INF5120
”Modellbasert Systemutvikling”
”Modelbased System development”

Lecture 13: 24.04.2017

Arne-Jørgen Berre

arneb@ifi.uio.no or **Arne.J.Berre@sintef.no**

Content

- Model Transformation
- Model to Model
- Model to Text

- NFR – Non Functional Requirements ?

Course parts (16 lectures) - 2017

- January (1-3) (Introduction to Modeling, Business Architecture and the Smart Building project):
- 1-16/1: Introduction to INF5120
- 2-23/1: Modeling structure and behaviour (UML and UML 2.0 and metamodeling) - (establish Oblig groups)
- 3-30/1: WebRatio for Web Apps/Portals and Mobile Apps – and Entity/Class modeling – (Getting started with WebRatio)

- February (4-7) (Modeling of User Interfaces, Flows and Data model diagrams, Apps/Web Portals - IFML/Client-Side):
- 4-6/2: Business Model Canvas, Value Proposition, Lean Canvas and Essence
- 5-13/2: IFML – Interaction Flow Modeling Language, WebRatio advanced – for Web and Apps
- 6-20/2: BPMN process, UML Activ.Diagrams, Workflow and Orchestration modelling value networks
- 7-27/2: Modeling principles – Quality in Models
- 27/2: Oblig 1: Smart Building – Business Architecture and App/Portal with IFML WebRatio UI for Smart Building

- March (8-11) (Modeling of IoT/CPS/Cloud, Services and Big Data – UML SM/SD/Collab, ThingML Server-Side):
- 8-6/3: Basis for DSL and ThingML -> UML State Machines and Sequence Diagrams
- 9-13/3: ThingML DSL - UML Composite structures, State Machines and Sequence Diagrams II
- 10-20/3: Guest lecture, "Experience with Modelling", Anton Landmark, SINTEF
- 11-27/3: ThingML part 2 and UML Service Modeling, Architectural models, SoaML. Role modeling and UML Collaboration diagrams

- April/May (12-14) (MDE – Creating Your own Domain Specific Language):
- 12-3/4: Model driven engineering – Metamodels, DSL, UML Profiles, EMF, Sirius Editors – intro to Oblig 3

- EASTER – 10/4 og 17/4
- 20/4: Oblig 2: Smart Building – Individual and group delivery - Internet of Things control with ThingML – Raspberry Pi, Wireless sensors (temperature, humidity), actuators (power control)

- 13-24/4: MDE transformations, Non Functional requirements – Discussion of Oblig2 and 3
- 1. Mai – Official holiday
- 4/5: Oblig 3 - Your own Domain Specific Language – (ArchiMate) (Delivery – Thursday May 4th)
- 14-8/5: SmartBuilding – Integrating App with Server side and Architmate editor (Discussion of Oblig 3)

- May (15-17): (Bringing it together)
- 15-15/5: Summary of the course – Final demonstrations
- 16-22/5: Previous exams – group collaborations (No lecture)
- 17-29/5: Conclusions, Preparations for the Exam by old exams
- June (Exam)
- 13/6: Exam (4 hours), June 13th, 0900-1300

Content

- MOF and EMF
- Model transformations
- MOFScript
- ATL
- Acceleo

- OCL – UML Object Constraint Language

MDA-compliant Eclipse technologies

- Eclipse Modeling Tools: http://www.eclipse.org/downloads/
- Eclipse Modeling Framework (EMF)
  - http://www.eclipse.org/emf/
  - EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model.
- Eclipse Graphical Editing Framework (GEF)
  - http://www.eclipse.org/gef/
  - The Graphical Editing Framework (GEF) allows developers to take an existing application model and quickly create a rich graphical editor.
- Eclipse Graphical Modeling Framework (GMF)
  - http://www.eclipse.org/gmf/
  - The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF.
- Eugenia - plus Spray/Graphiti
  - EuGENia is a tool that automatically generates the .gmfgraph, .gmftool and .gmfmap models needed to implement a GMF editor from a single annotated Ecore metamodel
  - http://www.eclipse.org/epsilon/doc/eugenia/
- Atlas Transformation Language
  - http://www.eclipse.org/gmt/atl/
  - The ATL project aims at providing a set of transformation tools for GMT. These include some sample ATL transformations, an ATL transformation engine, and an IDE for ATL (ADT: ATL Development Tools).
- Eclipse Process Framework (EPF)
  - http://www.eclipse.org/epf/
  - To provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.

EMFText - MOF to text

- http://www.emftext.org/index.php/EMFText

Introduction

ATHENA Model-Driven Interoperability (MDI) Framework

- **MDA & Interoperability**
- **Metamodelling**
- **UML Profiles & DSLs**
- **Model Transformations**
- **Method Engineering**

Reusable MDI Assets

- Method chunks
- Tools and services
- Models and metamodels
- Model transformations
- DSLs and UML profiles
- Reference examples

Model to Text transformation 1/2

- MDA places modelling at the heart of the software development process.
- Various models are used to capture various aspects of the system in a platform independent manner.
- Sets of transformations are then applied to these platform independent models (PIM) to derive platform specific models (PSM).
- These PSMs need to be eventually transformed into software artefacts such as code, deployment specifications, reports, documents, etc.
- It is also common to generate code directly from PIM-like models. (DSL approach)

Model to Text transformation 1/2

- QVT, ATL and MOFScript M2M addresses the need of model – to – model transformation (e.g., PIM – to – PIM, PIM – to – PSM and PSM – to – PSM)
- The MOF Model to Text (mof2text) standard addresses how to **translate a model to various text artefacts** such as code, deployment specifications, reports, documents, etc.
- Essentially, the mof2text standard needs to address how to transform a model into a linearized text representation.
- An intuitive way to address this requirement is **a template based approach** wherein the text to be generated from models is specified as a set of text templates that are parameterized with model elements.

Motivation

- Why do we need model-to-text transformation?
  - **Raise the level of abstraction**
    - Systems are getting more complex
    - Raise of abstraction has proven useful (for instance: Assembly to COBOL)
  - **Automation of the software development process**
    - Decrease development time
    - Increase software quality
    - Focus on the creative part
  - **Automatic generation of new artefacts from your models**
    - Java, EJB, JSP, C#
    - SQL Scripts
    - HTML
    - Test cases
    - Model documentation

Alternatives

- What are the alternatives?
  - **Programming languages** (e.g Java),
  - **Template/scripting languages** (e.g XSLT, OaW, Eclipse Java Emitter Templates – JET, OMG MOF Model 2 Text)
  - **Model Transformation Languages** (e.g. ATLAS Transformation Language (ATL)), proprietary **UML-based** script languages, **DSL**-based approaches, Other MOF-based text/code generators

- Properties of the alternatives:
  - Neither programming languages nor scripting languages tend to take advantage of source metamodels.
    - However, it can be done programmatically in Java (e.g. using Eclipse Modelling Framework (EMF))
  - Model 2 Model Transformation languages such as ATL is metamodel-based, but is not designed with text generation in mind. However, it can be done also in ATL
  - UML tool script languages are tied to both UML and a vendor, and are not based on standards.
  - DSLs provides the flexibility of metamodel-based tools; they typically hard code code generation for each domain-specific language.
    - The difference between a MOF-based approach and a DSL is not significant, as transformations in MOF-based approaches also will depend on a particular metamodel.
  - Other MOF-based text generators have not been available, but will emerge.

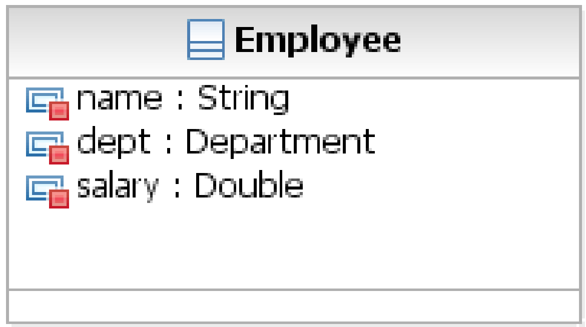
OMG Request for Proposal for a model-to-text transformation language

- **OMG RFP Issued in 2004**
- **Mandatory Requirements:**
  - Generation from MOF 2.0 models to text
  - Reuse (if applicable) existing OMG specifications, in particular QVT
  - Transformations should be defined at the metalevel of the source model
  - Support for string conversion of model data
  - String manipulation
  - Combination of model data with hard coded output text
  - Support for complex transformations
  - Multiple MOF models as input (multiple source models)
- **Optional Requirements**
  - round-trip engineering
  - detection/protection of hand-made changes for re-generation
  - *traceability* is a (possible) means of supporting the last two.

MOF to Text Overview

- A **template-based** approach wherein a *Template* specifies a text template with placeholders for data to be extracted from models.
- These placeholders are essentially expressions specified over metamodel entities with queries being the primary mechanisms for selecting and extracting the values from models.
- These values are then converted into text fragments using an expression language augmented with a string manipulation library.
- *Template* can be composed to address complex transformation requirements. Large transformations can be structured into *modules* having public and private parts.

MOF to Text Example




```
[template public classToJava(c : Class)]
    class [c.name/] {
        // Attribute declarations
        [attributeToJava(c.attribute)]
        // Constructor
        [c.name/] () {
        }
    }
[/template]

[template public attributeToJava(a : Attribute)]
[a.type.name/] [a.name/];
[/template]
```

```
class Employee
{
        // Attribute declarations
        String name;
        Department dept;
        Double salary;

        // Constructor
        Employee()
        {

        }
}
```

Language details 1/3

- Instead of defining two templates separately, a template can iterate over a collection by using the *for* block.
- Using the *for* block preserves WYSIWYG-ness and improves readability.
- For example the *classToJava* template can use the *for* block as shown below:

```
[template public classToJava(c : Class)]
        class [c.name/] {
                // Attribute declarations
                [for(a : Attribute | c.attribute)]
                [a.type.name/] [a.name/];
                [/for]
                // Constructor
                [c.name/] () {
                }
        }
[/template]
```

The for block declares a loop variable ‘a’ of type Attribute and produces for each Attribute in the collection c.attribute the text between the *[for]* and *[/for]*.

Language details 2/3

- A template can have a guard that decides whether the template can be invoked. For example, the following *classToJava* template is invoked only if the class is concrete.

[template public classToJava(c : Class) ? (c.isAbstract = false)]

- Complex model navigations can be specified using *queries*. The following example shows use of a query *allOperations* to collect operations of all abstract parent classes of a class in a class hierarchy.

[query public allOperations(c: Class) : Set (Operation) =c.operation->union
(c.superClass->select(sc|sc.isAbstract=true)->iterate(ac : Class;
os:Set(Operation) = Set{}| os->union(allOperations(ac)))) /]

Language details 3/3

- As we have seen, a template has WYSIWYG nature with the text to be output being specified in exactly the way it should look in the output.
- There may be cases where the quantity of the text producing logic far outweighs the text being produced. In this case, it is more intuitive to specify the text producing logic without use of special delimiters. **@code-explicit**

```
@code-explicit
template public classToJava(c : Class)
'class 'c.name ' {
        // Constructor
        'c.name' () {
        }
}'
/template
```

```
@text-explicit
[template public classToJava(c : Class)]
class [c.name/] {
        // Constructor
        [c.name/] () {
        }
}
[/template]
```

OMG Standard for model-to-text

- MOF2Text: A merge of the different model to text proposals, where MOFScript was one of several

- Many similarities with MOFScript:
  - imperative language w/ explicit rule calls
  - reusing selected parts of QVT/OCL

- Differs from MOFScript:
  - Mainly syntactical
  - Context type does not have its own slot, inserted in the parameter list
  - More traditional for-statements instead of forEach
  - Escaping direction is flexible: The transformation code can be escaped, or the output text can be escaped (as in MOFScript).

