Software Architectures

Assignment 2: Model-to-model transformation

Assistants: Humberto Rodriguez Avila, Kennedy Kambona
Email: {rhumbert, kkambona}@vub.be
Office: {10F719, 10F732}

Deadline: May 21, 2017, 23:59

In this assignment, you will use the ATLAS Transformation Language (ATL[1]) to translate an input (*source*) model into one or more output (*target*) models via transformation rules.

Assignment

For this assignment you will create a ATL project to declare the *meta model*(s) and *model transformation*(s) required for the following problem.

Deadline: May 21, 2017 at 23:59. The deadline is fixed and will not be extended.

Deliverables A *report* (in English, max 1 page excluding screenshots) explaining the solution to the problem.

The report should be handed in as a single PDF file.

The file should follow the naming schema `(Firstname-Lastname_)*2.pdf`, for example: `Humberto-Rodriguez_Kennedy-Kambona_2.pdf`.

Source code of the project exported from eclipse.
Submit the *report* and *source code* as a single zip file on the Software Architectures course page[2] in PointCarré, by clicking on *Assignments (Opdrachten) > Assignment 2*.

Team work You are allowed to work alone or in a team of two. Only one of you should submit the report on PointCarré, but be sure to **mention both names** in the report!

Note that copying – whether from previous years, from other teams, or from the internet – will not be tolerated, and can lead to a zero for the complete course.

[1] ATL Documentation: `http://wiki.eclipse.org/MMT/ATL_Transformation_Language_(ATL)`
[2] Use the English variant "Software Architectures", rather than the Dutch one "Software Architecturen".

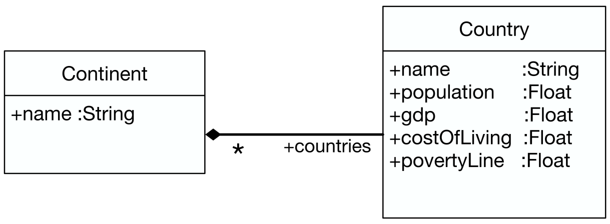
If you use any other resources besides those provided in the lectures and in this document, remember to cite them in your report.

Grading The exercises will be graded and can become subject of an additional defense.

Problem Description

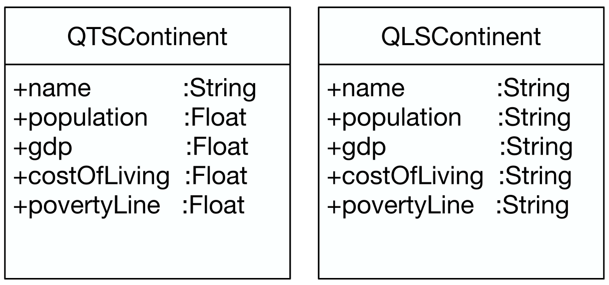
In this assignment you will work with the ATL language transformation and tool. The objectives of this assignment are to perform several transformations from a list of **Continents** to a list of **Statistics** based on quantitive information about these continents.

We want to classify different continents according to various statistics of its countries. On one side (the source), we have a list of **Continents**. In the metamodel **Continents** the class **Continent** has a *name*, and a list of *countries.* This metamodel also contains a class **Country** with the following attributes *name, population, gdp, costOfLiving, povertyLine.* The value of all these attributes are expressed in **millions** and their types are *floating points (or double)* except for *name* that is *string*, and *povertyLine* that is a percentage. Each country belongs to a continent (only one).



For this assignment you have to do the following transformations:

1. Transform the **Continents** models into a new models of *quantitative statistics*(the target), we can call it **"QTSContinent"**. The metamodel of the new models contains the following attributes *name of the continent, total of population, average of gdp, average of costOfLiving, average of povertyLine.* The value of all these attributes will be calculated based on the values of the continent's list of countries. In other words, all countries of a Continent have to be visited to calculate the values of these attributes.



2. Transform the **Continents** models into a new models of *qualitative statistics*(the target), we can call it **"QLSContinent"**. The metamodel of the new models contains the same attributes as **"QTSContinent"**, but its values will be qualitative. In other to translate the quantitive values to qualitative ones you should follow these rules[3]:
    - Population (Millions of People)
        - *Low*: < 80
        - *Medium*: > 80 and < 500
        - *High*: > 500
    - GDP (Millions of Dollars)
        - *Low*: < 100000
        - *Medium*: > 100000 and < 1000000
        - *High*: > 1000000
    - Cost of Living (Dollars)
        - *Low*: < 500
        - *Medium*: > 500 and < 2000
        - *High*: > 2000
    - Poverty Line (%)
        - *Low*: < 16
        - *Medium*: > 16 and < 28
        - *High*: > 28

Consider the following file[4] as input (sample) for your transformations.

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:Continents="platform
    :/resource/<YourProjectNameHere>/Continents.ecore">
 <Continents:Continent name="Asia">
    <Country name="China" population="1350" costOfLiving="731.14" gdp="8360000"
    povertyLine="13.4" />
    <Country name="India" population="1220" costOfLiving="452.1" gdp="1840000"
    povertyLine="29.8" />
    <Country name="Russia" population="142.5" costOfLiving="686.16" gdp="2009999.99"
    povertyLine="12.7" />
  </Continents:Continent>
  <Continents:Continent name="Africa">
    <Country name="Nigeria" population="174.51" costOfLiving="731.68" gdp="262610"
    povertyLine="70" />
```

[3]This classification is only for the academic purpose of this assignment

[4]Add your project name to the XMI url. Most of the values of this sample input file were extracted from `http://www.nationmaster.com/`. In some cases the value was randomly generated because the real value was not available.

