Chapitre 1

EMF

1.1 Eclipse Modeling Framework (EMF)

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

The core Eclipse Modeling Framework (EMF) includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.

EMF (core) is a common standard for data models, many technologies and frameworks are based on. This includes server solutions, persistence frameworks, UI frameworks and support for transformations. Please have a look at the modeling project for an overview of EMF technologies.

1.2 EMF(Core)

EMF consists of three fundamental pieces :

- EMF - The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
- EMF.Edit - The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides
  - Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
  - A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.
- EMF.Codegen - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse.

Three levels of code generation are supported :

- Model - provides Java interfaces and implementation classes for all the classes in the model, plus a factory and package (meta data) implementation class.
- Adapters - generates implementation classes (called ItemProviders) that adapt the model classes for editing and display.
- Editor - produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors and serves as a starting point from which to start customizing.

All generators support regeneration of code while preserving user modifications. The generators can be invoked either through the GUI or headless from a command line.

Chapitre 2

Ecore tools

The Ecore Tools component provides a complete environment to create, edit and maintain Ecore models. This component eases handling of Ecore models with a Graphical Ecore Editor and bridges to other existing Ecore tools (Validation, Compare, generators...). The Graphical Ecore Editor implements multi-diagram support, a custom tabbed properties view, validation feedbacks, refactoring capabilities... The long-term goal is to provide the same level of services as does JDT for Java.

2.1 Architecture

EcoreTools is a modeler defined using the Eclipse Sirius project.

The graphical modeler is defined in the plugin org.eclipse.emf.ecoretools.design which contains :

- ./description/ecore.odesign : the editor specification which is interpreted by sirius.
- ./src/org/eclipse/emf/ecoretools/design/service : some small java utilities used by the editor specification.
- and other classes for specific integrations with Eclipse (wizards...)

Properties views and embedded wizards are defined in the plugin org.eclipse.emf.ecoretools.design.propert
which is generated using the EEF project.