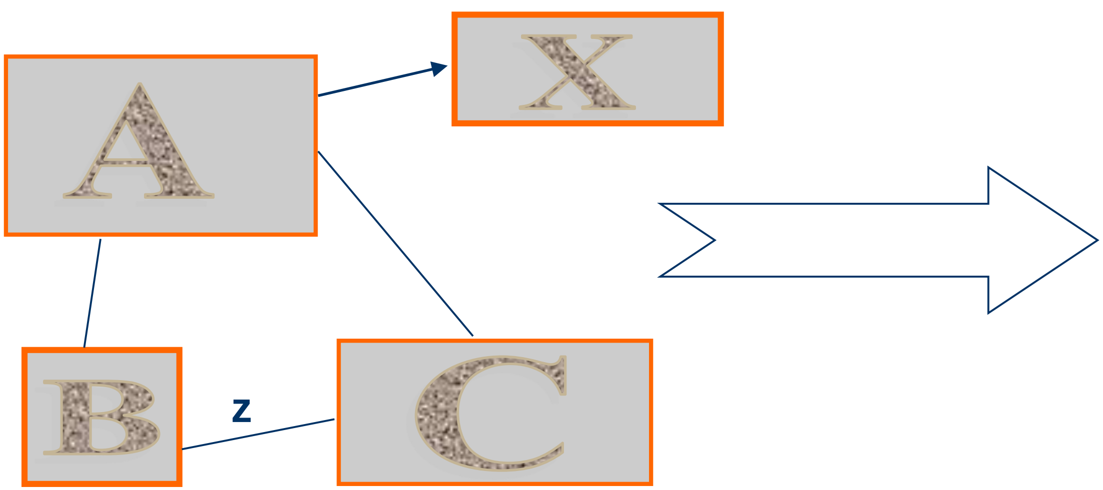
Tool Support

- Eclipse M2T Project:
  - Acceleo MTL:
    - http://www.eclipsecon.org/2009/sessions?id=387
- MOFScript M2T:
  - Parser, model and editor that uses the MOFScript runtime
    - http://modelbased.net/modelplex/mof2text/index.html

MOFScript Model to Text language and Tool

Introduction

- Model Driven Development (MDD) emphasizes the use of models as first class artifacts
- CIM <--> PIM <--> PSM <--> TEXT / CODE
- MOFScript bridges Model → Text



Public class a extends x

………………..

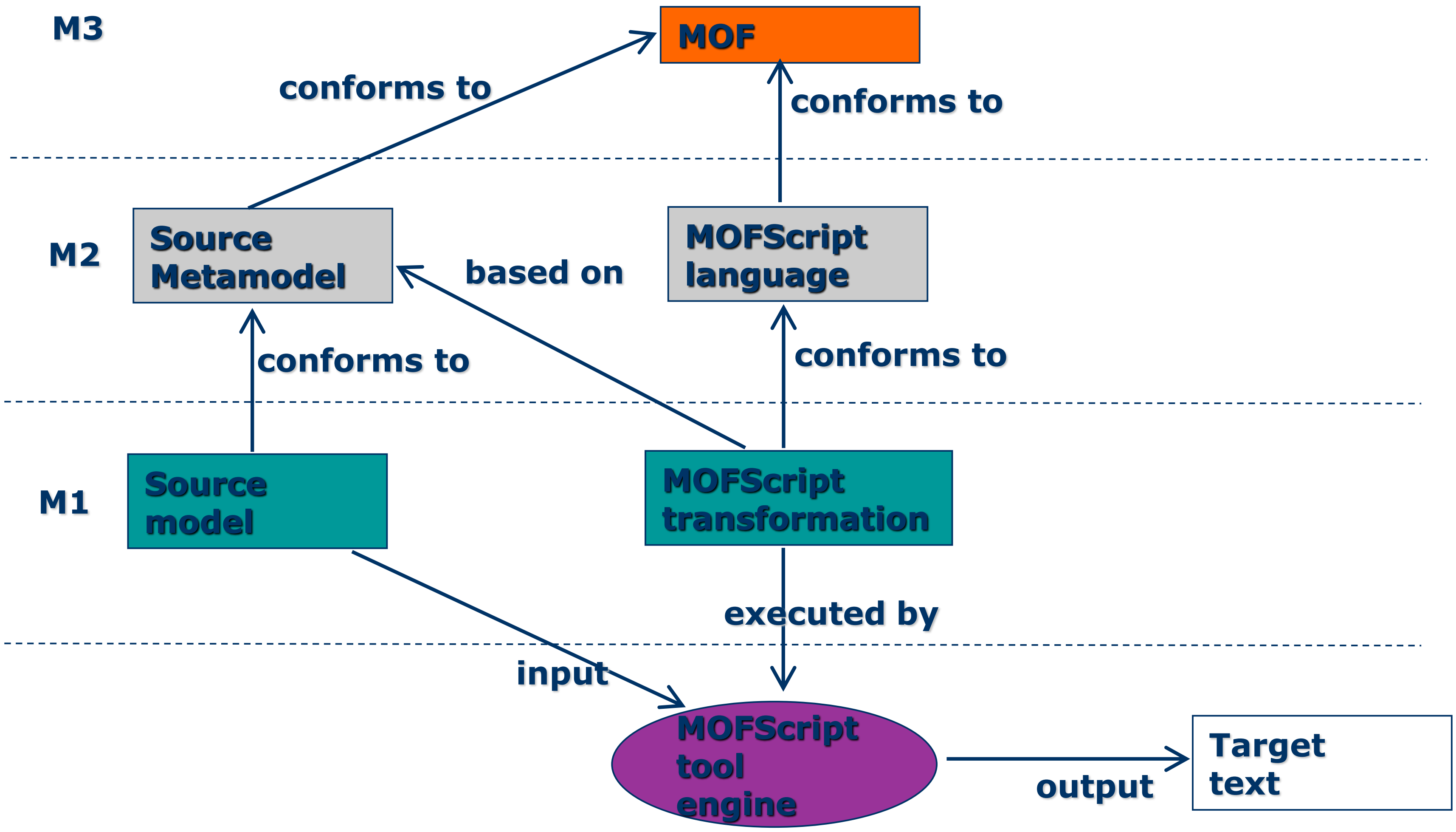
B.wsdl

……………….

Z association type Simple

c.Html

MOFScript placed in the 4-layer architecture



What is MOFScript?

- The MOFScript tool is an implementation of the MOFScript model to text transformation language
- Developed at Sintef ICT in the EU-supported MODELWARE project
- An Eclipse plug-in
- Mapping of model artifacts to a multitude of textual languages
- Was part of standardization process within OMG
  - OMG RFP MOF Model to Text Transformation process

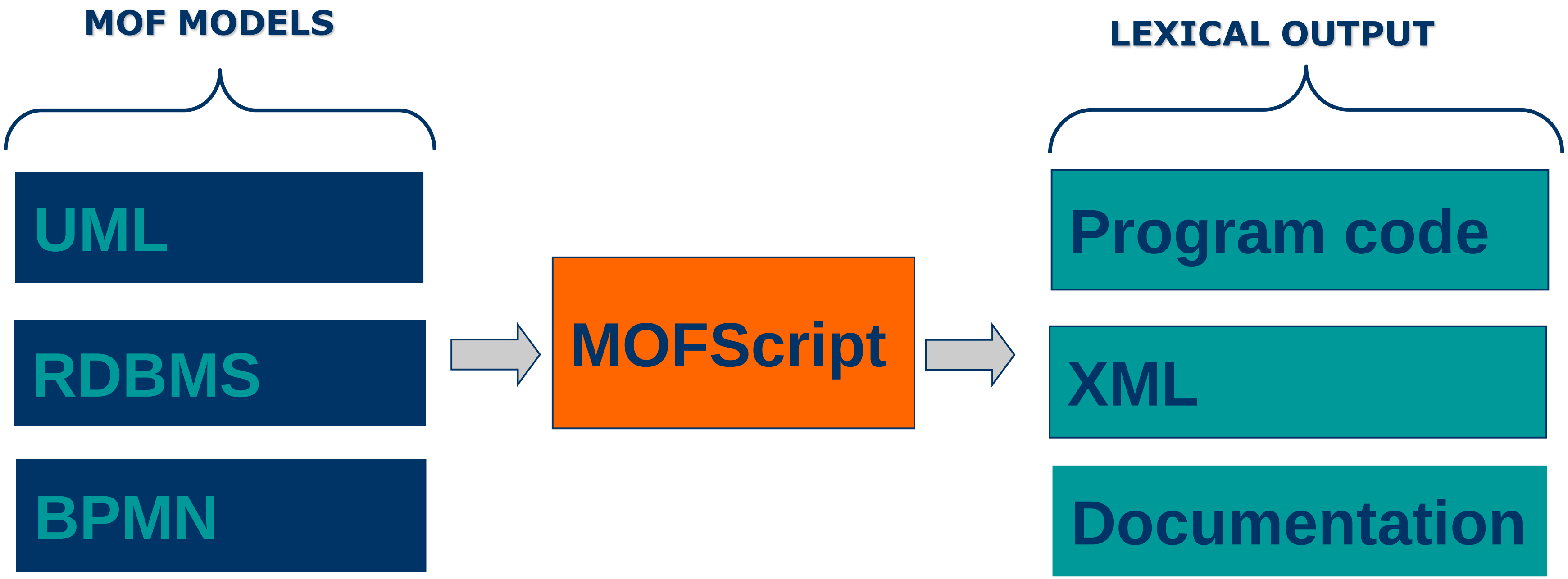
MOFScript a transformation language

- Language for writing model to text transformations
- Rules / Operations are called explicit (Procedural language)
- Partly based on the current QVT specification (keeps it within the family)
- Transforms input models to output text files
  - Generate text from any MOF-based model, e.g., UML models or any kind of domain model.