```
    <Country name="South Africa" population="48.6" costOfLiving="1400.01" gdp="384310"
    povertyLine="31.3" />
      <Country name="Kenya" population="44.04" costOfLiving="481.21" gdp="37340"
    povertyLine="50" />
    </Continents:Continent>
    <Continents:Continent name="Europe">
      <Country name="Germany" population="81.15" costOfLiving="2851.85" gdp="3400000"
    povertyLine="15.5" />
      <Country name="France" population="65.95" costOfLiving="2761.99" gdp="2610000"
    povertyLine="7.8" />
      <Country name="Belgium" population="10.44" costOfLiving="2564.89" gdp="483710"
    povertyLine="15.2" />
    </Continents:Continent>
    <Continents:Continent name="North America">
      <Country name="United States" population="316.67" costOfLiving="3258.85" gdp
    ="15680000" povertyLine="15.1" />
      <Country name="Mexico" population="116.22" costOfLiving="729.94" gdp="1180000"
    povertyLine="51.3" />
      <Country name="Cuba" population="11.06" costOfLiving="25.05" gdp="60810"
    povertyLine="12.5" />
    </Continents:Continent>
    <Continents:Continent name="South America">
      <Country name="Brazil" population="201.01" costOfLiving="757.92" gdp="2250000"
    povertyLine="21.4" />
      <Country name="Argentina" population="42.61" costOfLiving="1018.58" gdp="470530"
    povertyLine="30" />
      <Country name="Bolivia" population="10.46" costOfLiving="564.23" gdp="27040"
    povertyLine="49.6" />
    </Continents:Continent>
  </xmi:XMI>
```

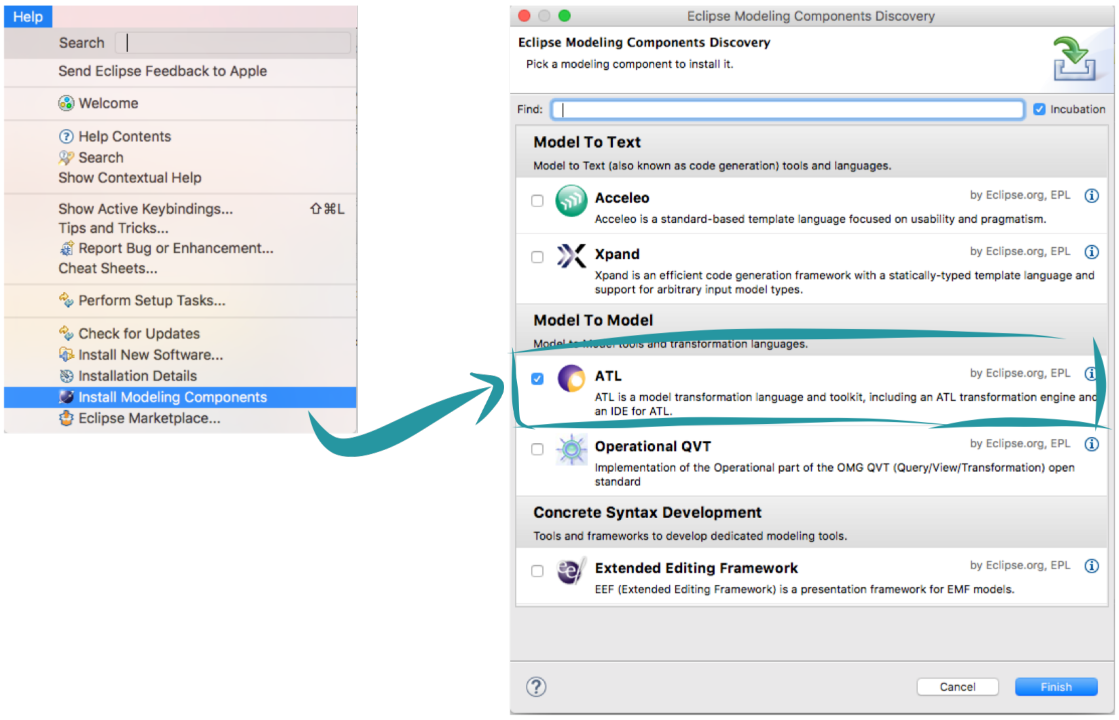
Preparing your Development Environment

In this assignment we will use the *Eclipse Modelling Tools* package of Eclipse. The Modeling package provides tools and runtimes for building model-based applications. For this assignment we are going to use the ATL[5] model transformation language and toolkit plugin.

Requirements

- A JDK 1.8 or higher.
- Download **Eclipse Modelling Tools**: `http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/neonr`
- Install **ATL** plugin. Start this Eclipse and click **Menu Help** > **Install modeling components**, and then select **ATL**. Follow the steps to install ATL tools for Eclipse.

[5] **ATL**: http://www.eclipse.org/atl/



Eclipse Modeling Components Discovery

Pick a modeling component to install it.

Model To Text

Model to Text (also known as code generation) tools and languages.

- [ ] **Acceleo** — by Eclipse.org, EPL
  Acceleo is a standard-based template language focused on usability and pragmatism.
- [ ] **Xpand** — by Eclipse.org, EPL
  Xpand is an efficient code generation framework with a statically-typed template language and support for arbitrary input model types.

Model To Model

Model to Model tools and transformation languages.

- [✓] **ATL** — by Eclipse.org, EPL
  ATL is a model transformation language and toolkit, including an ATL transformation engine and an IDE for ATL.
- [ ] **Operational QVT** — by Eclipse.org, EPL
  Implementation of the Operational part of the OMG QVT (Query/View/Transformation) open standard

Concrete Syntax Development

Tools and frameworks to develop dedicated modeling tools.

- [ ] **Extended Editing Framework** — by Eclipse.org, EPL
  EEF (Extended Editing Framework) is a presentation framework for EMF models.

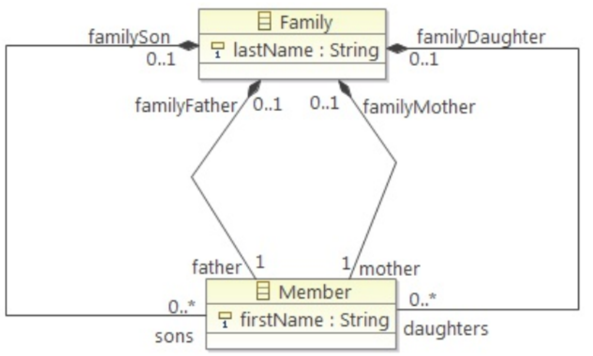
After installation, restart Eclipse. Proceed with the tutorial in order to fully understand the concepts.

Tutorial: Families to Persons

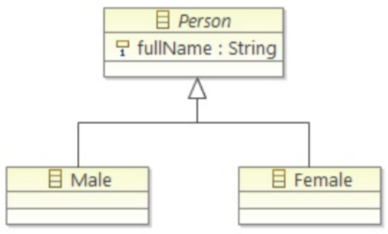
The objective of this tutorial is to become familiar with EMF metamodels and the ATL transformation language. This tutorial is adapted from `http://wiki.eclipse.org/ATL/Tutorials`. In this tutorial, you will perform a transformation from a **list of families** to a **list of persons**. On one side (the source), we have a list of families. Each family has a last name, contains a father, a mother, and a number of sons and daughters (0 or more), all with a first name. We want to transform this list into a new list of persons (the target). This means that each member of the family will become a person, without differentiating between parents and children, and with no link between members of the same family (except a part of their name). In the end, we only have a person with his/her full name (*first name* and *last name*), who can be male or female. Below you can find the EMF (Ecore) metamodels for **families** and **persons**.

Step 1: Create a new ATL project

- To create a new *ATL project*, you need to go to **File** > **New** > **Other**.
- Then select **ATL** > **ATL Project...**, and click the **Next** button.

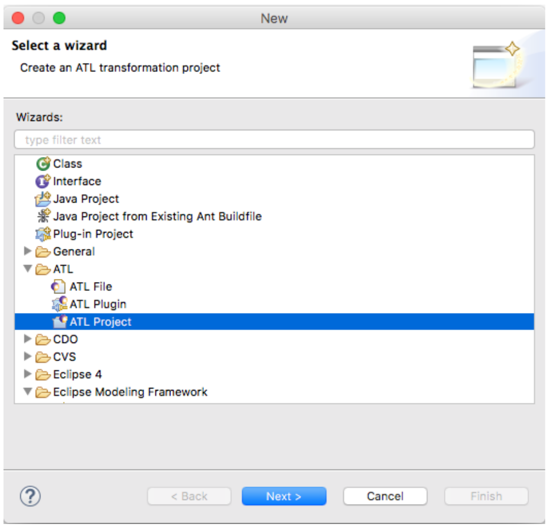


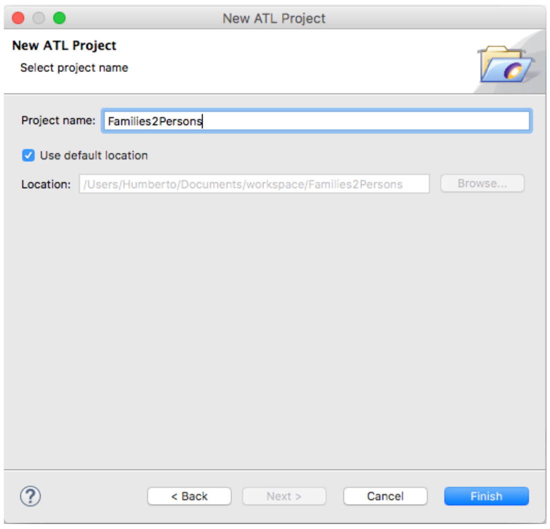
Families metamodel



Persons metamodel

- Type a name for the project (”**Families2Persons**” for our example), and click **Finish**.





New ATL Project

Select project name

Project name: Families2Persons

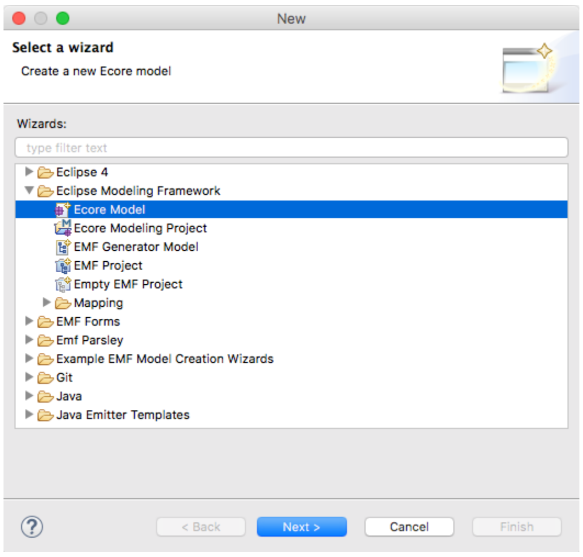
Use default location

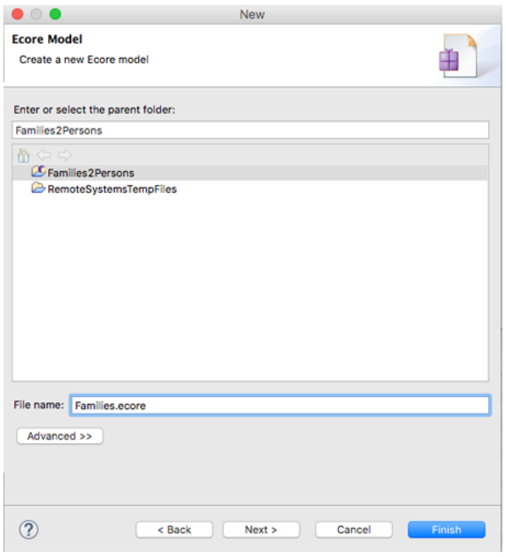
Location: /Users/Humberto/Documents/workspace/Families2Persons

