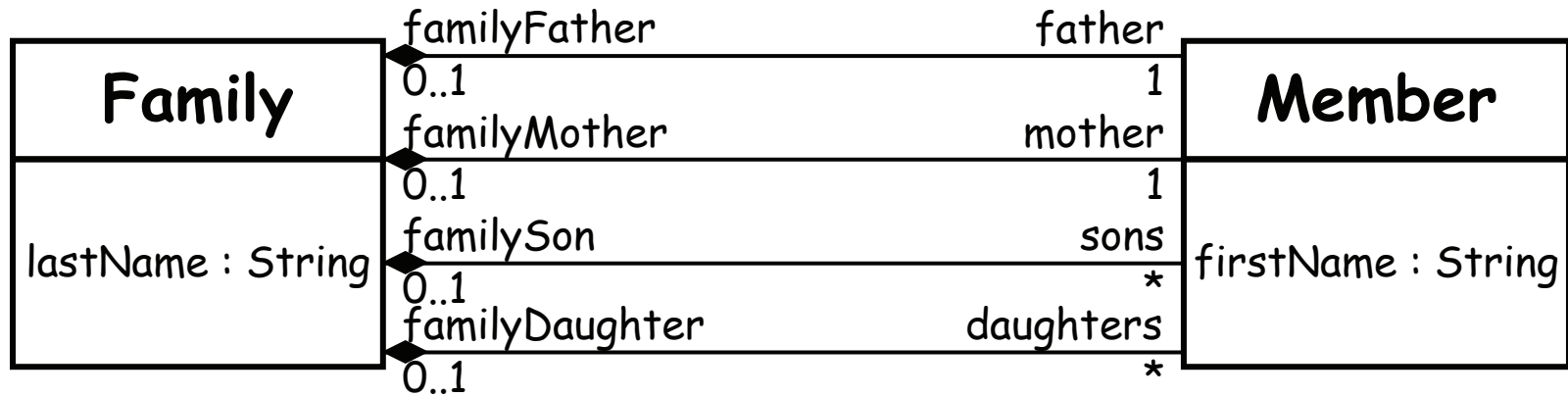
Several daughters

Each family member has a first name.

Family March
 Father: Jim
 Mother: Cindy
 Son: Brandon
 Daughter: Brenda
 Family Sailor
 Father: Peter
 Mother: Jackie
 Sons: David, Dylan
 Daughter: Kelly



"Families" metamodel (visual presentation and KM3)



```

package Families {

  class Family {
    attribute lastName : String;
    reference father container : Member oppositeOf familyFather;
    reference mother container : Member oppositeOf familyMother;
    reference sons[*] container : Member oppositeOf familySon;
    reference daughters[*] container : Member oppositeOf familyDaughter;
  }

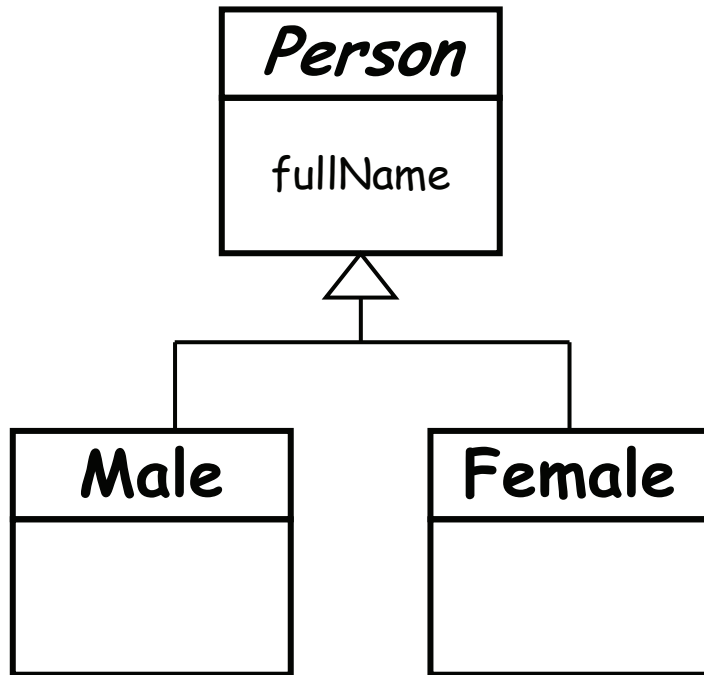
  class Member {
    attribute firstName : String;
    reference familyFather[0-1] : Family oppositeOf father;
    reference familyMother[0-1] : Family oppositeOf mother;
    reference familySon[0-1] : Family oppositeOf sons;
    reference familyDaughter[0-1] : Family oppositeOf daughters;
  }

}

package PrimitiveTypes {
  datatype String;
}