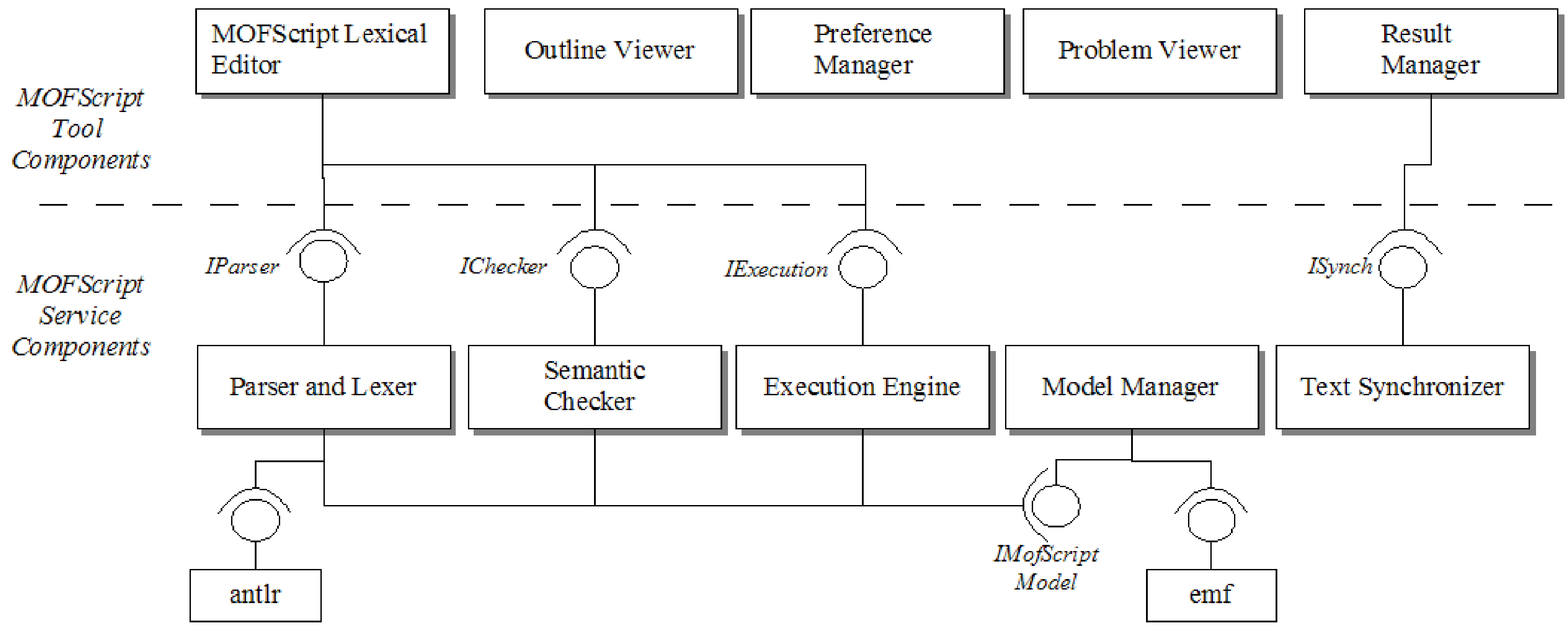
MOF Script - background

- Usability
  - Ease of use: Writing and understanding
  - Few constructs
- End user recognizability
  - Similar to programming and scripting languages
  - Imperatively oriented
- Sequential execution semantics
  - Rules are called explicitly
    - *Might also support pattern matching execution*
  - Contents of rules is executed sequentially
- Compatibility
  - Alignment with latest QVT (QVTMerge) specification

MOFScript in action

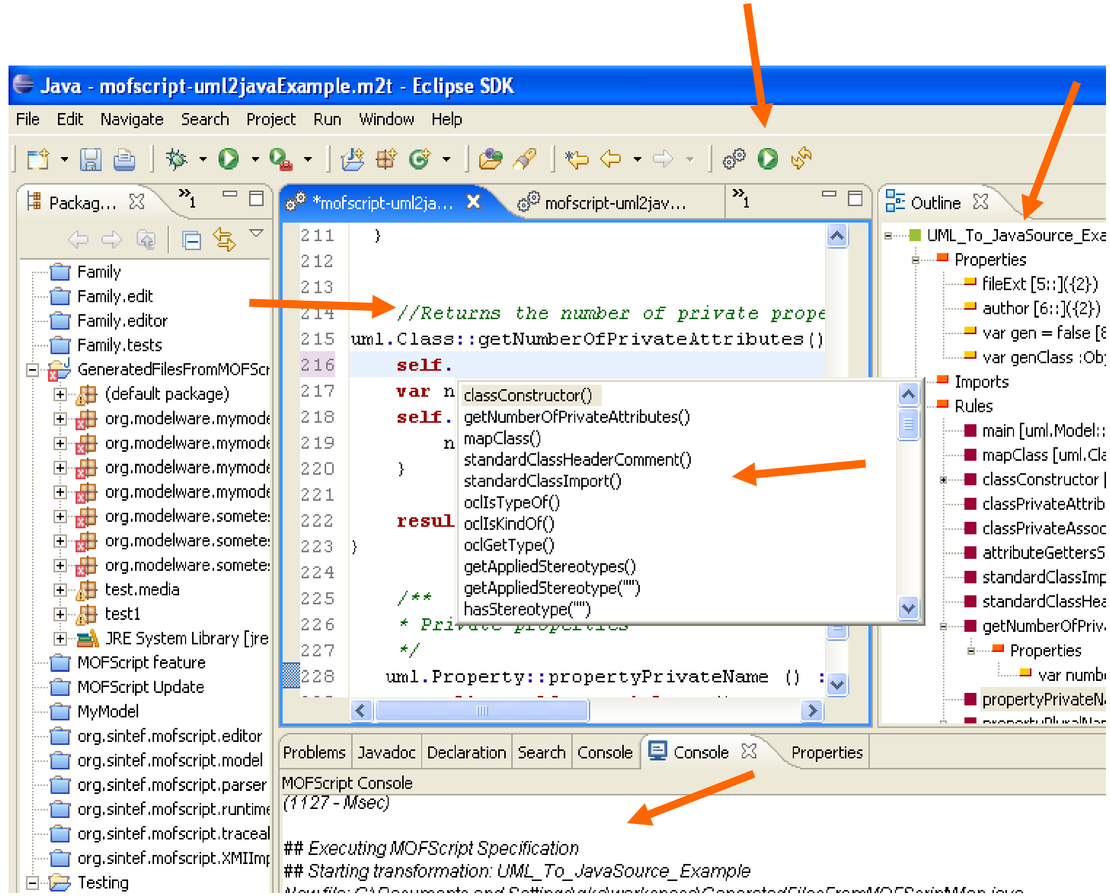


MOFScript architecture



MOFScript a model to text tool

- Provides the means of:
  - Editing, compiling and executing
- Syntax high-lightning
- Content assist
- Outline
- MOFScript Console



The main steps of using the MOFScript tool

- Task: Define a transformation from source model A to text t. (A→t)
  1. Import or define the source metamodel for A.
  2. Write the MOFScript to transform A to t in the MOFScript editor
  3. Compile the transformation. Any errors in the transformation will be presented.
     1. Fix errors, if any
  4. Load a source model corresponding to A's metamodel.
     1. Using the Eclipse plugin, this is prompted by the tool when trying to execute.
  5. Execute the MOFScript in the MOFScript tool.
     1. The transformation is executed. Output text / files are produced.

Built-in operations

- String operations
  - Various string manipulation operations, such as:
    - size, substring, subStringBefore|After, toLower, toUpper, indexOf, trim, normalizeSpace, endsWith, startsWith, replace, equals, equalsIgnoreCase, charAt, isLowerCase, isUpperCase
- Collection library
  - Standard collection operations…
    - *Hashmap: put, get, clear, size, keys, values, isEmpty, forEach,*
    - *List: add, size, clear, isEmpty, first, last, forEach*
    - *Model: size, first, isEmpty, forEach*
- System and utility operations
  - Various utility functions, such as
    - *time, date, getenv, setenv, position, count*
- *UML2 Operations*
  - *Operations available when UML2 models are loaded*
    - *hasStereoType, getAppliedStereotypes, getAppliedStereotype*

Textual syntax

- Textmodule
- Rules
- Files
- Escaped output

```
textmodule UML2Java (in myMod:uml2)
```

```
myMod.Package::mapPackage ()  {
  'package ' self.name ';'
}
```

```
file f2 (c.name + ".java")
' package ' c.ownerPackage.name';'
f2.println ("public class" + c.name);
```

```
   'public class ' c.name
       ' extends Serializable {'
```

Textual syntax

- Entry point rule

```
myMod.Model::main () {
   // code for entry point
}
```

- Iterators

```
self.ownedMember->forEach(c:myMod.Class)
  '<class name="' c.name ' "/>'
}
```

- Conditional statements

```
if (self.hasStereotype("Feature") {
  ' This is a feature type '
} else if (self.hasStereotype("Product")) {
  ' This is a product type '
} else {
  ' this is neither '
}
```

Textual syntax

- Collections

```
var packageNames_List:List
var packageName_Hashtable:Hashtable

self.ownedMember->forEach(p:uml.Package) {
   packageNames_List.add (p.name)
   packageName_Hashtable.put (p.id, p.name)
}


if (packageName_Hashtable.size () > 0) {
  ' Listing the package names that does not start with 'S' '
  packageName_Hashtable->forEach (s:String | not(s.startsWith("S")) {
     ' Package: ' s
  }
}
```

Textual syntax

- Invoking rules

```
uml.Package::interfacePackages () {
  if (self.getStereotype() = “Service”){
    file (self.name.toLower() + ".wsdl")
    self.wsdlHeader()
    self.wsdlTypes()
    self.ownedMember->forEach(i:uml.Interface)
{
      i.wsdlMessages()
      i.wsdlPortType()
      i.wsdlBindings()
      i.wsdlService()
    }
    self.wsdlFooter()
  }
}
```

- Return results

```
uml.Package::getPackageNameToLower(): String {
result = self.name.toLower()
}
```

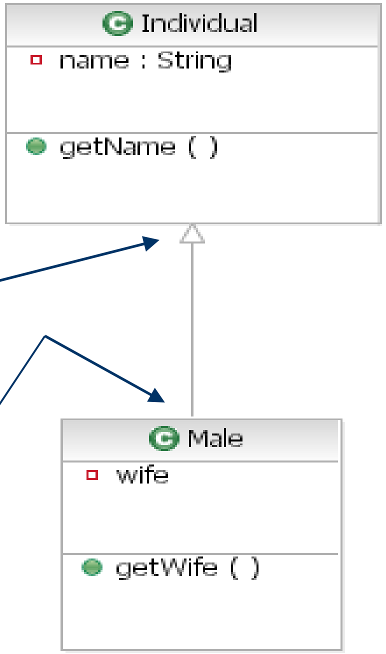
Uml2Java Example

```
//Context class
self.ownedAttribute->forEach(p : uml.Property | p.association = null) {
    p.attributeGetterSetters()
}
// Generate Getter and Setters
  uml.Property::attributeGettersSetters () {
      'public ' self.type.name ' get' self.name.firstToUpper() ' () {'
      'return ' self.name ';\n }\n'
      'public void set' self.name.firstToUpper() '(' self.type.name ' input ) { '
      self .name ' = input; \n } '
  }
```


```
public String getBookTitle(){
   return bookTitle;
}
public void setBookTitle(String input){
   bookTitle = input;
}
public Integer getNumberOfPages(){
   return numberOfPages
}
public void setNumberOfPages(Integer
input){
   numberOfPages = input;
}
```