The project should now appear on your projects list. The User Guide also provides a detailed section for the creation of a new ATL project.

Step 2: Create the metamodels Now that our project is ready to use, we can fill it. Our first files are the metamodels of a family and a person, both of which will be expressed in the Ecore language

- To create the Ecore file, go to **File** > **New** > **Other**..., and then select **Eclipse Modeling Framework** > **Ecore Model** and click **Next**.
- Select your **Families2Persons** project on the list, enter a name for the file (**Families**.ecore for instance), and click **Finish**. An empty file is added to your project.



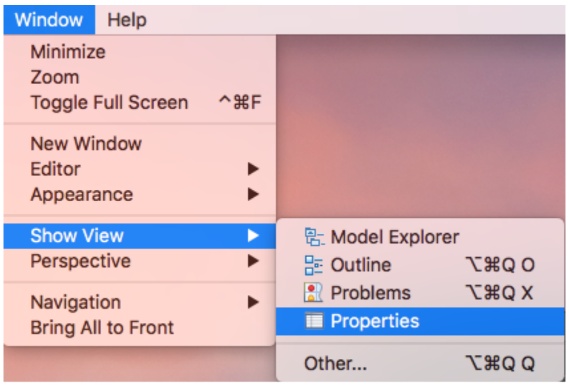


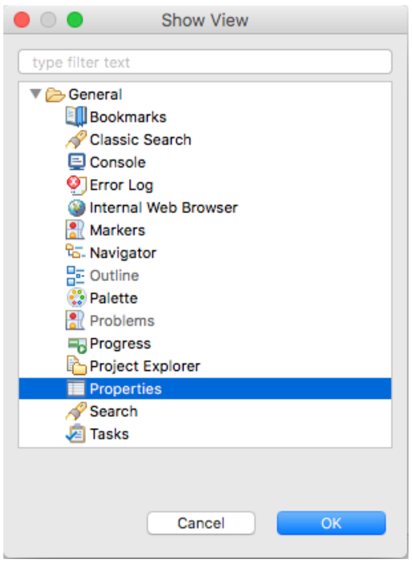
Repeat this task for the **Persons.ecore** metamodel.

Step 3: Define Families Metamodel

As we saw in the Objective part, a family has a last name, and a father, a mother, sons and daughters with a first name. That is what we need to tell to the Families.ecore file.

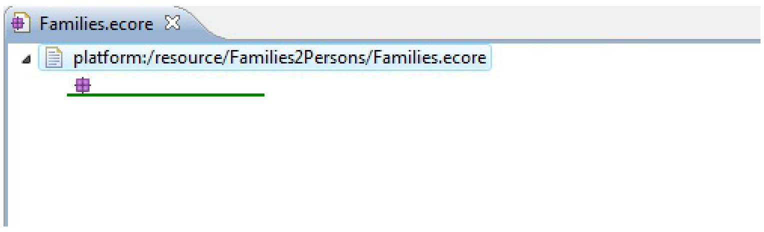
- Open it with the default editor (Sample Ecore Model Editor). We will also need the Properties view, so if it is not already opened, you can show it by going on **Window** > **Show View** > **Other...**, selecting **General** > **Properties** and clicking **OK**.

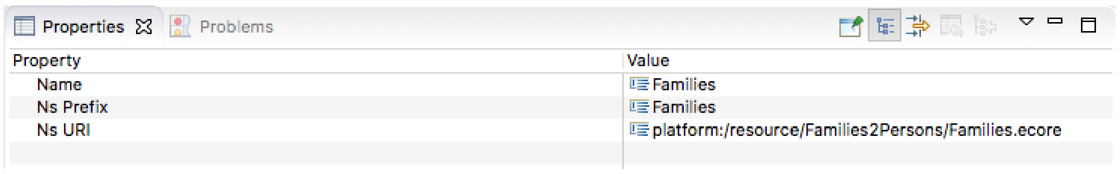




The Families.ecore file comes in the form of a tree. The root should be: **platform:/resource/Families2Persons/Families.ecore**. If you expand it, there is an empty node under it.

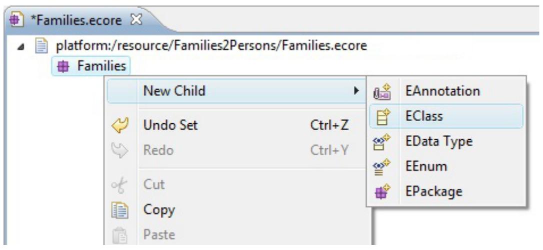
- Click on it, and in the **Properties** view, enter **"Families"** in the value of the **"Name"** and **"Ns Prefix"** properties.
- Set **"platform:/resource/Families2Persons/Families.ecore"** in the value of **"Ns URI"** property. This node is where we are going to put everything that makes a family.

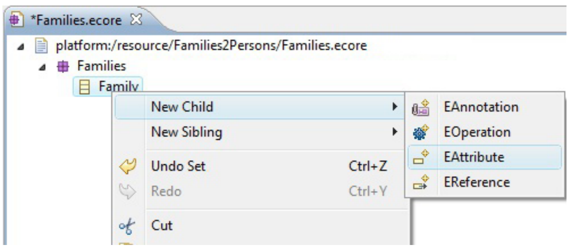




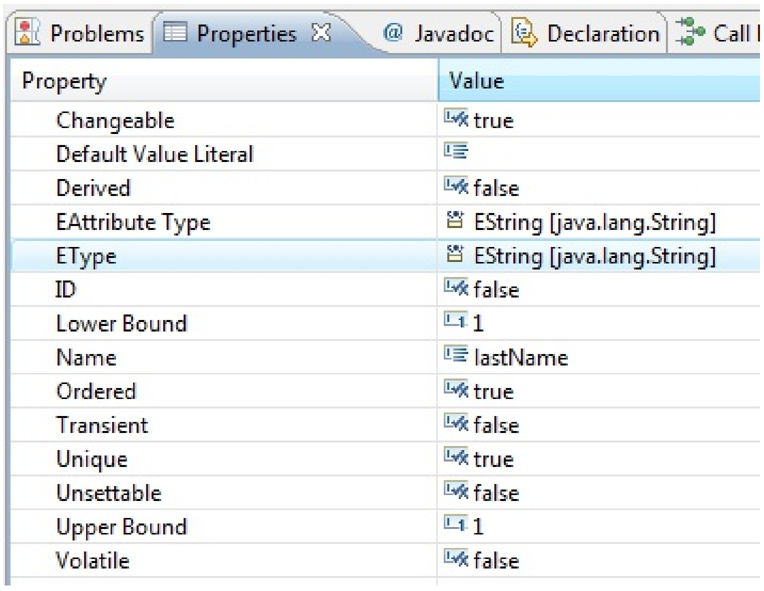
Property	Value
Name	Families
Ns Prefix	Families
Ns URI	platform:/resource/Families2Persons/Families.ecore

- So first we create a class **"Family"**, by right clicking on the **Families node**, and clicking on **New Child** > **EClass**. You can name it the same way you named the node **Families** above.
- Then we give it an attribute (**New Child** > **EAttribute**) and name it **"lastName"**.





- We want to have one and only one last name per family, so we control its multiplicity: set 1 for the **lower bound** (that should be set to 0 by default), and 1 for the **upper bound** (that should already be 1). These bounds can be set the same way than the name, but on the **Lower Bound** and **Upper Bound properties**. We can specify a type for this attribute, and we want it to be a **string**. So in the EType property, search for the **EString type**.



Property	Value
Changeable	true
Default Value Literal	
Derived	false
EAttribute Type	EString [java.lang.String]
EType	EString [java.lang.String]
ID	false
Lower Bound	1
Name	lastName
Ordered	true
Transient	false
Unique	true
Unsettable	false
Upper Bound	1
Volatile	false

At this moment, we have a family with its last name. Now we need members for this family. Therefore we are going to create another class (as we created the Family class): **“Member”**.

This class will be a Families node’s child, as the other Family class. These members have a first name, so we add an attribute **“firstName”** of type EString, and again a member has one and only one first name (see above if you don’t remember how to create an attribute, name it, give it a type and change its multiplicity).

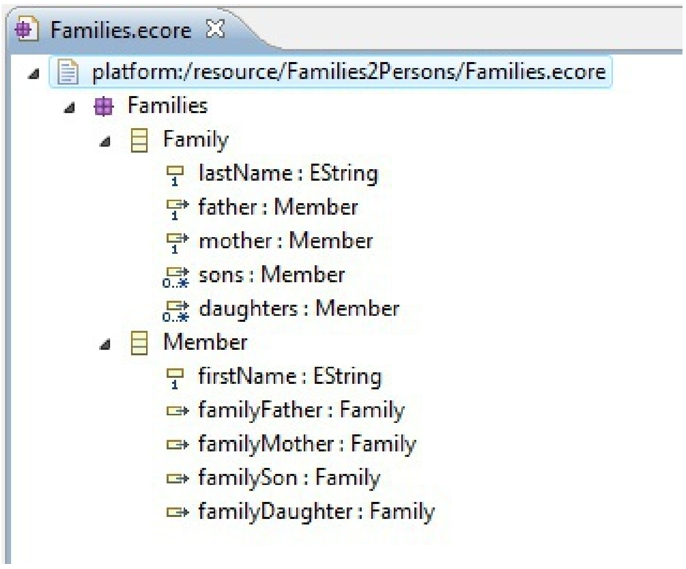
Now we have to make the links between the family and the members. For this purpose, you have to create children of the Family of the type **EReference**. Name these references ”father”, ”mother”, ”sons” and ”daughters”. They will have the EType **Member**. About the multiplicity, we have one father and one mother for one family (so upper and lower bounds set to 1), but we can have as many sons and daughters as we want, even 0 (so lower bound set to 0, and upper bound set to -1, which means *). And at last, put their Containment property to true so that they can contain members.

Once these attributes are created and configured, we do the same for the Member class. It also needs references towards the Family class. Just add 4 EReferences to the Member class: ”familyFather”, ”familyMother”, ”familySon” and ”familyDaughter” with EType **Family**. This time, each reference should have its multiplicity set to 0..1 (it is by default), because a member is either a father, or a mother, or a son, or a daughter, so the reference that is defined for a member shows its role in the family. Then, in order to tell which member refer to which family member, set their EOpposite field to their reference in the Family class (for example, **familyFather** refers to the **father** reference of the Family class).

And here we are with the metamodel for our families!

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
```