```

"Persons" metamodel (visual presentation and KM3)



```
package Persons {

    abstract class Person {
        attribute fullName : String;
    }

    class Male extends Person { }

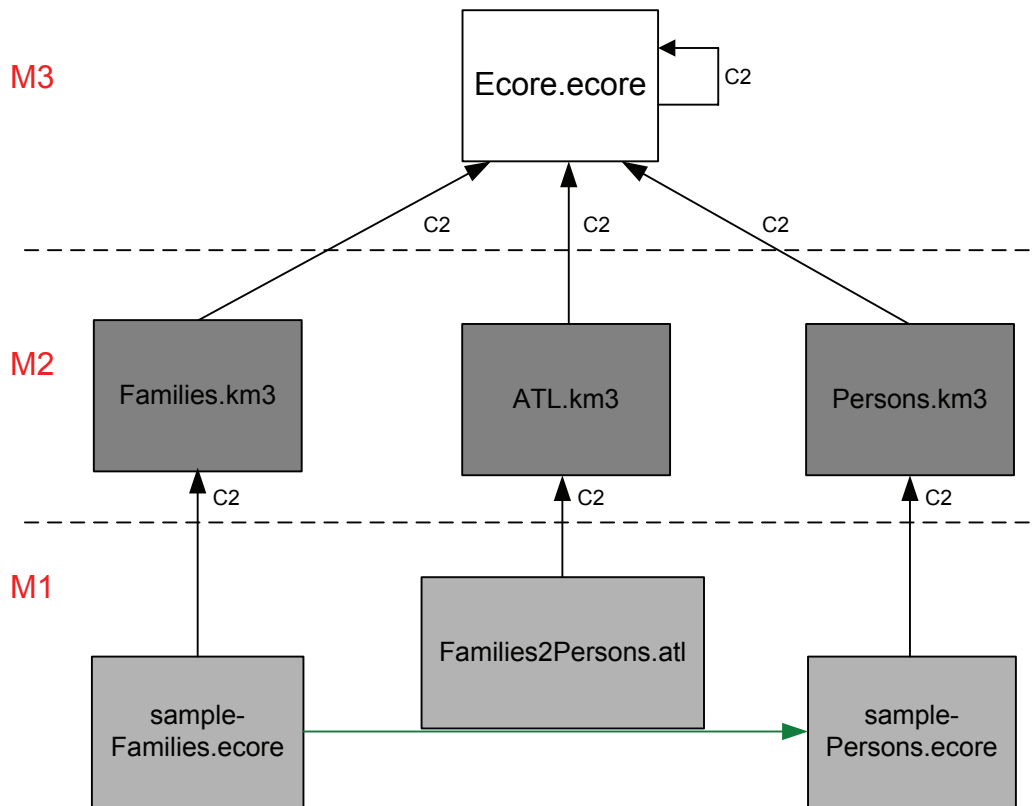
    class Female extends Person { }

}

package PrimitiveTypes {
    datatype String;
}
```

The big picture

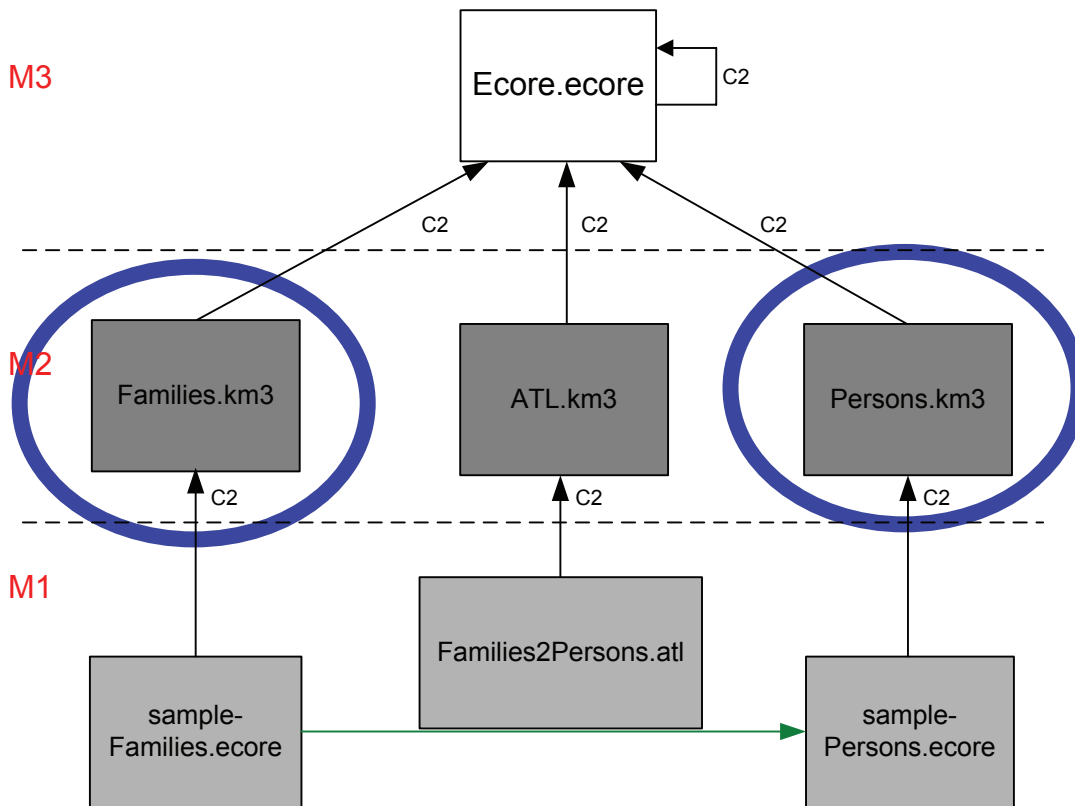
Eclipse Modeling Framework (EMF)