FamilyModel example

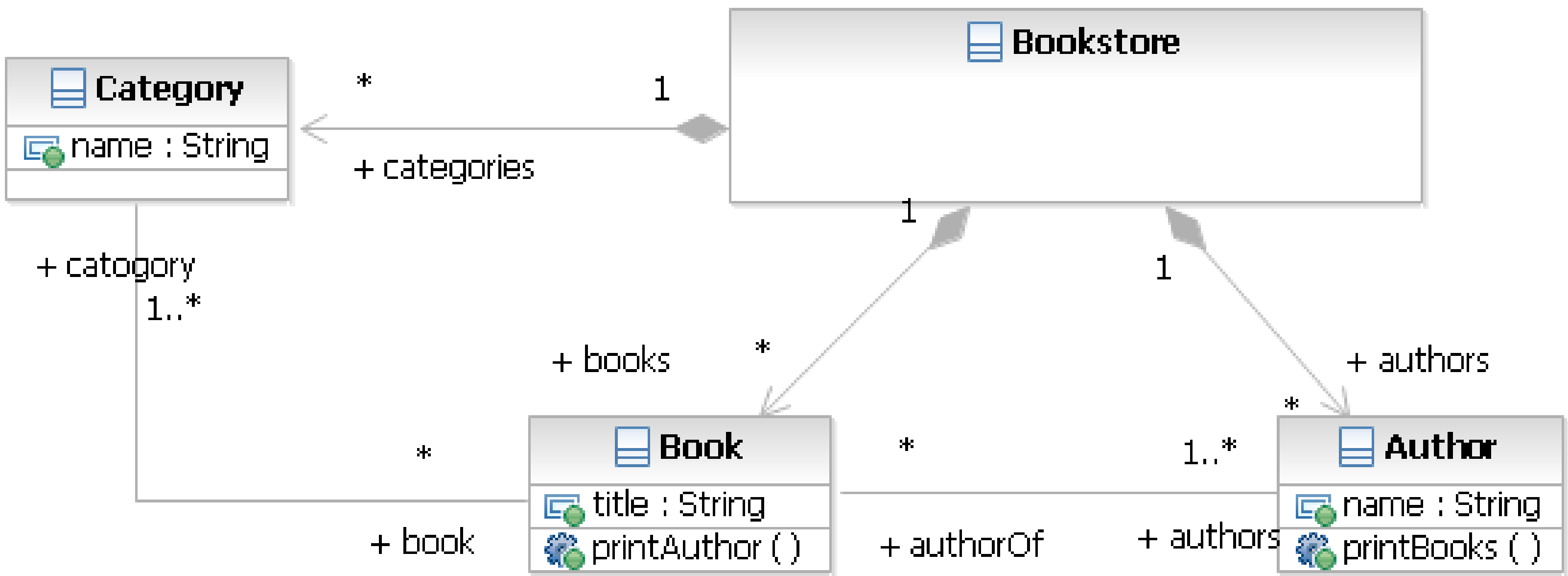
```
uml.Class::outputGeneralization(){
  self.generalization->forEach(g: uml.Generalization){
     if(not g.target.isEmpty()){
        g.target->forEach(c: uml.Class){
           stdout.println("Generalization target name: "+ c.name )
        } //g.target forEach
    }//if target
    if(not g.source.isEmpty()){
        g.source->forEach(c:uml.Class){
             stdout.println("Generalization source name: "+c.name)
        }//g.sourse forEach
    }//if source
}//self.generalization
}//outputGeneralization()
```



Generalization target name: Individual
Generalization source name: Male

MOFScript Example

Example Model



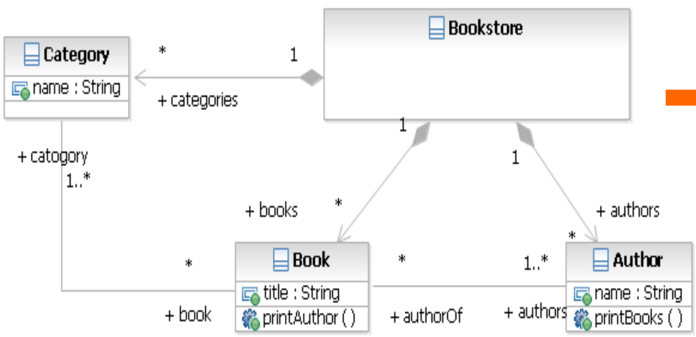
Example Transformation

```
uml.Class::main(){
    file(self.name+ ".java")
    'package ‘ packageName';\n
     import java.util.*;\n'
     self.visibility' class ' self.name'{
'
  self.ownedAttribute->forEach(p:uml.Property | p.association = null ){
    ' ' p.visibility' ' p.type.name' ' p.name';\n'

  }
self.ownedAttribute->forEach(p:uml.Property | p.association !=null ){
     '  // Association: authors:Author(1..-1)'
       '\t  ‘ p.visibility‘ HashMap<‘ p.type.name', ‘ p.name '>_'
          p.name.toLower()';'
  }
```

Example Generated Java Code

```
package org.sintef.no;
import java.util.HashMap;
public class Book {
   private String _title ;
// Association: authors:Author(1..-1)
    protected HashMap<String, Author>_authors;
// Association: category:Category(1..-1)
    protected HashMap<String, Category>_category;
}
```

Overview




```
uml.Class::main(){
   file(self.name+".java")
   'package 'packageName';\n
    import java.util.*;\n'
    self.visibility' class ' self.name'{
'
  self.ownedAttribute->forEach(p:uml.Property | p.association =
       null ){
   '   ' p.visibility' ' p.type.name' ' p.name';\n'

  }
self.ownedAttribute->forEach(p:uml.Property | p.association
       !=null ){
    '  // Association: authors:Author(1..-1)'
      '\t  'p.visibility'HashMap<'p.type.name', 'p.name '>_'
        p.name.toLower()';'
  }
```

```
package org.sintef.no;
import java.util.HashMap;
public class Book {
   private String _title ;
// Association: authors:Author(1..-1)
    protected HashMap<String, Author>_authors;
// Association: category:Category(1..-1)
    protected HashMap<String, Category>_category;
}
```

Model transformation service

- http://www.modelbased.net/tools/model-transformation-service

References

- OMG MOF Model to Text Transformation RFP
  - http://www.omg.org/cgi-bin/apps/doc?ad/04-04-07.pdf
- MOFScript submission
  - http://www.omg.org/cgi-bin/apps/doc?ad/05-05-04.pdf
- MOFScript tool
  - http://www.modelbased.net/mofscript
  - http://www.eclipse.org/gmt/mofscript
- MOFScript lecture:
  - http://www.modelware-ist.org/index.php?option=com_remository&Itemid=79&func=fileinfo&id=94
- OMG MOF to Text
  - http://www.omg.org/docs/ptc/06-11-01.pdf
  - http://modelbased.net/modelplex/mof2text/index.html

"Families to Persons"

A simple illustration of model-to-model transformation

Freddy Allilaire
Frédéric Jouault

ATLAS group, INRIA & University of Nantes, France

Adapted from

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 Marc
Mrs. Brenda Marc
… 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
<?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

```
<?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 text.

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

Each model conforms to a metamodel

Source metamodel

conformsTo

```xml
<?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"