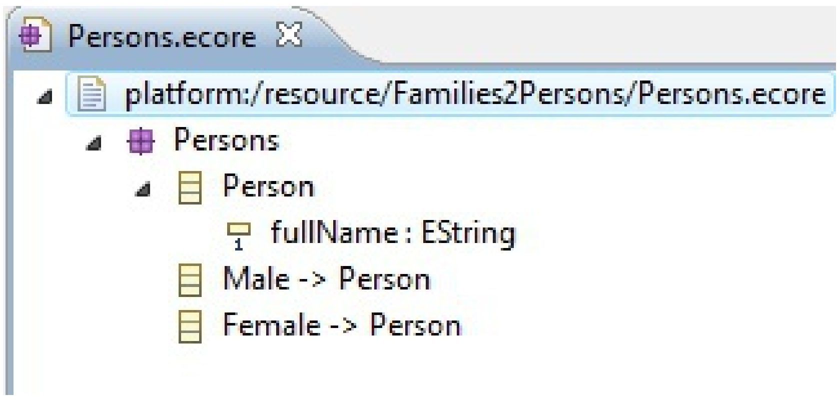
```
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="Families"
  nsURI="platform:/resource/Families2Persons/Families.ecore" nsPrefix="Families">
<eClassifiers xsi:type="ecore:EClass" name="Family">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="lastName" lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="father" lowerBound="0"
      eType="#//Member" containment="true" eOpposite="#//Member/familyFather"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="mother" lowerBound="0"
      eType="#//Member" containment="true" eOpposite="#//Member/familyMother"/>
      <eStructuralFeatures xsi:type="ecore:EReference" name="sons" upperBound="-1"
      eType="#//Member" containment="true" eOpposite="#//Member/familySon"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="daughters" upperBound="-1"
        eType="#//Member" containment="true" eOpposite="#//Member/familyDaughter"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Member">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="firstName" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="familyFather"
      eType="#//Family" eOpposite="#//Family/father"/>
<eStructuralFeatures xsi:type="ecore:EReference" name="familyMother"
    eType="#//Family"
        eOpposite="#//Family/mother" />
    <eStructuralFeatures xsi:type="ecore:EReference" name="familySon"
eType="#//Family"
        eOpposite="#//Family/sons"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="familyDaughter"
eType="#//Family"
        eOpposite="#//Family/daughters"/>
  </eClassifiers>
</ecore:EPackage>
```