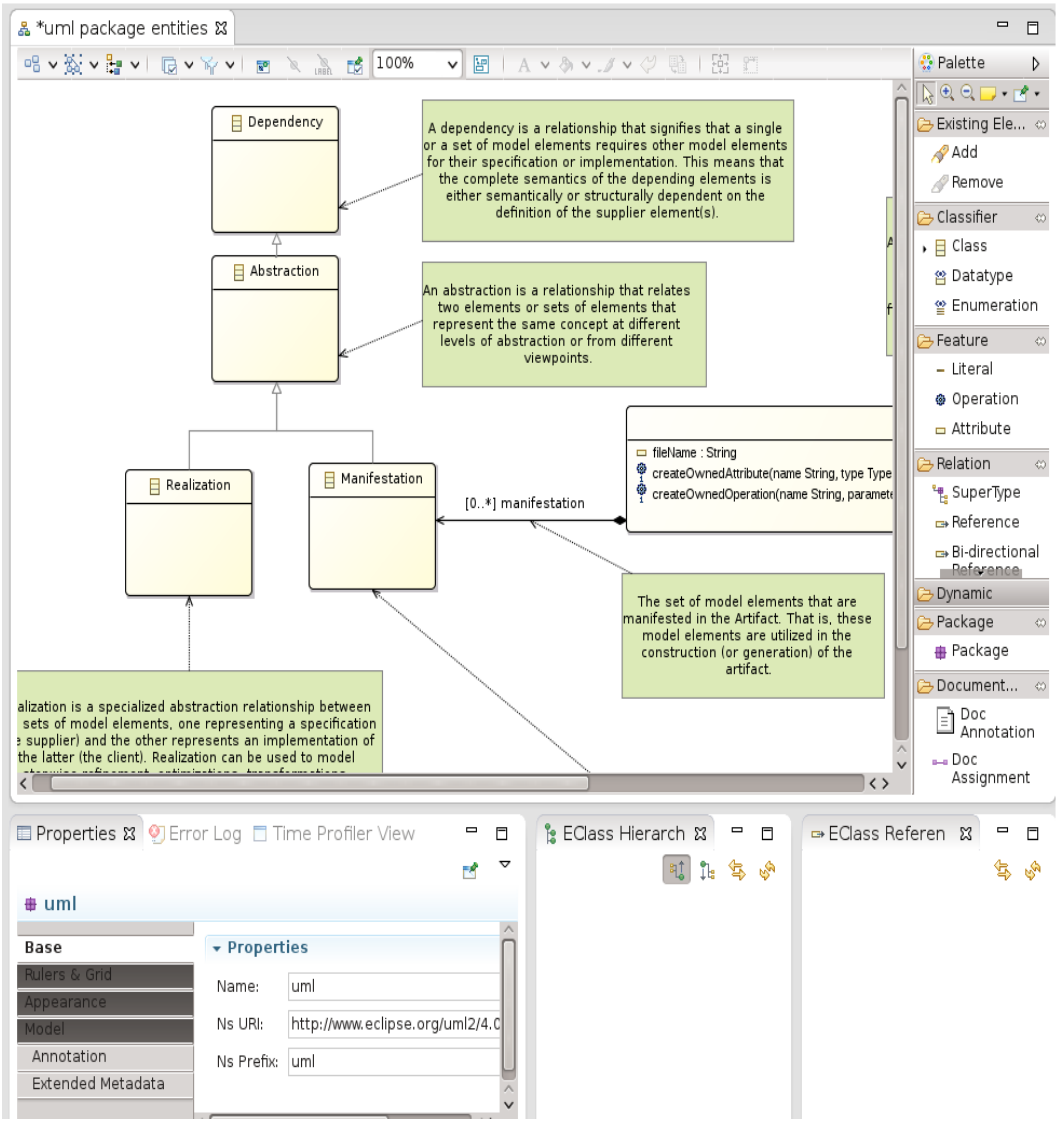


FIGURE 2.1 – Ecore Diagram Overview

2.2 Creating an Ecore Model

2.2.1 Overview

This part explains how to create your first EMF Metamodel

The instructions describe the creation of the BasicFamily metamodel. This metamodel defines 4 concepts that can be use to describe a family :

— The first concept is Family. A family has a name of type Estring.
— A family contains members of type Person. A person has a name of type EString.
— A person has 2 parents and many or no children.
— Two concepts herited from Person are Man and Woman.
— A person can have one father of type Man and/or one mother of type Woman