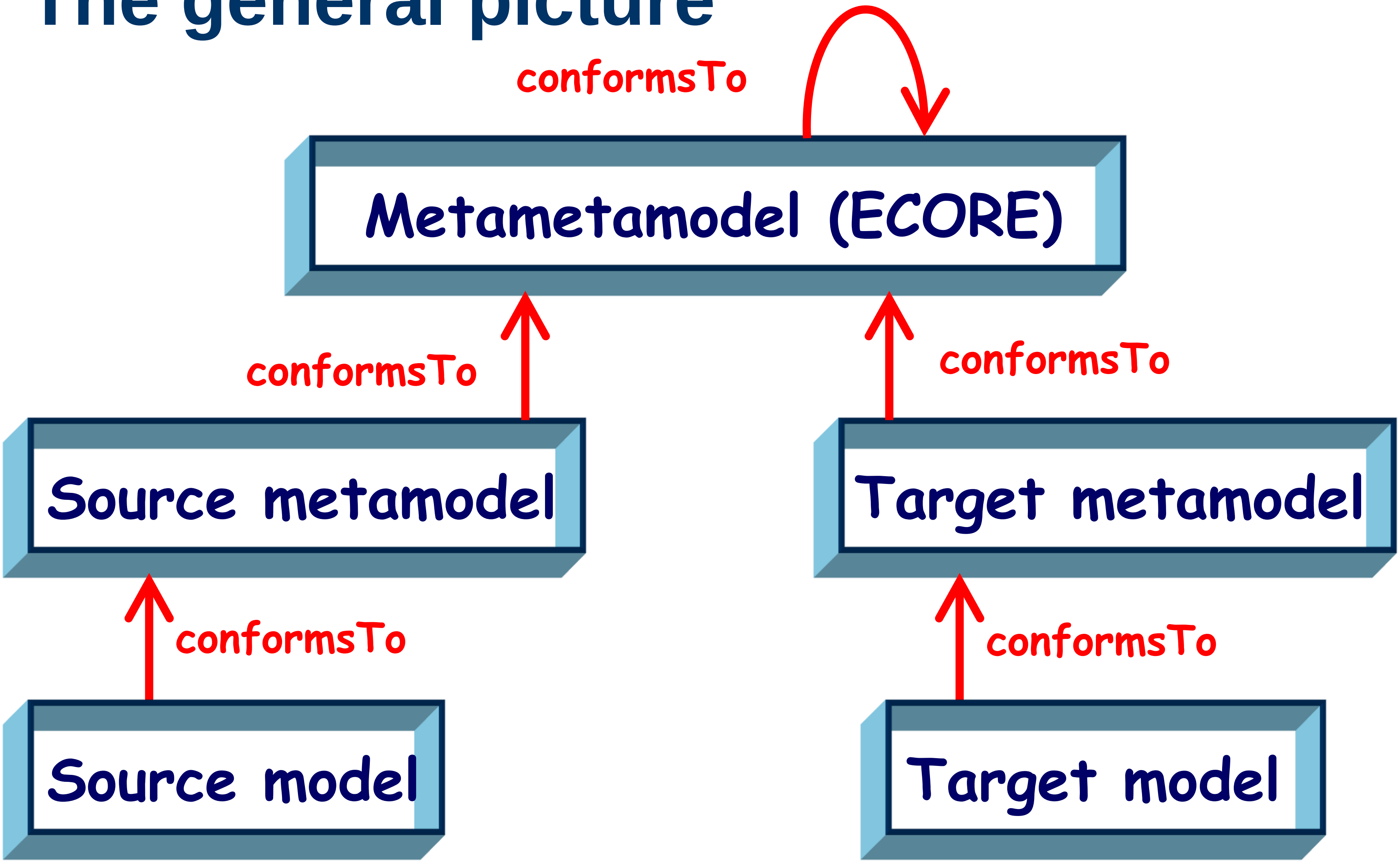
Target metamodel

conformsTo

```xml
<?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

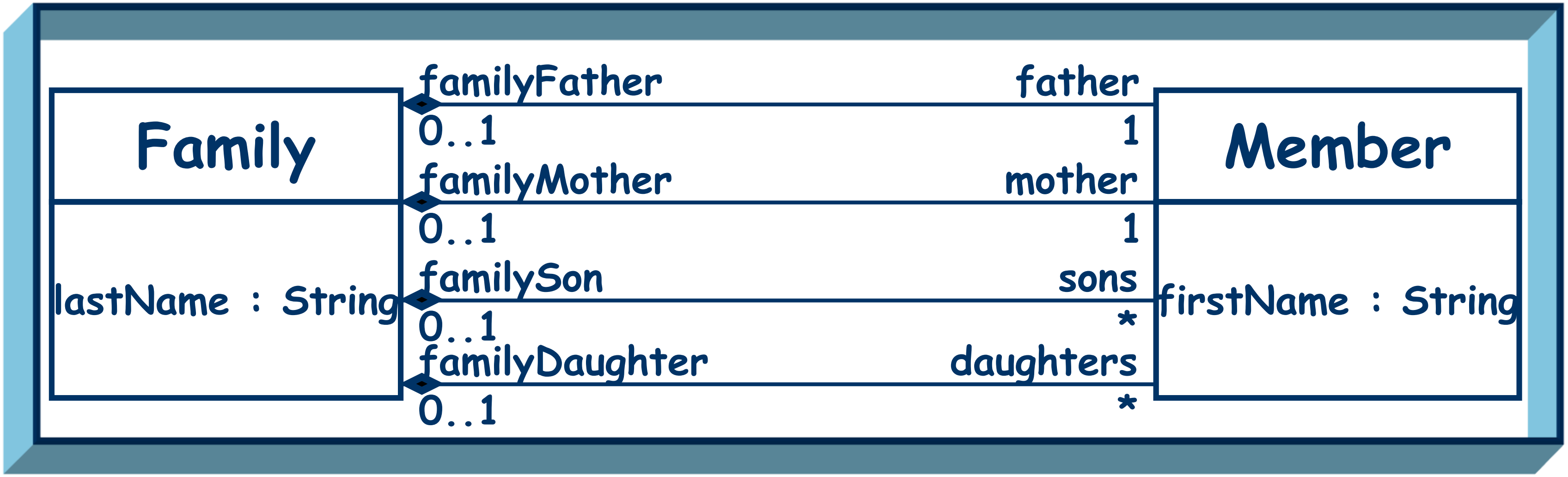
- In order to achieve the transformation, we need to provide:
  1. A source metamodel in KM3 ("Families").
  2. A source model (in XMI) conforming to "Families".
  3. A target metamodel in KM3 ("Persons").
  4. A transformation model in ATL ("Families2Persons").
- When the ATL transformation is executed,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

Definition of the source metamodel "Families"

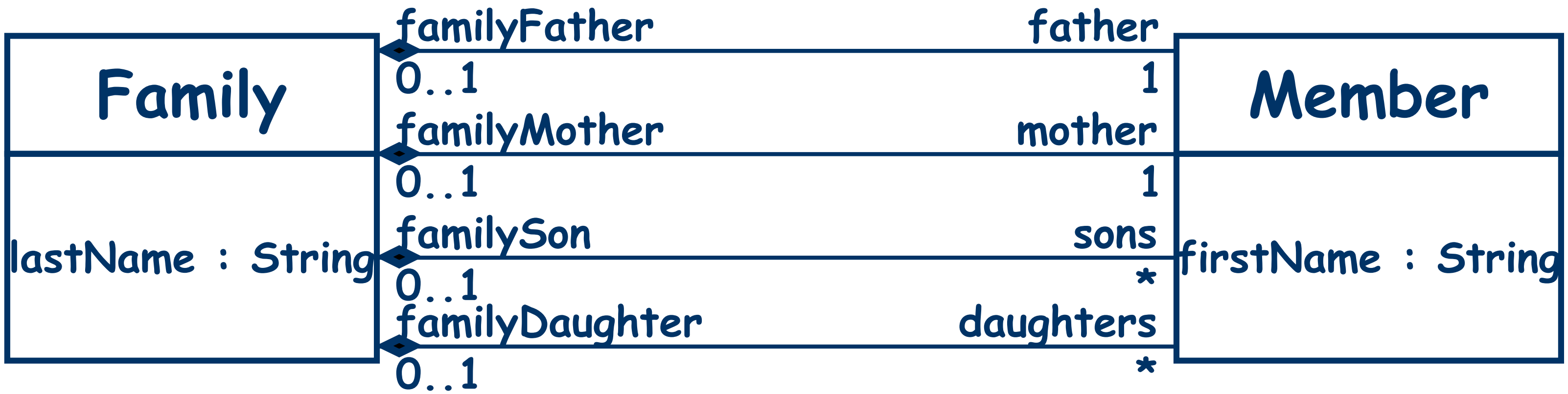
What is "Families":

- A collection of families.
- Each family has a <u>name</u> and is composed of <u>members</u>:
  - A <u>father</u>
  - A <u>mother</u>
  - Several <u>sons</u>
  - Several <u>daughters</u>
- Each family member has a <u>first name</u>.

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

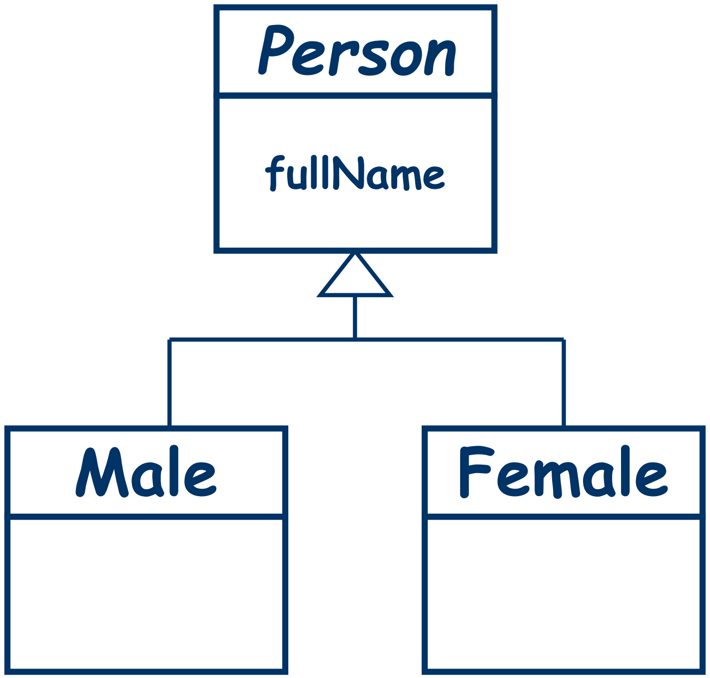


"Families" metamodel (visual presentation and KM3)



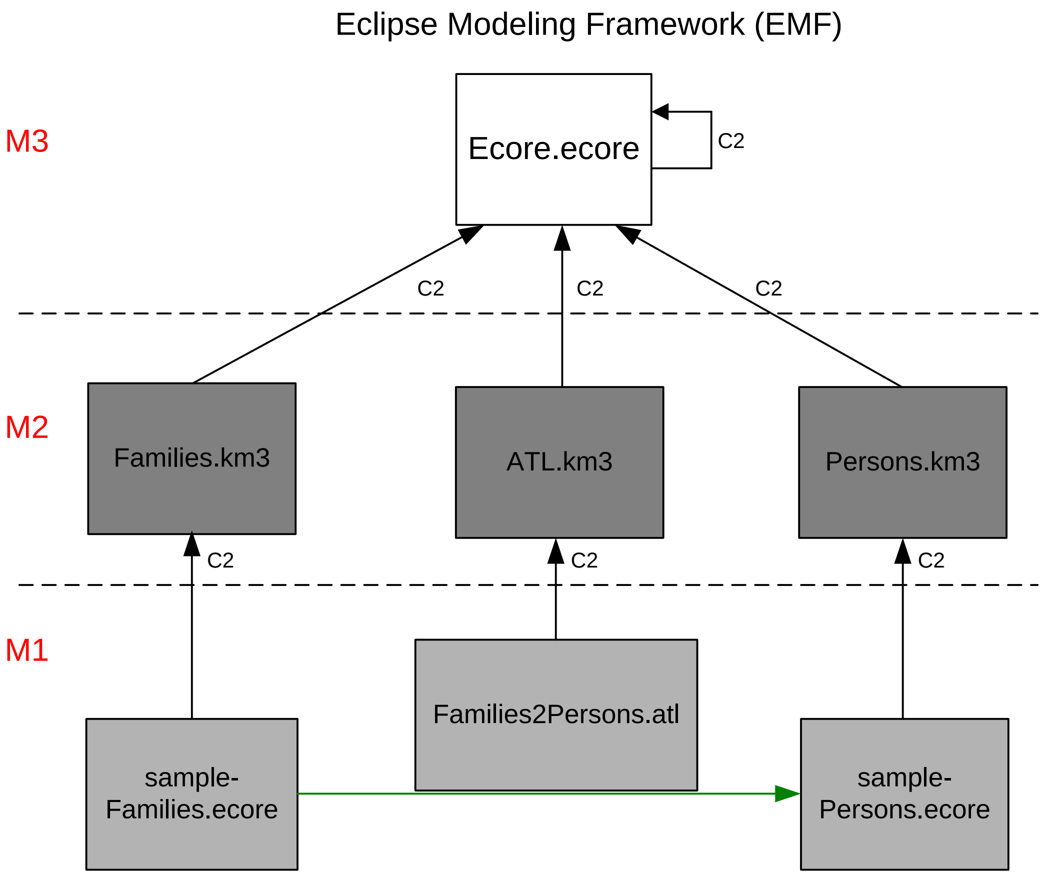

```
package Families {

	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



1. Our goal in this mini-tutorial is to write the ATL transformation, stored in the "Families2Persons" file.

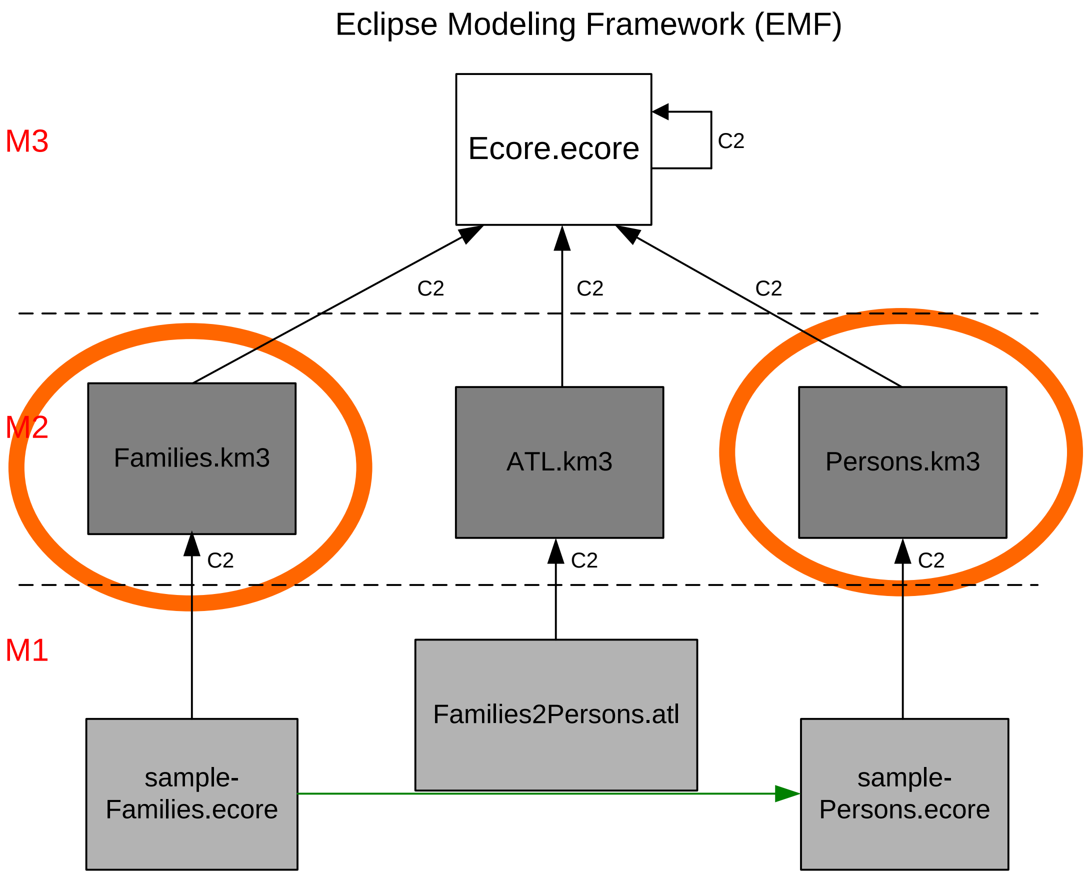
2. Prior to the execution of this transformation the resulting file "sample-Persons.ecore" does not exist. It is created by the transformation.