Step 4: Define Persons Metamodel
The principle is the same for the target metamodel, but less complicated:

- Open the **Persons.ecore** file, and set the following properties:
  - Enter **"Persons"** in the value of the **"Name"** and **"Ns Prefix"** properties.
  - Enter **"platform:/resource/Families2Persons/Persons.ecore"** in the value of **"Ns URI"** property.
- Add a class **"Person"** to it, with one attribute: **"fullName"** of **EType EString** and **multiplicity 1..1**.
- Set the **Abstract** attribute of the **Person** class to **"true"**. We need to do this because we won't directly implement this class, but two other subclasses: **"Male"** and **"Female"**, according to who was the person in the family, a man or a woman
- Create these two classes (**"Male"** and **"Female"**) at the same level as **"Person"**. We make them subclasses of **"Person"** by setting their ESuper Types property to **"Person"**.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="Persons"
    nsURI="platform:/resource/Families2Persons/Persons.ecore" nsPrefix="Persons">
  <eClassifiers xsi:type="ecore:EClass" name="Person" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="fullName" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
</eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Male"
eSuperTypes="#//Person"/>
  <eClassifiers xsi:type="ecore:EClass" name="Female"
eSuperTypes="#//Person"/>
</ecore:EPackage>
```

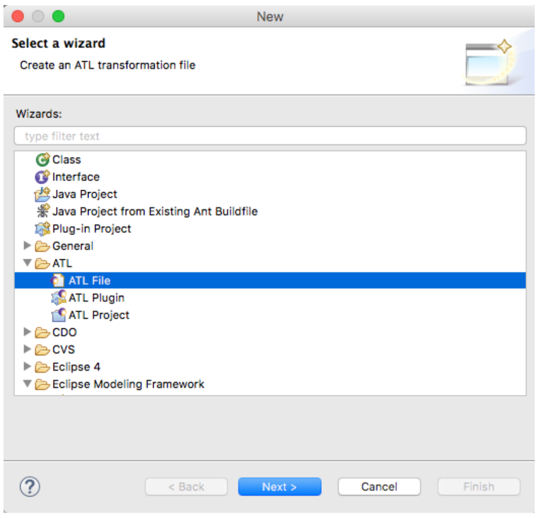

Persons.ecore

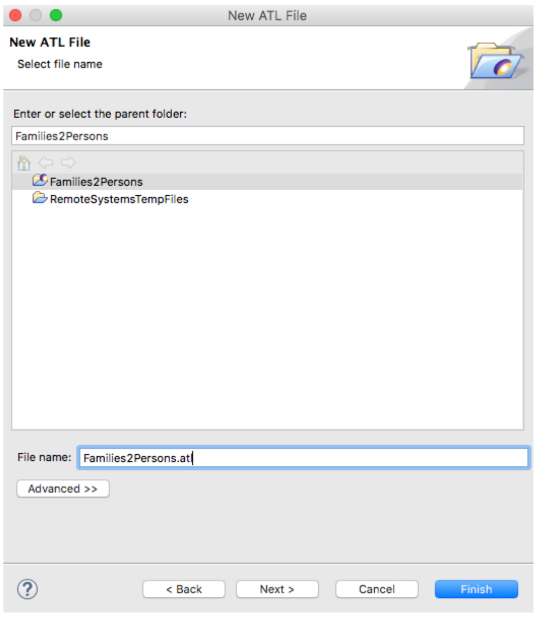
```
platform:/resource/Families2Persons/Persons.ecore
  Persons
    Person
      fullName : EString
    Male -> Person
    Female -> Person
```

Step 5: ATL transformation code

Now that we have represented what we have (Families, the **source**) and what we want to obtain (Persons, the target), we can concentrate on the core of the transformation: the ATL code. This code is going to match a part of the source with a part of the target. What we want in our example, is to take each **member** of each **family**, and transform it into a person. That implies merging the first and last name into a full name, determining whether it's a **Male** or a **Female**, and copy these pieces of information into a **Person** object.

- We first need a file to put this code into. So create a new ATL file. Right-click the project folder > **New** > **Other...**, and then **ATL** > **ATL File**. Then, click **Next** >
- Name it **"Families2Persons.atl"** for instance, and then click **Finish**.





- When asked to open the **ATL perspective**, click **Yes**.

In the new Families2Persons.atl file.

- First we add two lines at the top of the file, one for each metamodel, so that the editor can use the auto-completion and documentation when we type in some code concerning the two metamodels:

```
-- @path Families=/Families2Persons/Families.ecore
-- @path Persons=/Families2Persons/Persons.ecore
```

- Then we tell ATL that we have families in and we want persons out (add this after module declaration):

```
-- @path Families=/Families2Persons/Families.ecore
-- @path Persons=/Families2Persons/Persons.ecore
module Families2Persons;
create OUT: Persons from IN: Families;
```

- Now we must define some helpers:

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

helper context Families!Member def: familyName: String =
       Set{self.familyFather, self.familyMother,
          self.familySon, self.familyDaughter}
                ->any(f|not f.oclIsUndefined()).lastName;
```

These helpers will be used in the rules that we will see below.