To sum up, our metamodel have 4 concepts

— Family
— Person
— Man
— Woman

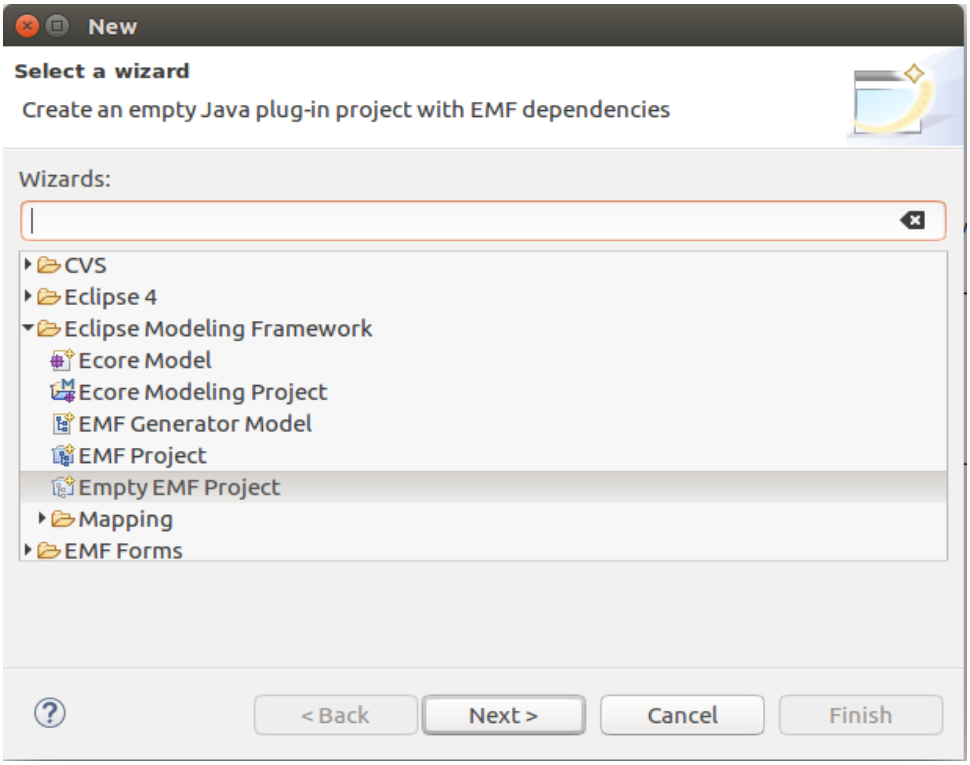
and 6 relations

— members from Family to Person
— children from Person to Person, with cardinality 0..*
— parents from Person to Person with cardinality 0..2
— father from Person to Man with cardinality 0..1
— mother from Person to Woman with cardinality 0..1
— an extend from Person for Man and Woman

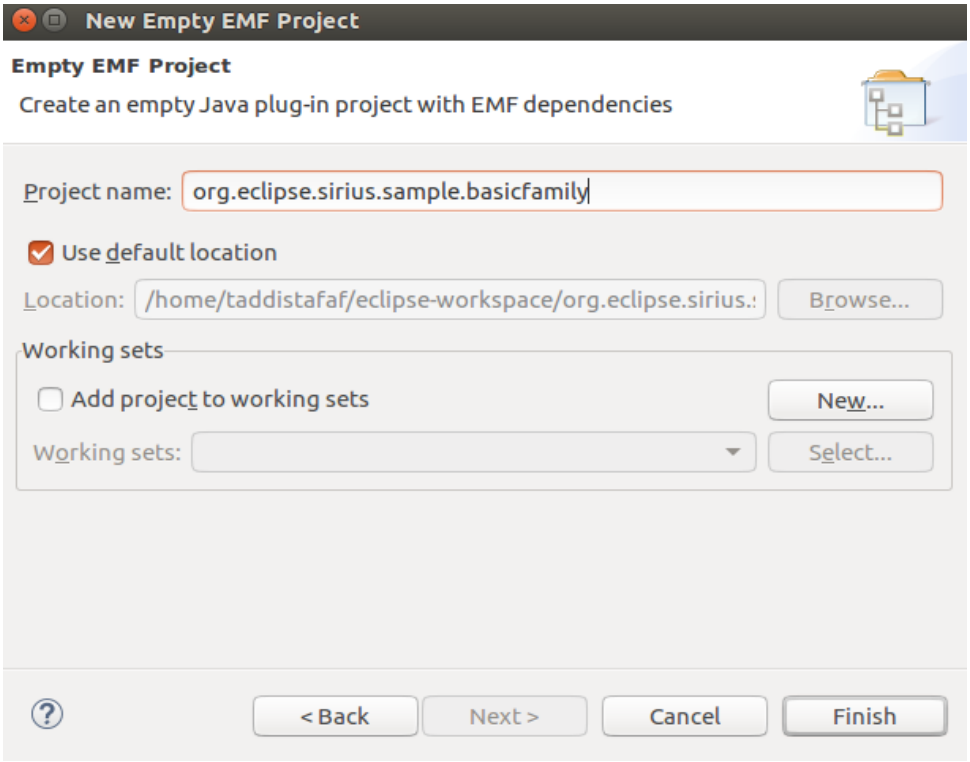
2.2.2 Create the Metamodel

We start by creating the Ecore model that will fulfill the role of metamodel. First, we create an empty EMF project.

1. In the Model Explorer View,right click on New->Other...
2. Select Empty EMF Project under Eclipse Modeling Framework and click Next.