1. Our goal in this mini-tutorial is to write the ATL transformation, stored in the "Families2Persons" file.
2. Prior to the execution of this transformation the resulting file "sample-Persons.ecore" does not exist. It is created by the transformation.
3. Before defining the transformation itself, we need to define the source and target metamodels ("Families.km3" and "Person.KM3").
4. We take for granted that the definition of the ATL language is available (supposedly in the "ATL.km3" file).
5. Similarly we take for granted that the environment provides the recursive definition of the metamodel (supposedly in the "Ecore.ecore" file).

Families to Persons Architecture

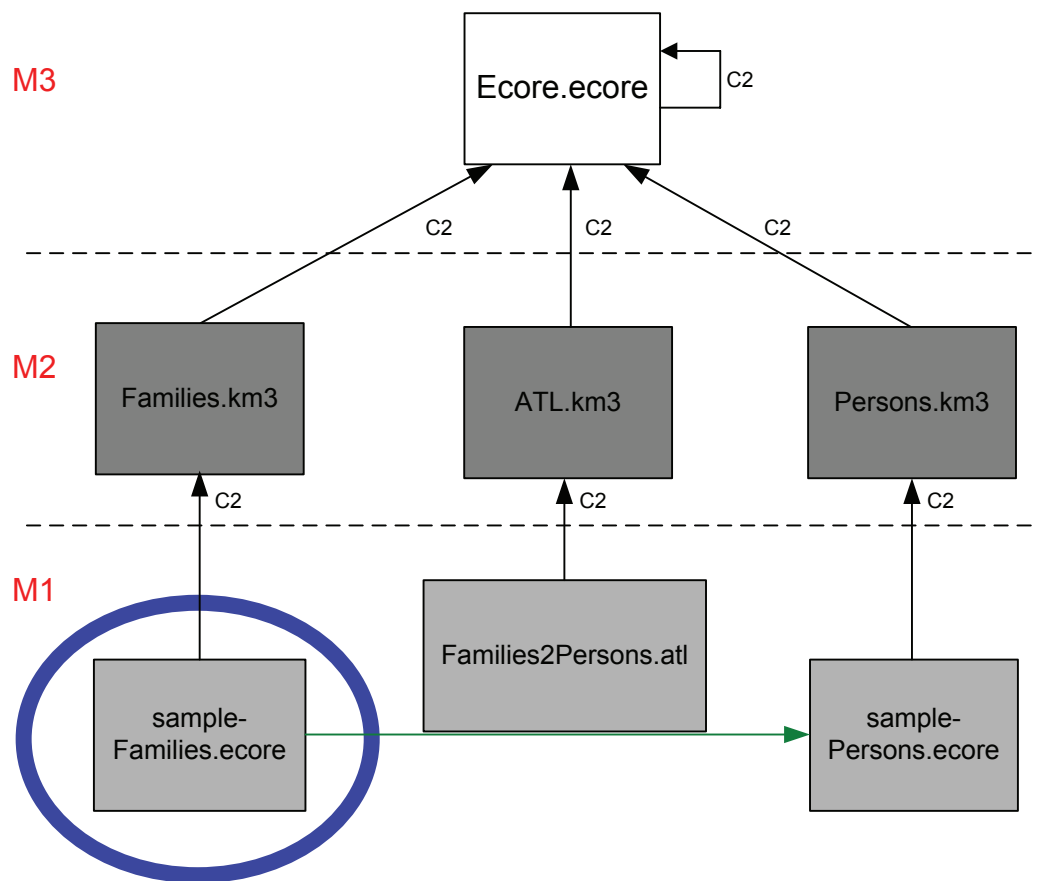
Eclipse Modeling Framework (EMF)



1. *Families* and *Persons* metamodels have been created previously.
2. They have been written in the KM3 metamodel specification DSL (Domain Specific Language).