- The first one is called on a member of a family (context Families!Member), gives us a boolean (: Boolean), and tells us whether the member is a female or not, by verifying if the familyDaughter or familyMother reference is defined or not.
- The second one is also called on a member of a family, this time gives us a string (: String) and returns the last name of the member. It must look for it in every reference to the family, to see which one is defined (familyFather, familyMother, familySon or familyDaughter)

And finally, we add two rules creating male and female persons from members of families:

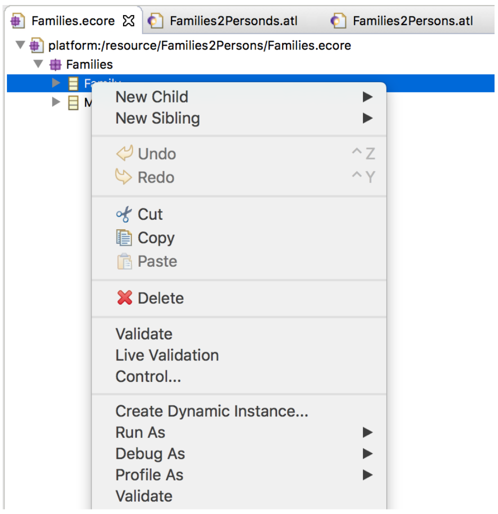
```
rule Member2Male {
    from
      s: Families!Member (not s.isFemale)
    to
      t: Persons!Male (
        fullName <- s.firstName + ’ ’ + s.familyName
      )
}

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

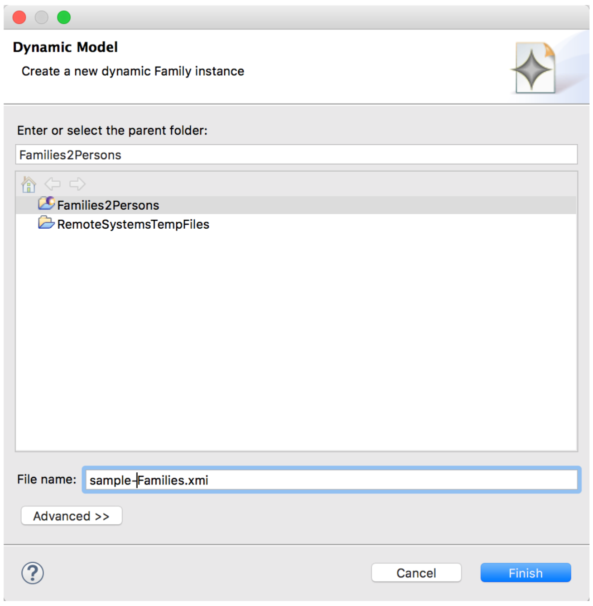
Each rule will be called on the object that respect the filter predicate in the **from** part. For instance, the first rule takes each member of each families (from s: Families!Member) that is not a female (using the helper we described above, not s.isFemale()). And then it creates a male person (to t: Persons!Male) and set its **fullName** attribute to the first name of the member followed by its last name (using the helper familyName we saw above). The principle is the same for the second rule, whereas this time it takes only the female members.

Step 6: Create a sample model

The transformation is ready to be used, we just need a sample model to run it on. Follow the next actions to create a sample families model.

- In the Families metamodel, right-click the **Family** class, and select **Create Dynamic Instance...**
- Name the file **"sample-Families.xmi"**, for instance, and click **Finish**.





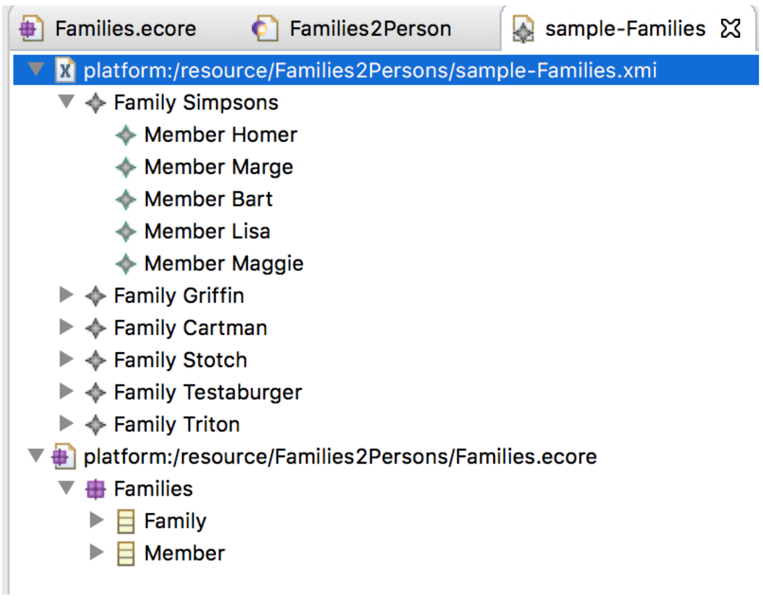
Enter or select the parent folder: Families2Persons

File name: sample-Families.xmi

- Open the new file in a text or XML editor.
- Edit the XML source to contain the following:

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:Families="
    platform:/resource/Families2Persons/Families.ecore">
  <Families:Family lastName="Simpsons">
    <father firstName="Homer"/>
    <daughters firstName="Lisa"/>
    <daughters firstName="Maggie"/>
    <mother firstName="Marge"/>
    <sons firstName="Bart"/>
</Families:Family>
<Families:Family lastName="Griffin">
    <father firstName="Peter"/>
    <sons firstName="Stewie"/>
    <sons firstName="Chris"/>
    <mother firstName="Lois"/>
    <daughters firstName="Meg"/>
</Families:Family>
<Families:Family lastName="Cartman">
    <sons firstName="Eric"/>
    <mother firstName="Liane"/>
</Families:Family>
<Families:Family lastName="Stotch">
    <father firstName="Stephen"/>
    <sons firstName="Butters"/>
    <mother firstName="Linda"/>
</Families:Family>
<Families:Family lastName="Testaburger">
    <mother firstName="Alice"/>
    <daughters firstName="Wendy"/>
</Families:Family>
<Families:Family lastName="Triton">
    <father firstName="King"/>
    <daughters firstName="Ariel"/>
</Families:Family>
</xmi:XMI>
```