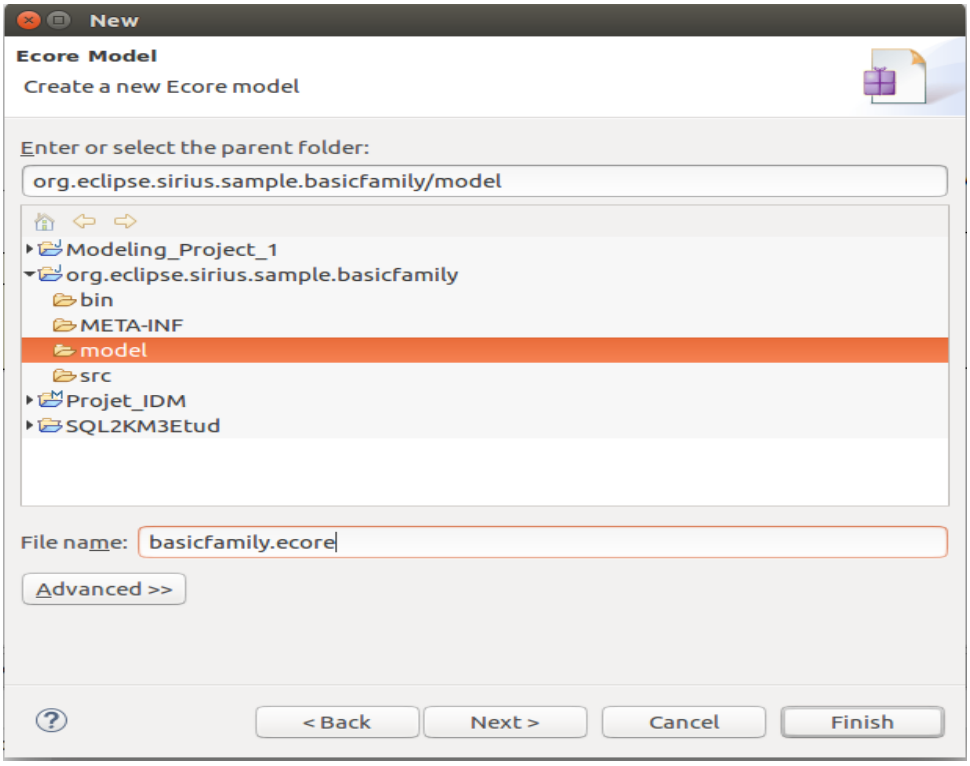


3. Choose a name for the empty EMF project and click Finish.

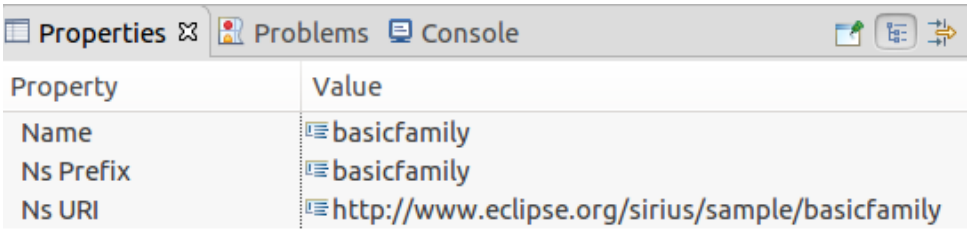


In this project we create the Ecore model.

1. Right-click the folder named model in the new project»New»Other.
2. Select Ecore Model under Eclipse Modeling Framework and click Next.
3. Choose a name for the Ecore model and click Next