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).

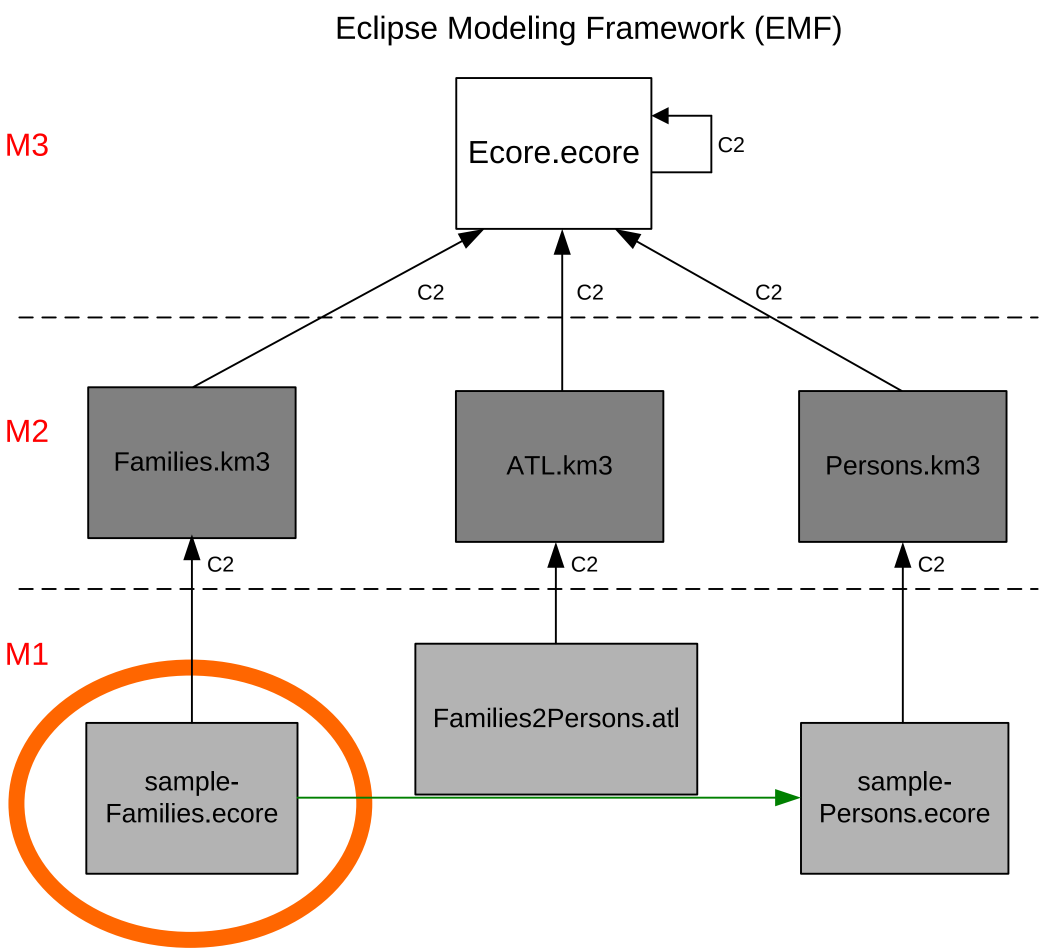
5. Similarly we take for granted that the environment provides the recursive definition of the metametamodel (supposedly in the "Ecore.ecore" file).

Families to Persons Architecture



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

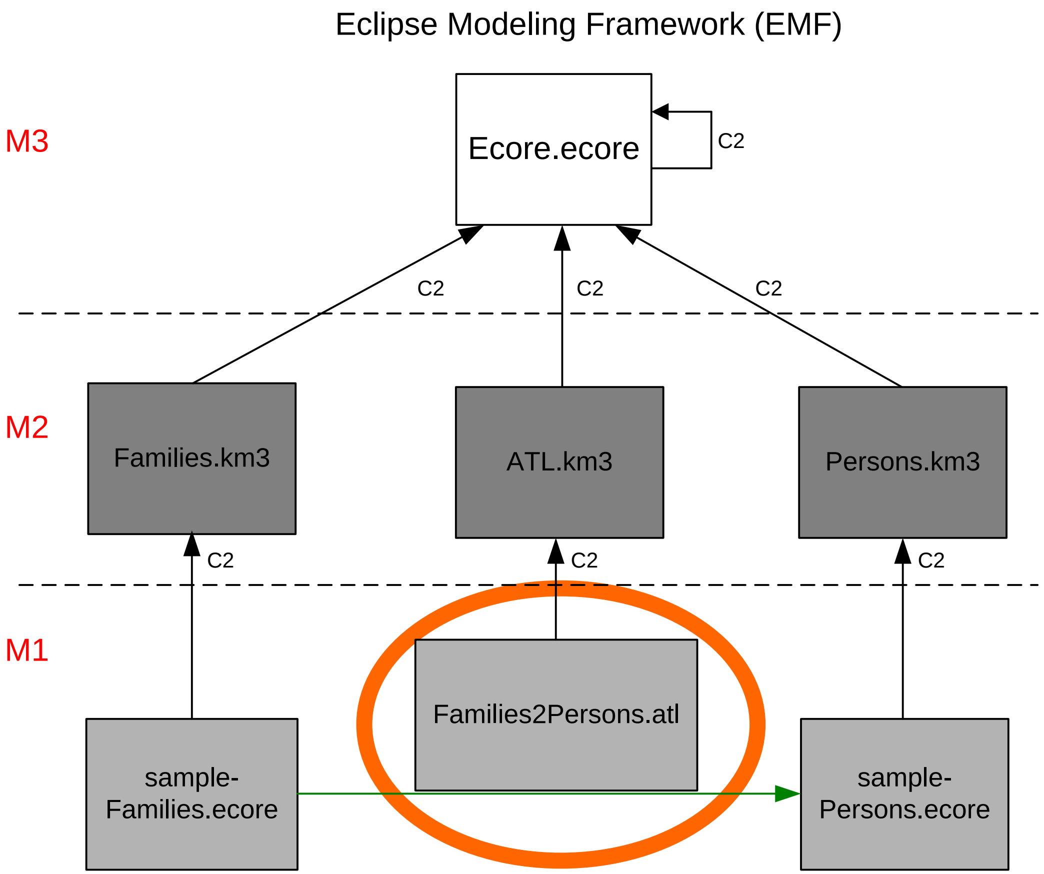
Families to Persons Architecture



1. 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



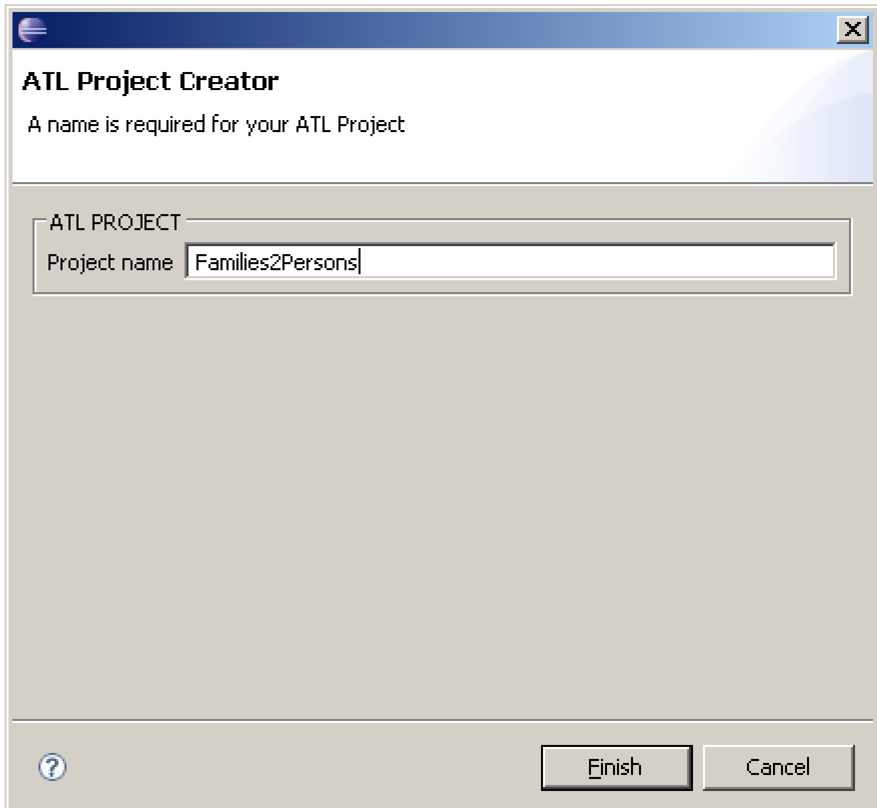
1. Now, let us start the creation of the ATL transformation *Families2Persons.atl*.

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.