Families to Persons Architecture

Eclipse Modeling Framework (EMF)

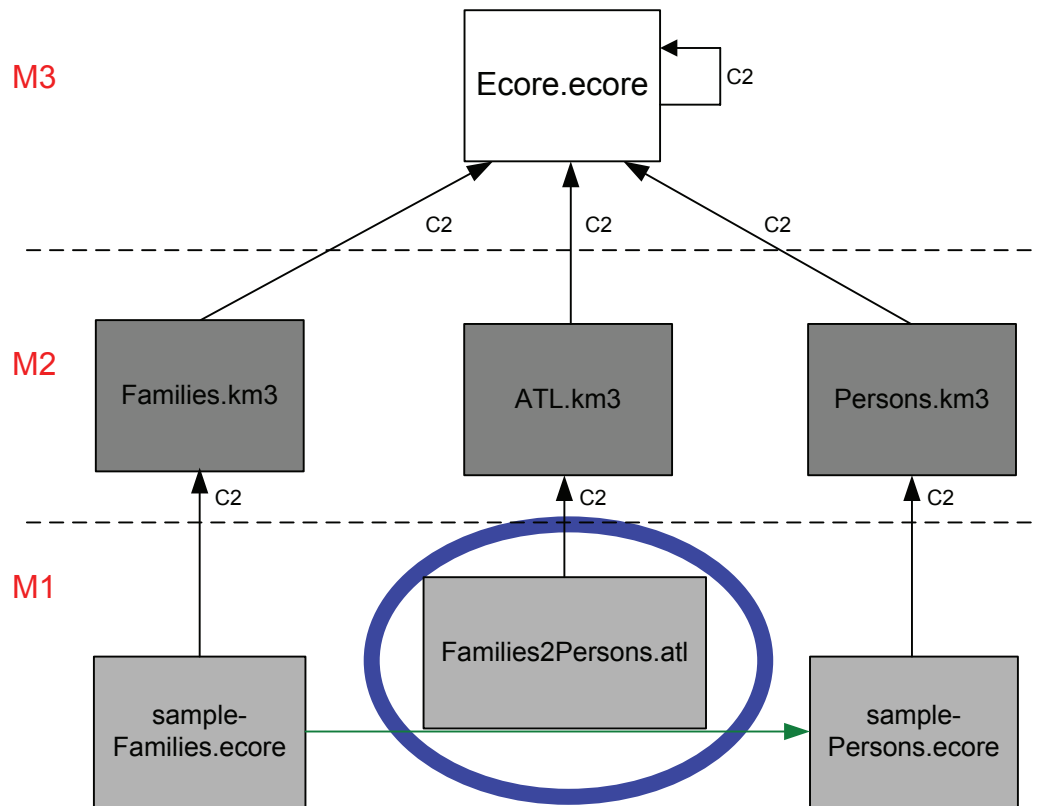


- The following file is the sample that we will use as source model in this use case:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://
www.omg.org/XMI" xmlns="Families">
  <Family lastName="March">
    <father firstName="Jim"/>
    <mother firstName="Cindy"/>
    <sons firstName="Brandon"/>
    <daughters firstName="Brenda"/>
  </Family>
  <Family lastName="Sailor">
    <father firstName="Peter"/>
    <mother firstName="Jackie"/>
    <sons firstName="David"/>
    <sons firstName="Dylan"/>
    <daughters firstName="Kelly"/>
  </Family>
</xmi:XMI>
```

Families to Persons Architecture

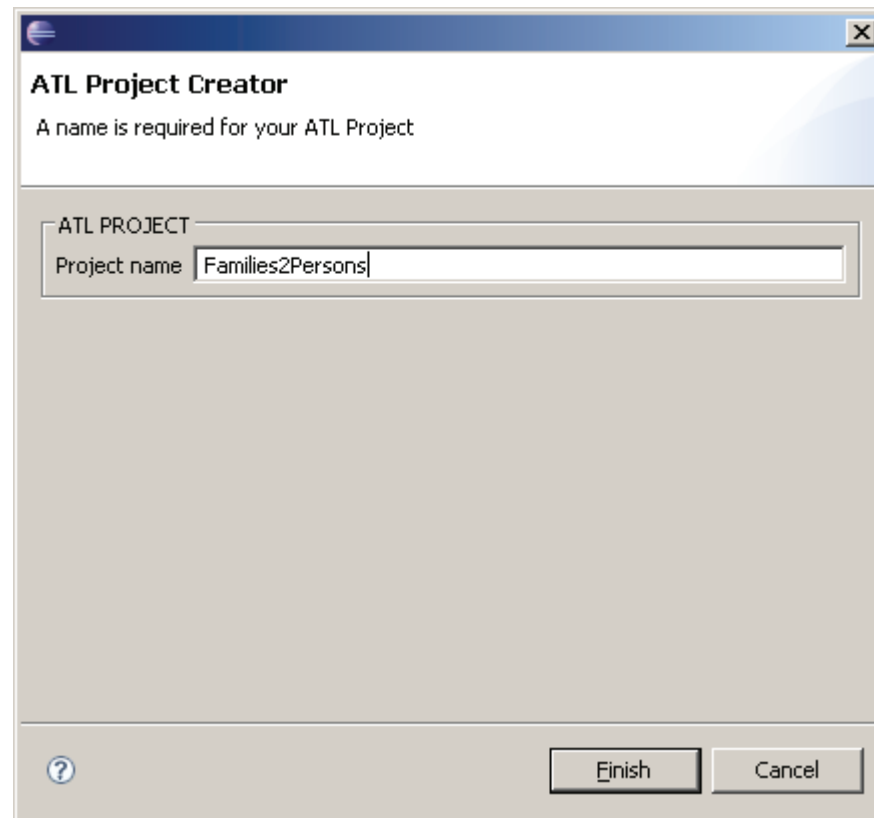
Eclipse Modeling Framework (EMF)



1. Now, let us start the creation of the ATL transformation *Families2Persons.atl*.
2. We suppose the ATL environment is already installed.
3. The creation of the ATL transformation will follow several steps as described in the next slides.