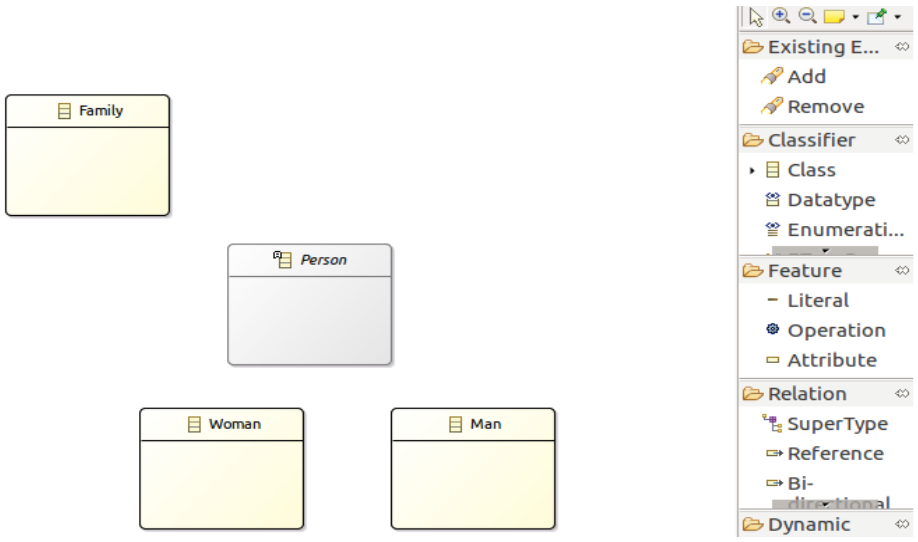
4. Open the Ecore model by double clicking it (if it is not opened).
5. Click on the small triangle in the tree-based editor to show the EPackage, which is represented by a small purple figure.
6. Right-click the small figure and select Show Properties View(or just a click).
7. Change the data in the properties view to the data shown below.



2.2.3 Create the elements of the metamodel

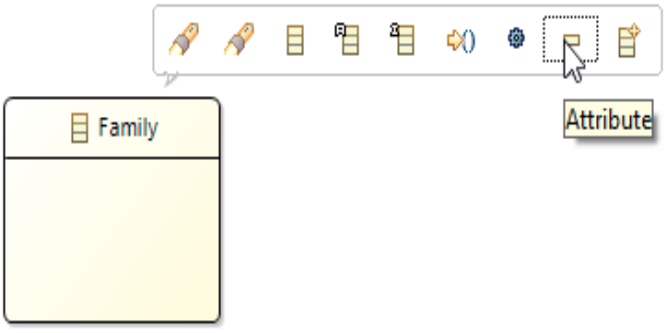
First right click on our ecore (basicfamily.ecore) > Initialize Ecore Diagram >next>Entities in a class Diagram >finish.

1. Use the Class tool in the palette to create Family, Person, Man and Woman. Use the properties view to set Person as abstract.

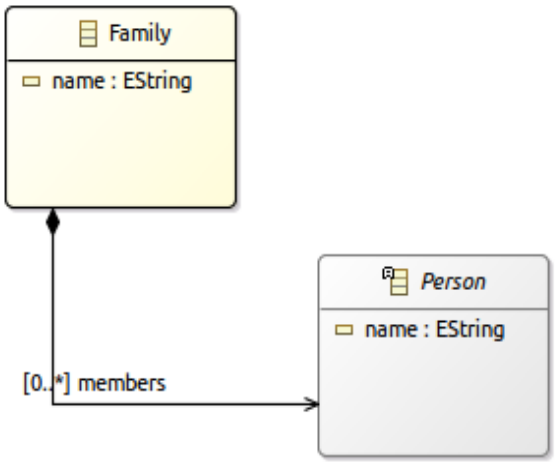


Note that you can edit the name of a model element (class, attribute, relation) directly from the diagram : just click on a selected element to activate the label edition (let some time before the selection and the click to avoid triggering a double-clik).

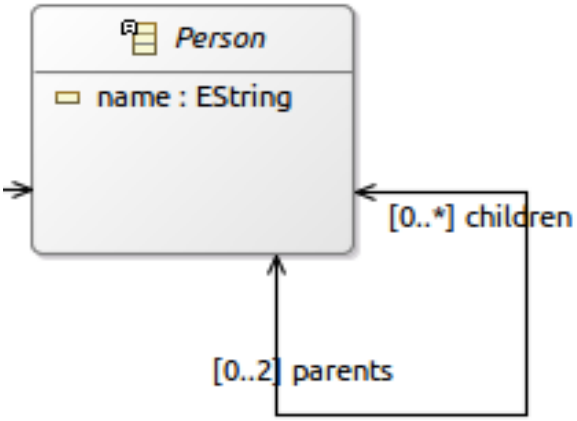
2. Use the Attribute tool in the palette, or the popup, to create an attribute named name on Family and Person.