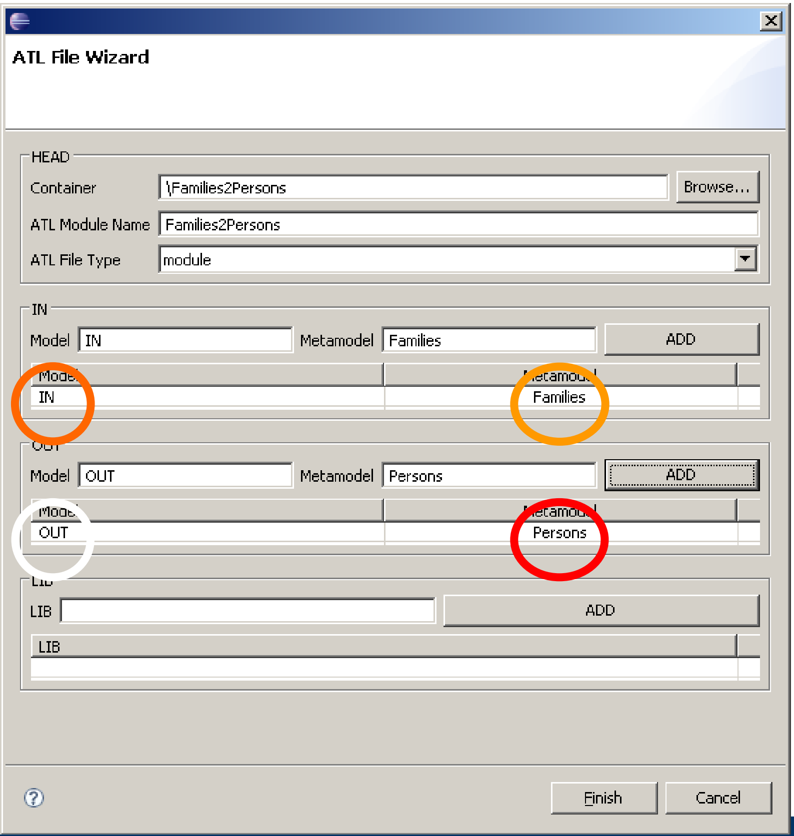


Families to Persons: ATL transformation creation

- 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



Families: Name of the source metamodel in the transformation

Persons: Name of the target metamodel in the transformation

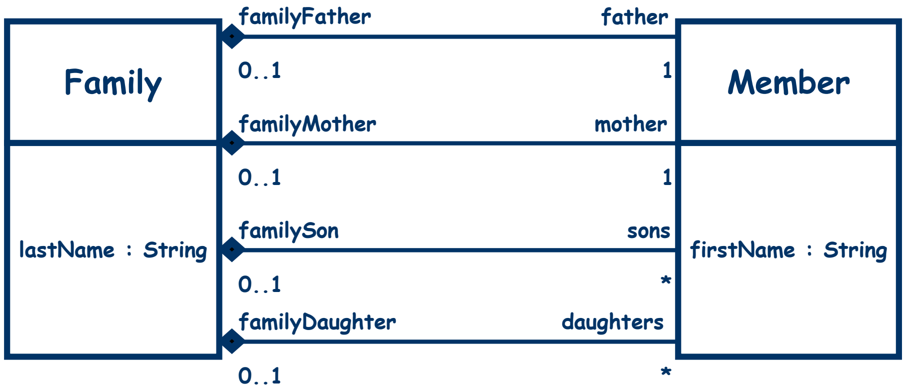
Families to Persons: header section

- 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()"

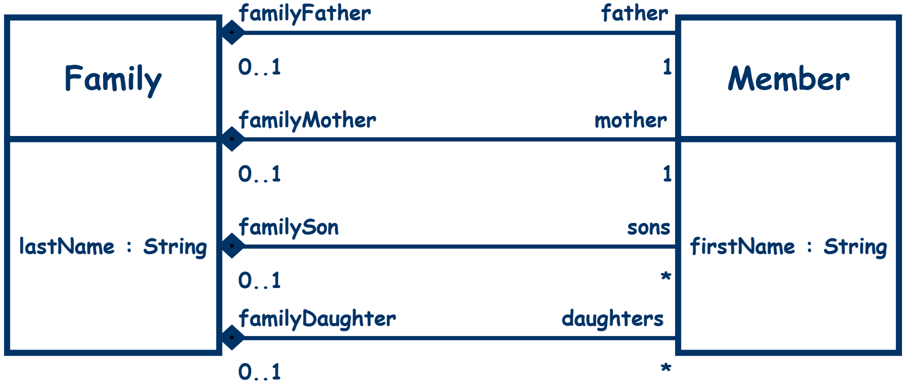
- A helper is an auxiliary function that computes a result needed in a rule.
- The following helper "isFemale()" computes the gender of the current member:




```
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

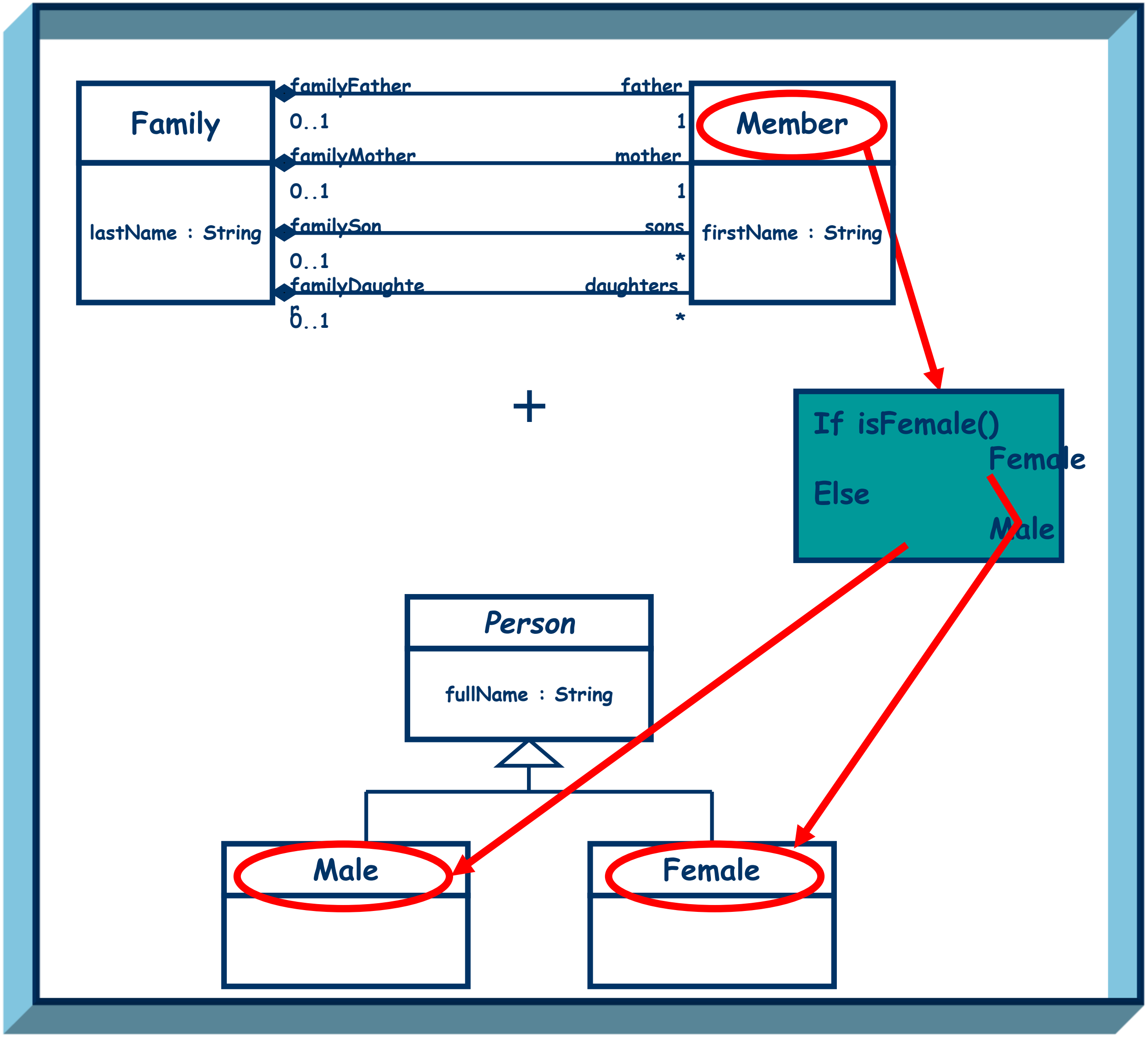
- After the helpers we now write the rules:
  - Member to Male