Families to Persons: project creation

- First we create an ATL project by using the ATL Project Wizard.



Families to Persons: ATL transformation creation

- Next we create the ATL transformation. To do this, 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

ATL File Wizard

HEAD

Container: \\Families2Persons [Browse...]

ATL Module Name: Families2Persons

ATL File Type: module

IN

Model: IN Metamodel: Families [ADD]

Model: IN Metamodel: Families

OUT

Model: OUT Metamodel: Persons [ADD]

Model: OUT Metamodel: Persons

LIB

LIB [ADD]

LIB

[?] [Finish] [Cancel]

Families:
Name of the source metamodel in the transformation

Persons:
Name of the target metamodel in the transformation

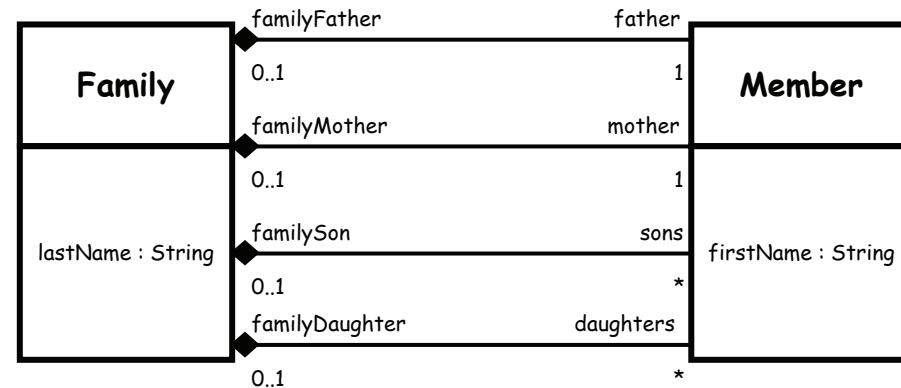
Families to Persons: header section

- 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;
```

Families to Persons: helper "isFemale()"

- 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:

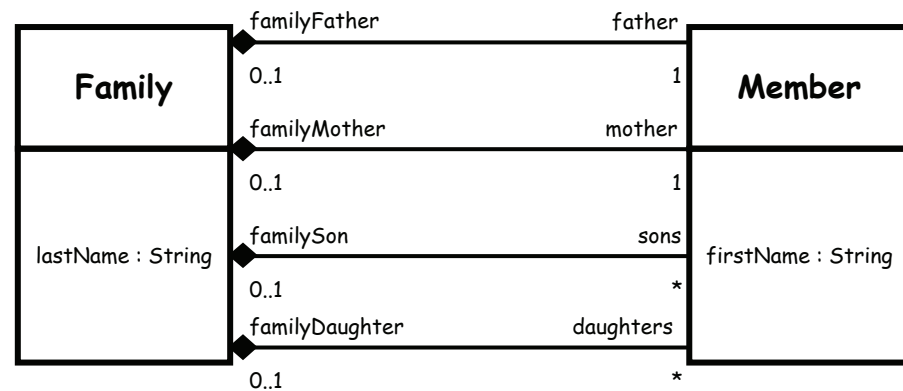


```

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;
  