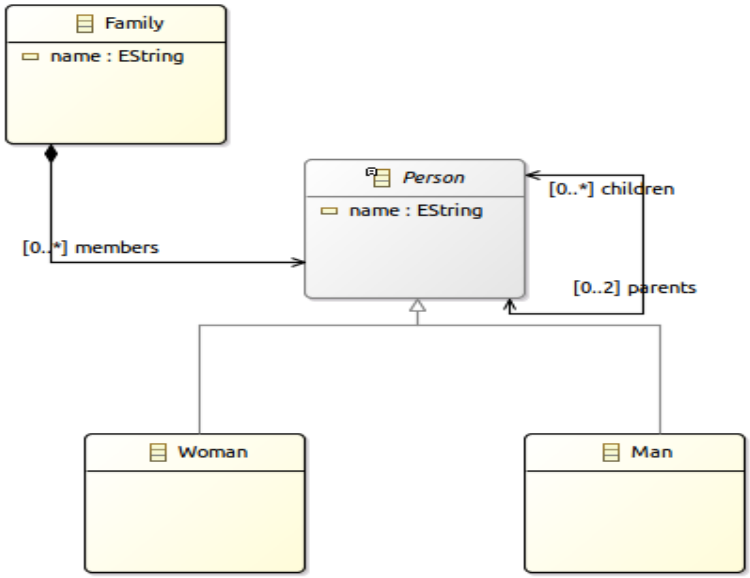
3. Use the Composition tool in the palette to create a composition relation named members between Family and Person.



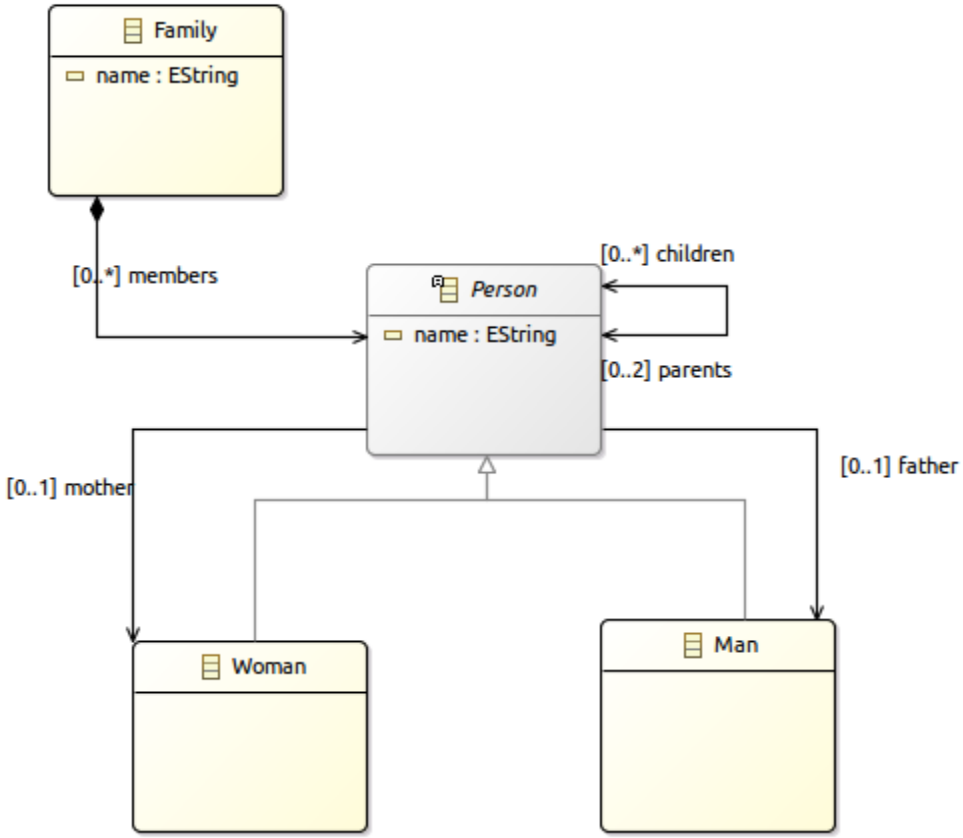
4. Use the Bi-directional Reference tool in the palette to create the relation named children and parents from Person to Person.