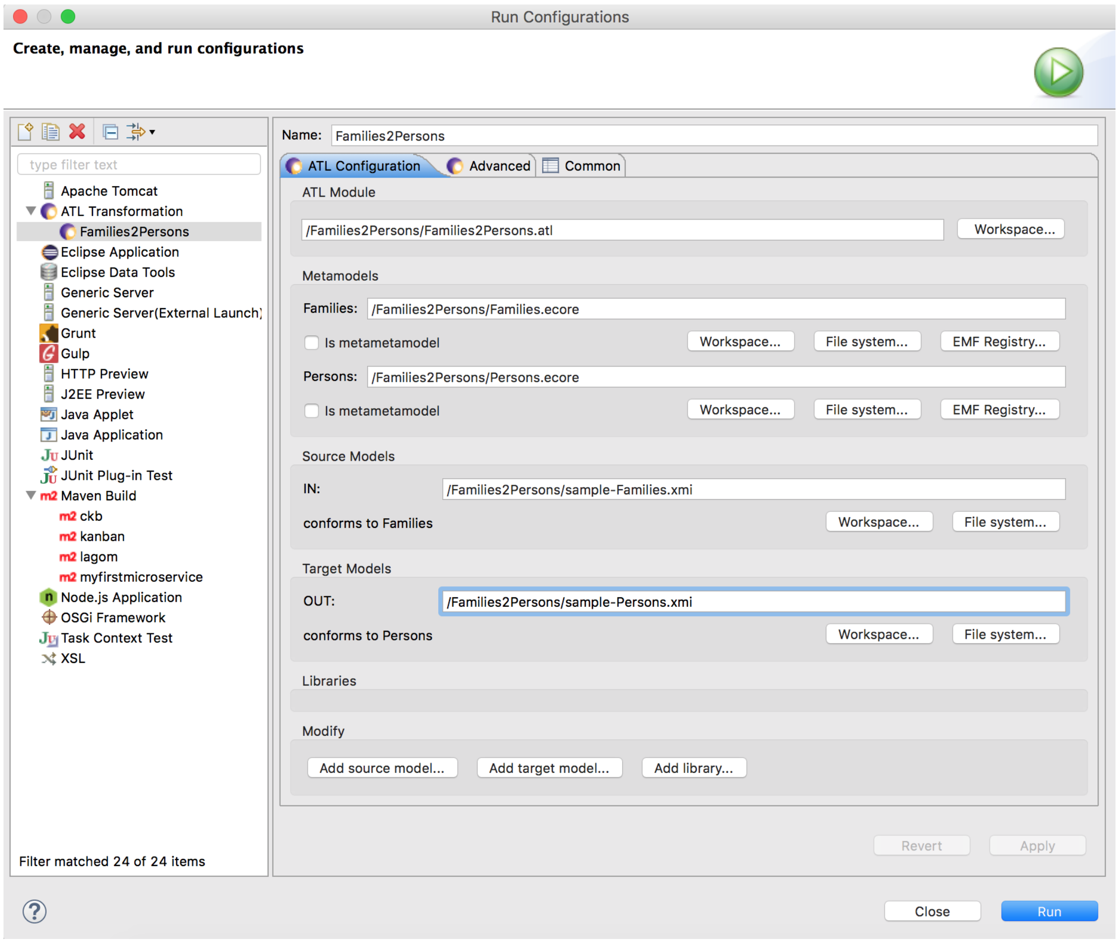
- Right-click the **"sample-Families.xmi"** file in the Project Explorer panel, and select **Open With** > **Sample Reflective Ecore Model Editor**. You should now see the model as displayed here.



```
▼ platform:/resource/Families2Persons/sample-Families.xmi
    ▼ Family Simpsons
        Member Homer
        Member Marge
        Member Bart
        Member Lisa
        Member Maggie
    ▶ Family Griffin
    ▶ Family Cartman
    ▶ Family Stotch
    ▶ Family Testaburger
    ▶ Family Triton
▼ platform:/resource/Families2Persons/Families.ecore
    ▼ Families
        ▶ Family
        ▶ Member
```

Step 7: Setup run configuration

We have everything we need to execute the transformation, but there is one more step before we launch it, at least the first time: we have to configure the launch parameters.



Name: Families2Persons

ATL Module: /Families2Persons/Families2Persons.atl

Metamodels:
- Families: /Families2Persons/Families.ecore — Is metametamodel [ ]
- Persons: /Families2Persons/Persons.ecore — Is metametamodel [ ]

Source Models:
- IN: /Families2Persons/sample-Families.xmi — conforms to Families

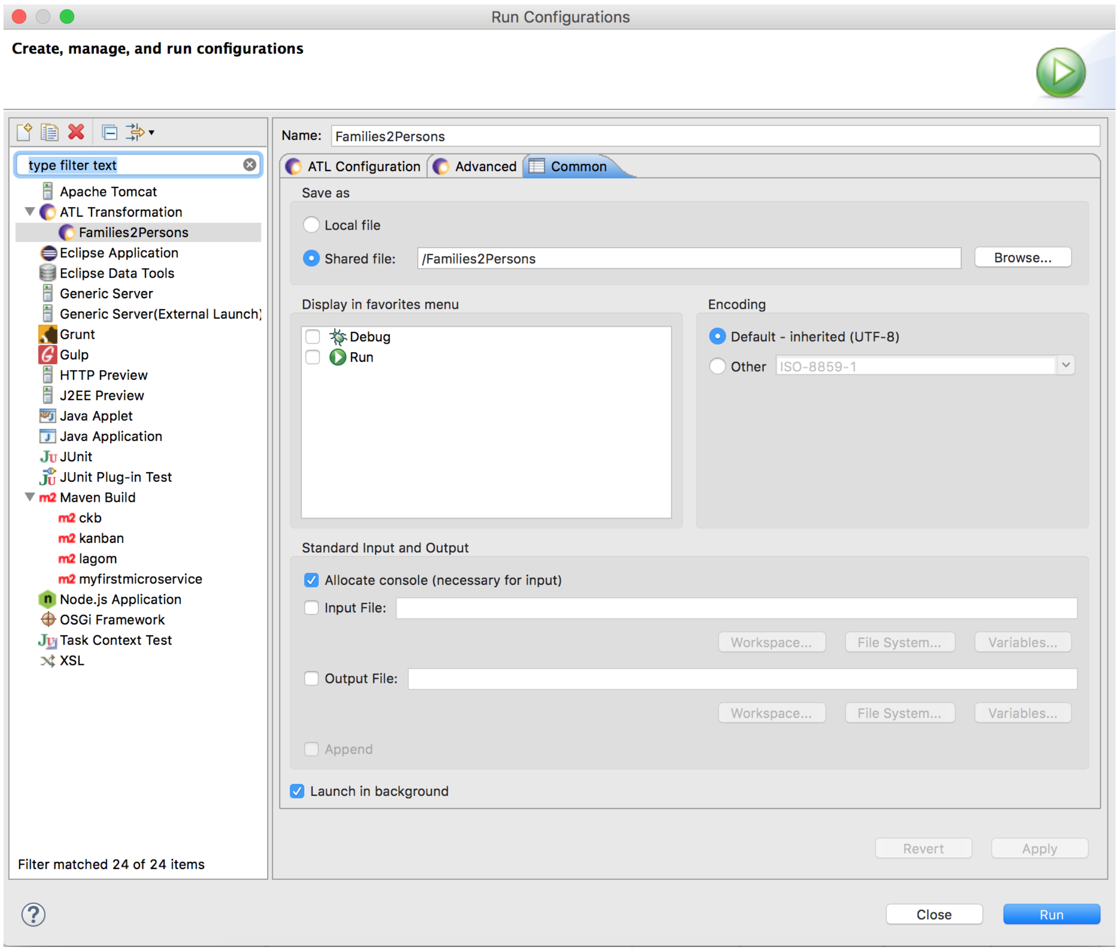
Target Models:
- OUT: /Families2Persons/sample-Persons.xmi — conforms to Persons