```

Families to Persons: helper "familyName"

- 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;
  
```

Families to Persons: writing the rules

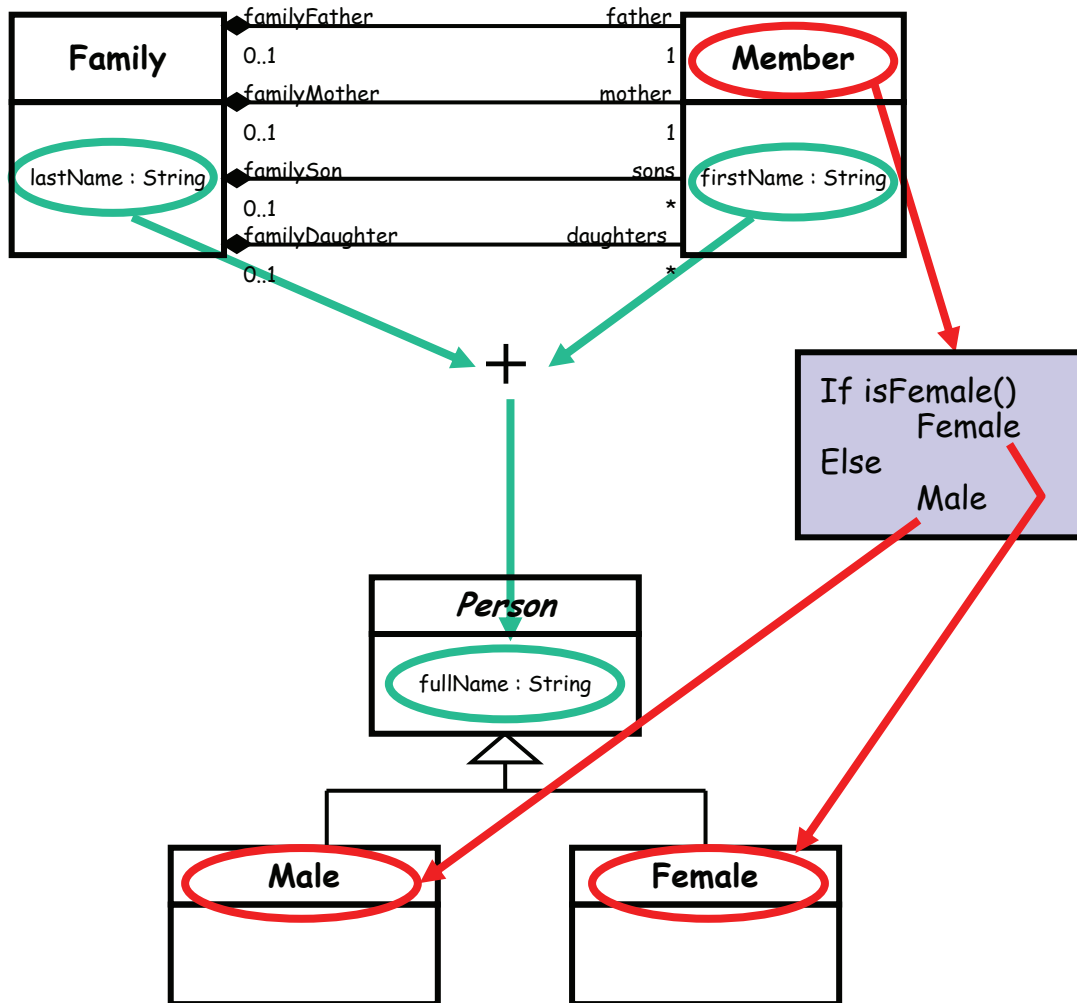
- After the helpers we now write the rules:
 - Member to Male