5. Use the SuperType tool in the palette to create the inheritance relations from Man and Woman to Person

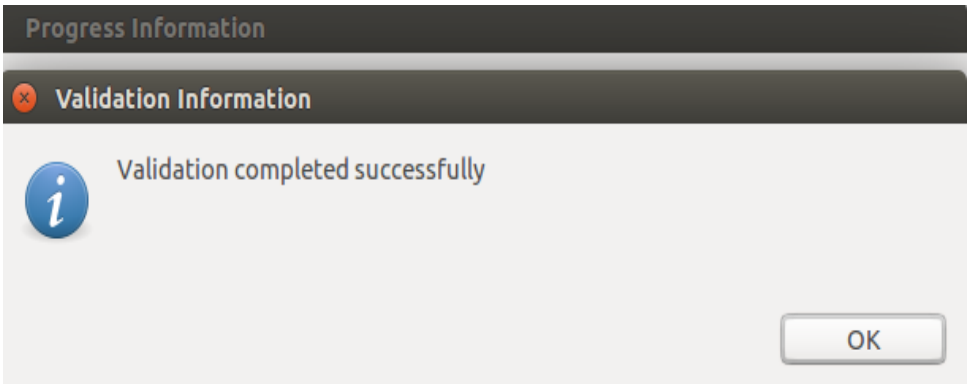


6. Use the Reference tool in the palette to create the two relations named father and mother from Person to Man and Woman. Check the derived property of these references in the Properties View.

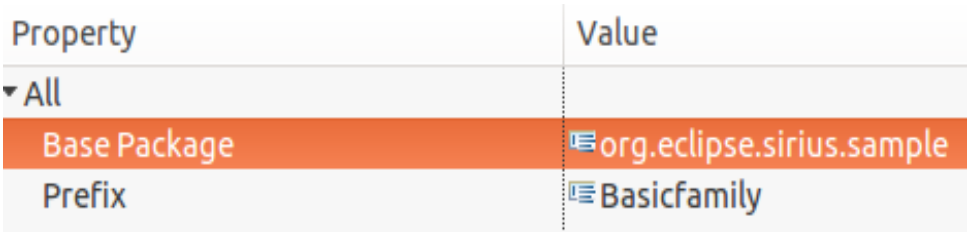


2.2.4 Generate the source code of the Metamodel

1. validate Ecore model :
   (a) Open the Ecore model by double clicking it.
   (b) Click on the small triangle in the tree-based editor to show the EPackage, which is represented by a small purple figure.
   (c) Right-click the small figure and select validate.



2. right click on Ecore model» new » others ...
3. Select EMF Generator Model under Eclipse Modeling Framework
4. Click next » Ecore model » Load » Finish
5. Open the Generator File *basicfamily.genmodel* by double clicking it.
6. In the Properties View, fill the property Base Package with *org.eclipse.sirius.sample.*



This value forces the generation of the source code into a package named *org.eclipse.sirius.sa*

7. Right-click on the package basicfamily and select Generate All.(this genmodel file contains a model that allows EMF to generate the Java code corresponding to the metamodel).