Libraries

Modify

- When you are in the ATL file (**Families2Persons.atl**), click on the menu **Run** > **Run** or click on the icon **Run** of the Toolbar, and then select **Run Configurations**
- A dialog opens. Add the following: the ATL module (our transformation file, Families2Persons.atl), the metamodels (Families.ecore and Persons.ecore), and to complete the form like this:
  - The **Source Models** (IN:, conforms to Families) part is the model we want to transform, that is to say our **sample-Families.xmi**; browse the workspace to add it.
  - The **Target Models** (Out:, conforms to Persons) part is the model to be generated; browse the workspace to find your project and enter a name for the file (say **"sample-Persons.xmi"**).

- A useful option can be found in the **Common** tab of the page: we can save our configuration so that ATL can find it the next time we would want to run it or if the project is exported.
  - If you check Shared file and browse within your project, you can save this configuration in a file (it will be named **"Families2Persons.launch"**).



Run Configurations

Create, manage, and run configurations

Name: Families2Persons

Tabs: ATL Configuration | Advanced | Common

Save as
- Local file
- [✓ Shared file:] /Families2Persons — Browse...

Display in favorites menu
- ☐ Debug
- ☐ Run

Encoding
- [✓ Default - inherited (UTF-8)]
- Other: ISO-8859-1

Standard Input and Output
- ☑ Allocate console (necessary for input)
- ☐ Input File:
- ☐ Output File:
- ☐ Append

☑ Launch in background

Filter matched 24 of 24 items

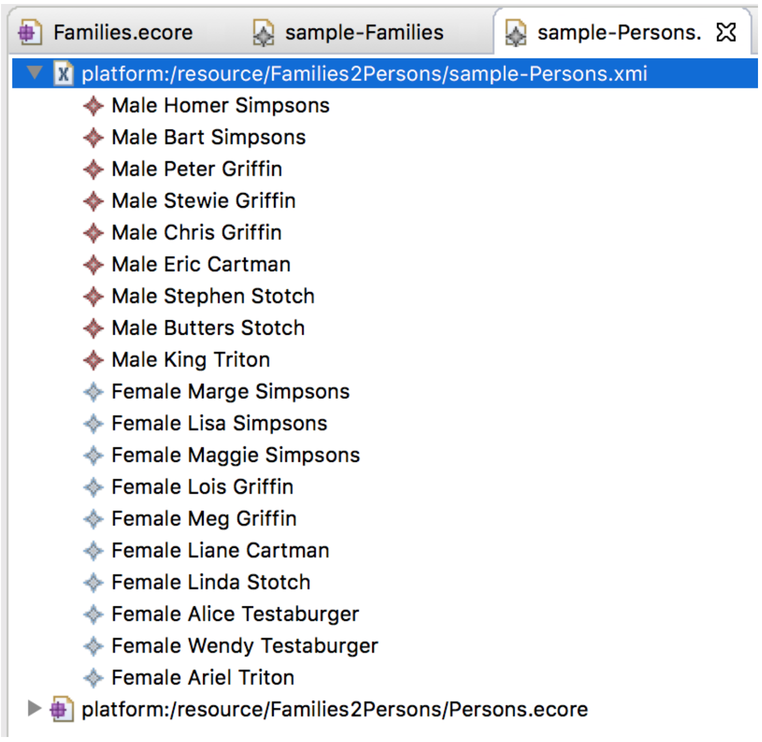
Step 8: Running the transformation

At last we can run the transformation by clicking **Run** menu item or button on the toolbar. A file is then generated, named **sample-Persons.xmi**, and containing the list of your family members transformed into persons.

Here is close to what you should get if you open it with a text editor:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:Persons="platform:/
    resource/Families2Persons/Persons.ecore">
  <Persons:Male fullName="Homer Simpsons"/>
  <Persons:Male fullName="Bart Simpsons"/>
  <Persons:Male fullName="Peter Griffin"/>
  <Persons:Male fullName="Stewie Griffin"/>
  <Persons:Male fullName="Chris Griffin"/>
  <Persons:Male fullName="Eric Cartman"/>
  <Persons:Male fullName="Stephen Stotch"/>
  <Persons:Male fullName="Butters Stotch"/>
  <Persons:Male fullName="King Triton"/>
  <Persons:Female fullName="Marge Simpsons"/>
  <Persons:Female fullName="Lisa Simpsons"/>
  <Persons:Female fullName="Maggie Simpsons"/>
  <Persons:Female fullName="Lois Griffin"/>
  <Persons:Female fullName="Meg Griffin"/>
  <Persons:Female fullName="Liane Cartman"/>
  <Persons:Female fullName="Linda Stotch"/>
  <Persons:Female fullName="Alice Testaburger"/>
  <Persons:Female fullName="Wendy Testaburger"/>
  <Persons:Female fullName="Ariel Triton"/>
</xmi:XMI>
```

The next figure shows some contents of the generated model.



```
platform:/resource/Families2Persons/sample-Persons.xmi
    Male Homer Simpsons
    Male Bart Simpsons
    Male Peter Griffin
    Male Stewie Griffin
    Male Chris Griffin
    Male Eric Cartman
    Male Stephen Stotch
    Male Butters Stotch
    Male King Triton
    Female Marge Simpsons
    Female Lisa Simpsons
    Female Maggie Simpsons
    Female Lois Griffin
    Female Meg Griffin
    Female Liane Cartman
    Female Linda Stotch
    Female Alice Testaburger
    Female Wendy Testaburger
    Female Ariel Triton
platform:/resource/Families2Persons/Persons.ecore
```

Running an ATL launch configuration is also explained on the User Guide.

More information about ATL

- `http://www.eclipse.org/atl/`
- `http://www.eclipse.org/atl/documentation/`

- `http://wiki.eclipse.org/ATL/User_Guide_-_Introduction`
- `http://www.eclipse.org/atl/documentation/basicExamples_Patterns/`
- `http://www.eclipse.org/atl/atlTransformations/`