```
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  
    )  
}
```

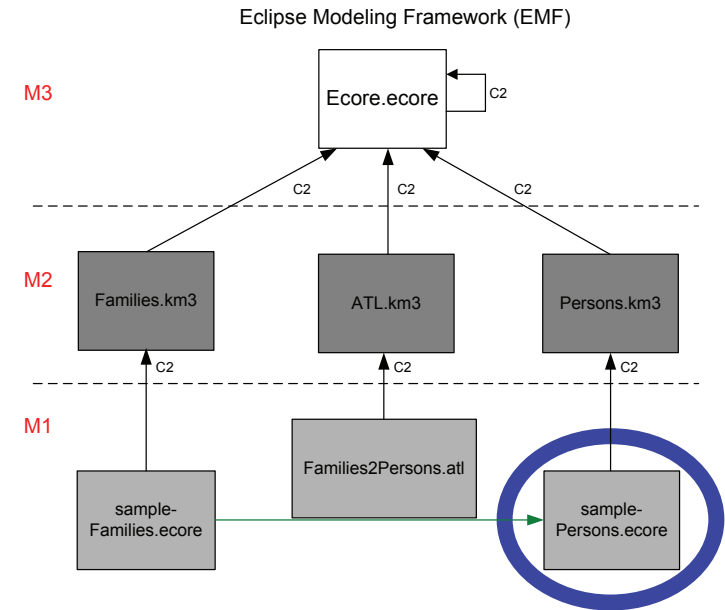

Summary of the Transformation



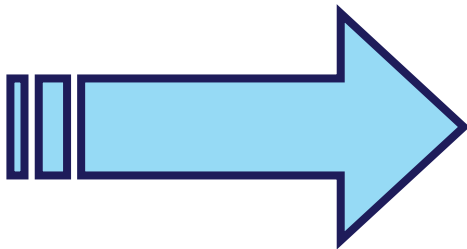
1. For each instance of the class "Member" in the IN model, create an instance in the OUT model.
2. If the original "Member" instance is a "mother" or one of the "daughters" of a given "Family", then we create an instance of the "Female" class in the OUT model.
3. If the original "Member" instance is 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.

Families to Persons Architecture

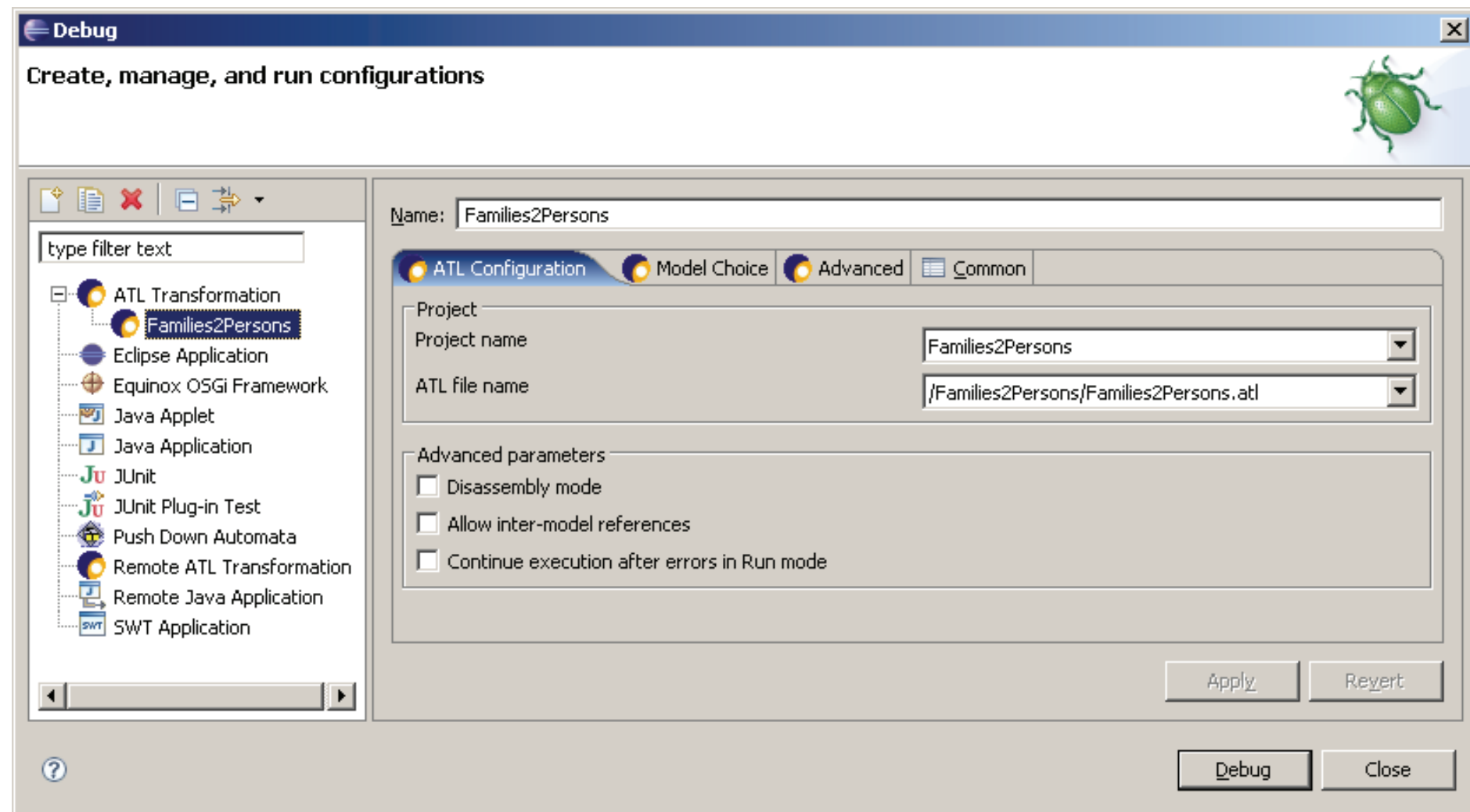
1. Once the ATL transformation "Families2Persons" is created, we can execute it to build the OUT model.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns="Persons">
  <Male fullName="Dylan Sailor"/>
  <Male fullName="Peter Sailor"/>
  <Male fullName="Brandon March"/>
  <Male fullName="Jim March"/>
  <Male fullName="David Sailor"/>
  <Female fullName="Jackie Sailor"/>
  <Female fullName="Brenda March"/>
  <Female fullName="Cindy March"/>
  <Female fullName="Kelly Sailor"/>
</xmi:XMI>
```