The "Generate All" produces these elements :

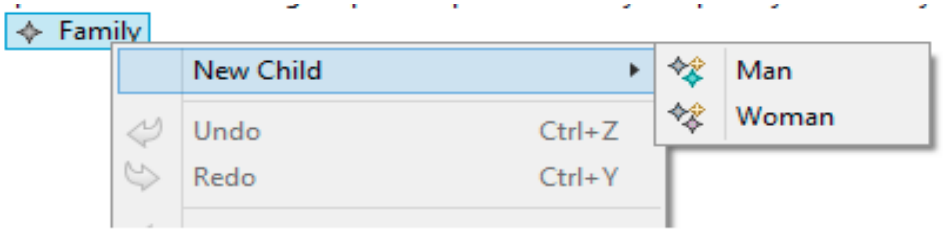
- The folder src in *org.eclipse.sirius.sample.basicfamily*
- The file *MANIFEST.MF* in the META-INF directory
- The project *org.eclipse.sirius.sample.basicfamily.edit org.eclipse.sirius.sample.basicfamily.editor org.eclipse.sirius.sample.basicfamily.tests*

2.2.5 Test the metamodel with Epsilon

1. Right click on Ecore model a,d select Register EPackage.
2. Right click on project folder » New » Other » Epsilon » EMF Model.
3. choose Family as Root instance type and browse to select EPackage , type *basicfamily and select the appropriate uri

EMF has created a new model and automatically opened the default tree editor.

To add elements to this model, right click on the Family element and select Man or Woman menu.



With the properties view you can edit the person's name and set itsParents or Children relations to other persons in the model.

Chapitre 3

Xtext

Xtext is a framework for building DLSs (domain specific languages). In fact, it can be seen as a DSL for defining DSLs.

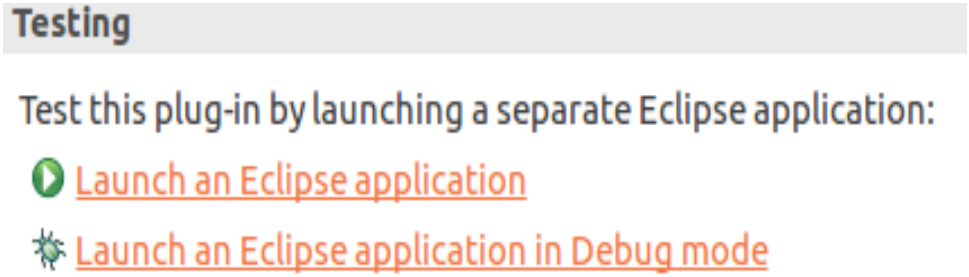
3.1 Create Xtext project

1. create a new Xtext project by selecting File –> New… Project… –> Xtext Project From Existing Ecore Models.
2. Choose EMF generator model of basicfamily and Enter the rule (Family) » Next » Choose the extension (family in our case) and change the project name to *org.xtext.example.basicfamily* and language name to *org.xtext.example.Basicfamily*.
3. Click Finish to let Xtext create the three projects that make up your DSL :
   (a) org.xtext.example.entity – this project contains the DSL itself, including all back end infrastructure like the parser and the meta model.
   (b) org.xtext.example.entity.ui – as the name implies, this project contains all UI centric code of the DSL : the editor, the outline, content assist and so forth
   (c) org.xtext.example.entity.generator – this project contains a code generator which will transform the DSL scripts (aka models) you write in your DSL into something useful. In our example, we will create some POJOs and DAOs from our models.

Upon finishing creating these three projects, Xtext opens a file basicfamily.xtext, which contains a sample grammar, which we can change if we want to .

3.2 Compiling the DSL

1. select org.xtext.example.basicfamily » src » org.xtext.example.basicfamily
2. Right-click on BasicFamily.xtext » Run As » Generate Xtext Artifacts
3. select org.xtext.example.basicfamily.ui » plugin.xml » Testing » Launch an Eclipse application in Debug mode.



Seperate Eclipse application will be launched

4. File » New » Other » General » Project ... » Finish
5. Right click on the project » New » File » name the file test.family Try some tests ...

Source : http ://www.eclipse.org/emf

https ://wiki.eclipse.org/Ecore_Tools

https ://www.eclipse.org/ecoretools/

http ://www.eclipse.org/modeling/emf/docs/

http ://help.eclipse.org/mars/index.jsp

https ://wiki.eclipse.org/Sirius/Tutorials

http ://www.win.tue.nl/ mvdbrand/courses/GLT/1415/exercises/metamodel-tutorial.pdf

http ://www.vogella.com/tutorials/EclipseEMF/article.html