ATL Launch Configuration - 1



ATL Launch Configuration - 2

module Families2Persons;

create OUT : Persons **from** IN : Families;

Debug

Create, manage, and run configurations

Warning, none model (or metamodel) is registered

Name: Families2Persons

ATL Configuration
 Model Choice
 Advanced
 Common

IN

Model : Meta Model :

Model	Meta-model
IN	Families

Add
Remove

OUT

Model : Meta Model :

Model	Meta-model
OUT	Persons

Add
Remove

Path Editor

Model	Path	Model H...
Families	/Families2Persons/Families.ecore	EMF
Persons	/Families2Persons/Persons.ecore	EMF
OUT	/Families2Persons/sample-Persons.ecore	
IN	/Families2Persons/sample-Families.ecore	

EMF
MDR

Select Model Handler

Set path

Set external path

MM Is MOF-1.4

MM is Ecore

Metamodel by URI

Libs

Lib :

Libs	Path

Add

Set path

Set external path

Remove lib

Apply Revert

Debug Close

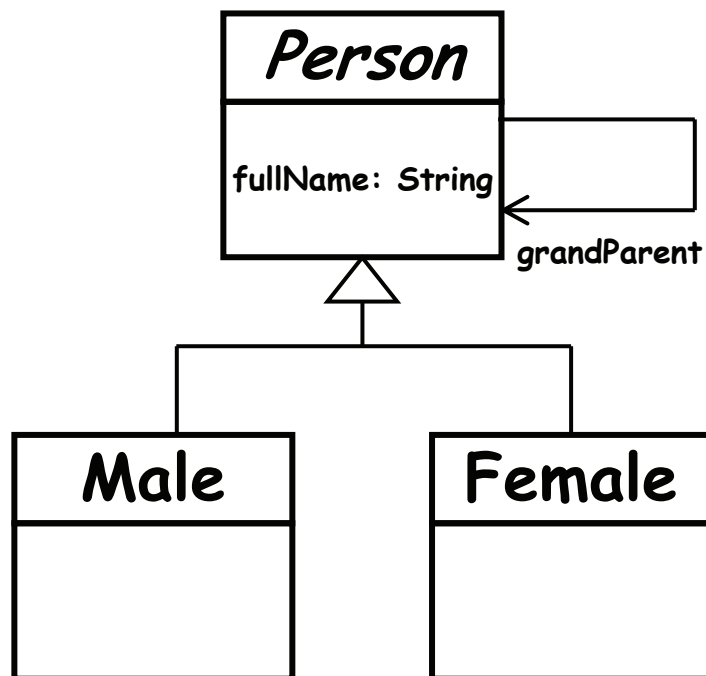
Summary

- We have presented here a "hello world" level basic ATL transformation.
- This is not a recommendation on how to program in ATL, just an initial example.
- Several questions have not been answered
 - Like how to transform a text into an XMI-encoded model.
 - Or how to transform the XMI-encoded result into text.
- For any further questions, see the documentation mentioned in the resource page (FAQ, Manual, Examples, etc.).

ATL Resource page

- ATL Home page
 - <http://www.eclipse.org/m2m/atl/>
- ATL Documentation page
 - <http://www.eclipse.org/m2m/atl/doc/>
- ATL Newsgroup
 - <news://news.eclipse.org/eclipse.modeling.m2m>
- ATL Wiki
 - <http://wiki.eclipse.org/index.php/ATL>

Working on the example



- There are a lot of exercise questions that could be based on this simple example.
- For example, modify the target metamodel as shown and compute the "grandParent" for any Person.