```
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
            )
}
```

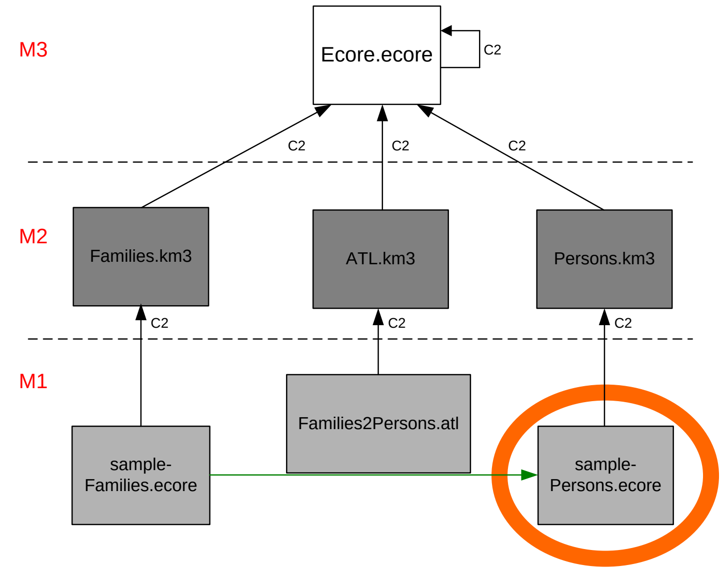
Summary of the Transformation



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

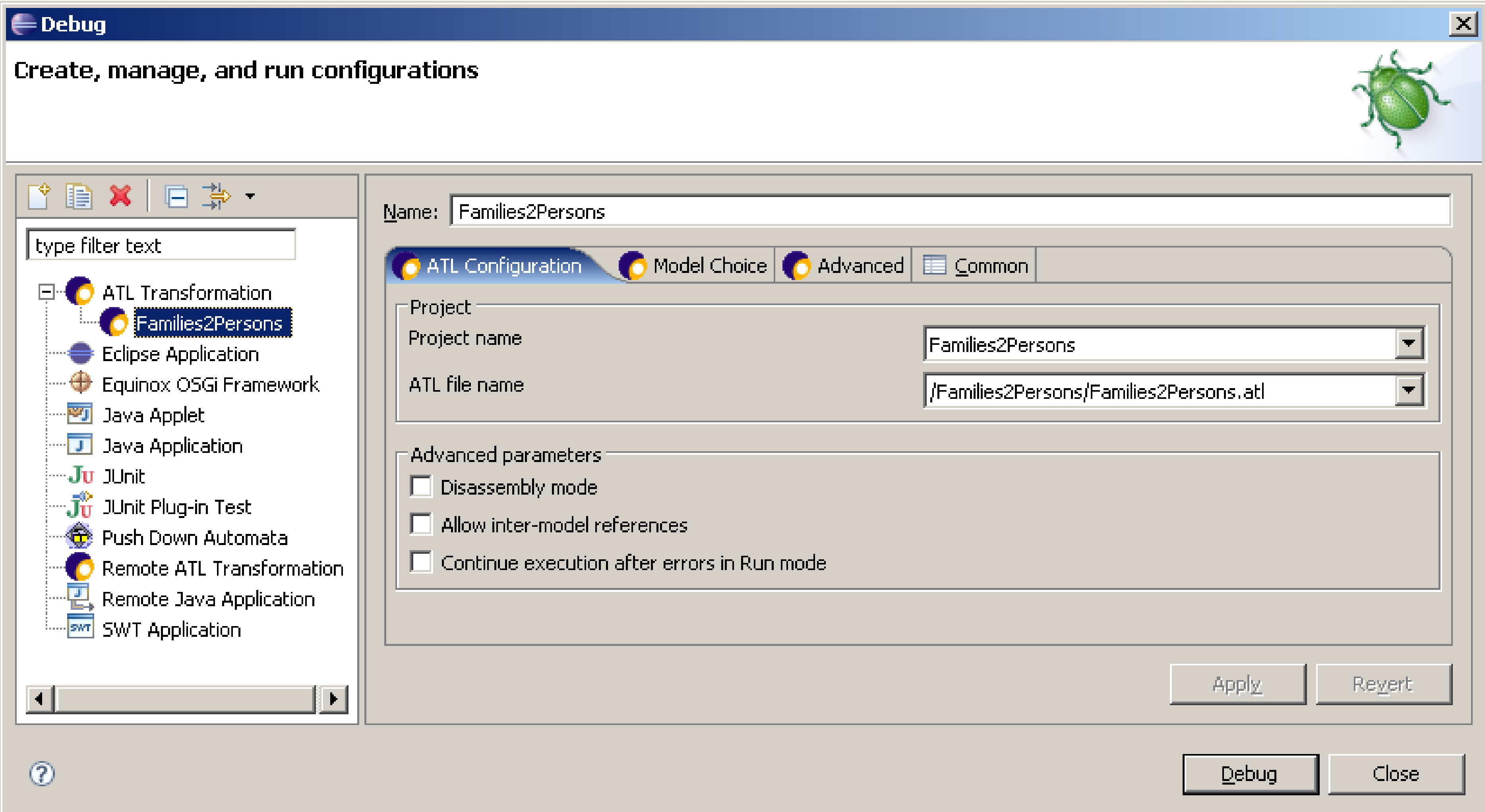
Eclipse Modeling Framework (EMF)



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

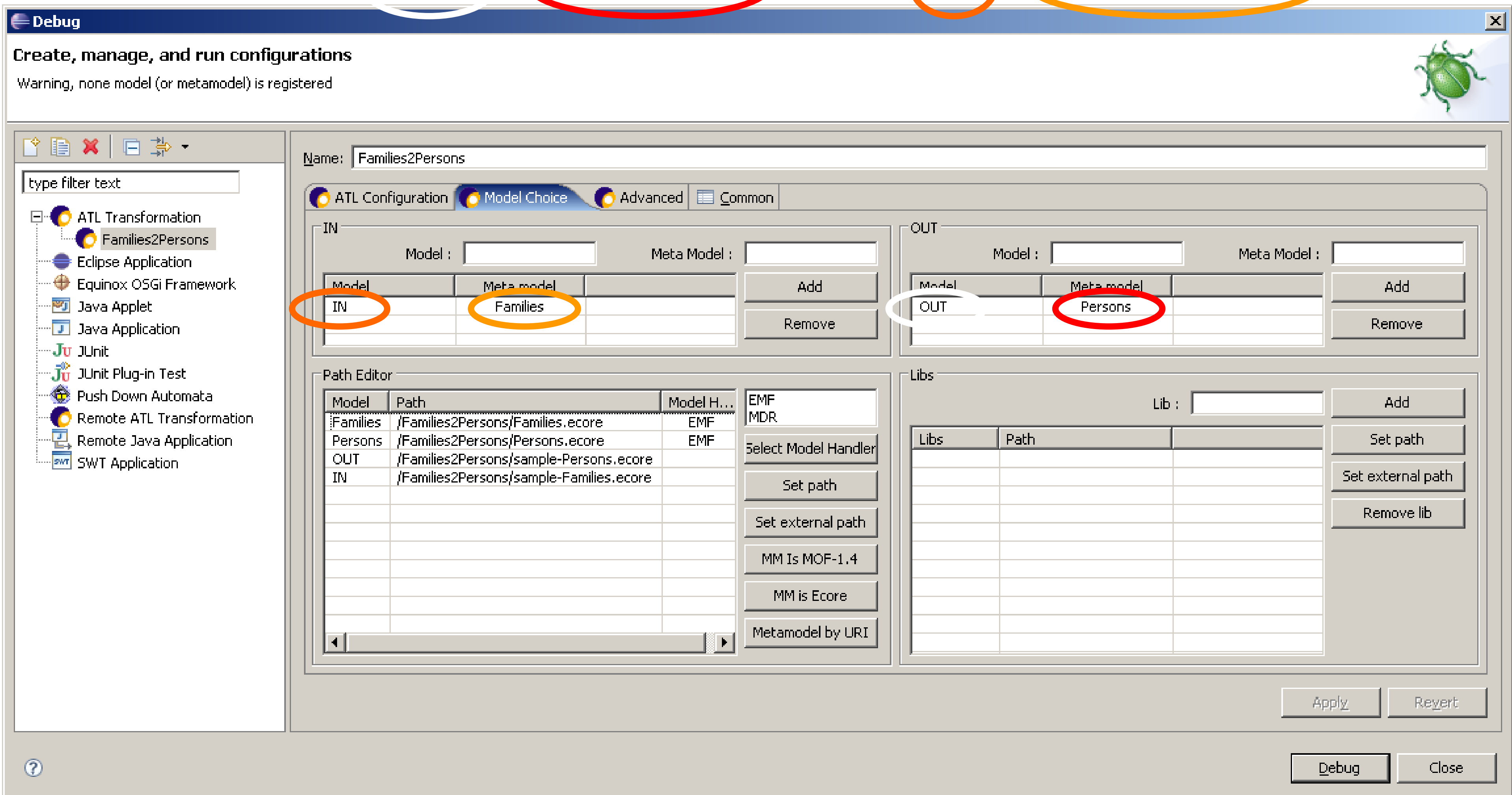
```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
xmlns="Persons">
  <Male fullName="Dylan Sailor"/>
  <Male fullName="Peter Sailor"/>
  <Male fullName="Brandon March"/>
  <Male fullName="Jim March"/>
  <Male fullName="David Sailor"/>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



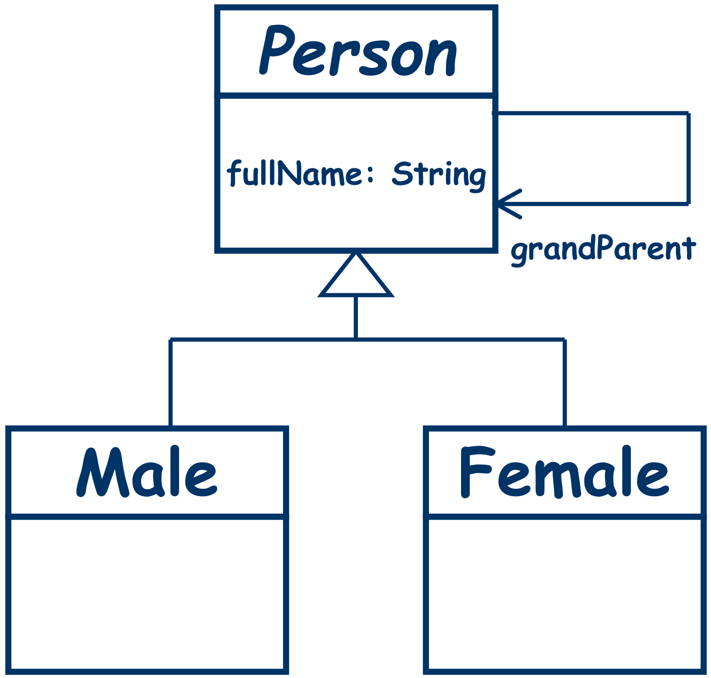
Summary

- 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.

Acceleo

https://en.wikipedia.org/wiki/Acceleo

HOME / PROJECTS / ACCELEO

About this project

- Wiki
- Newsgroup
- Project Plan
- Bugs
- File a Bug

Developers

- Git
- Gerrit
- Mailing List

Related Projects

- EMF
- Sirius

Acceleo

About Acceleo

Acceleo is a pragmatic implementation of the **Object Management Group (OMG) MOF Model to Text Language (MTL)** standard. You do not need to be an expert to start using the plug-ins and create your first code generator : using the provided example projects and the powerful completion feature of the Acceleo editor, it is very easy to get started and understand the basic principles.

Acceleo is the result of several man-years of R&D started in the French company Obeo. Junction between the OMG MTL standard, its team's experience with industrial code generation and the latest research advances into the M2T field, it offers outstanding advantages : High ability to customize, Interoperability, Easy kick off, and much more!

OBEO

ACCELEO FOR MARS IS NOW AVAILABLE

- **Download Acceleo**

The Eclipse Mars Modeling Package is the perfect package for modeling developers. **Download it** and install Acceleo and Sirius and you are ready to go to build a complete modeling workbench.

Tweets by @acceleo

acceleo Retweeted

Stéphane Bégaudeau @sbegaudeau

UML to Java Generator - The version 2.0 of the UML to Java Generator is finally here! tmblr.co/ZnKN_xiVy45Q #acceleo #eclipse

12 Apr

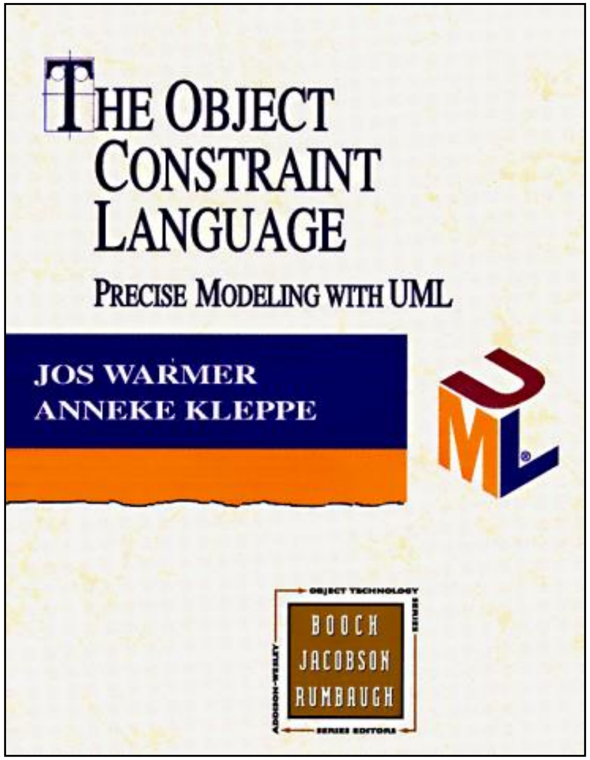
acceleo Retweeted

Laurent Broudoux @lbroudoux

Launching #acceleo generation from Maven : lbroudoux.wordpress.com/2012/07/24/lau...

UML OCL
Object Constraint Language

- The Object Constraint Language
  - ISBN 0-201-37940-6
- OCL home page
  - www.klasse.nl/ocl/index.htm



Model examples

ThiNgami
nos knnn
doZzzkf() karPhew(zAA)

Editor
text Font
changeFont(font) addElem(elem) spellCheck()

NuclearReactorCore
add(ControlRod, int) ControlRod remove(int)

Precise modeling – Details in models

- Avoid misunderstanding
- Completeness
- Baseline for code generation
- Model analysis
  - Consistence among models
  - Relationships and mappings between models
  - Analysis of models

Simplify with OCL

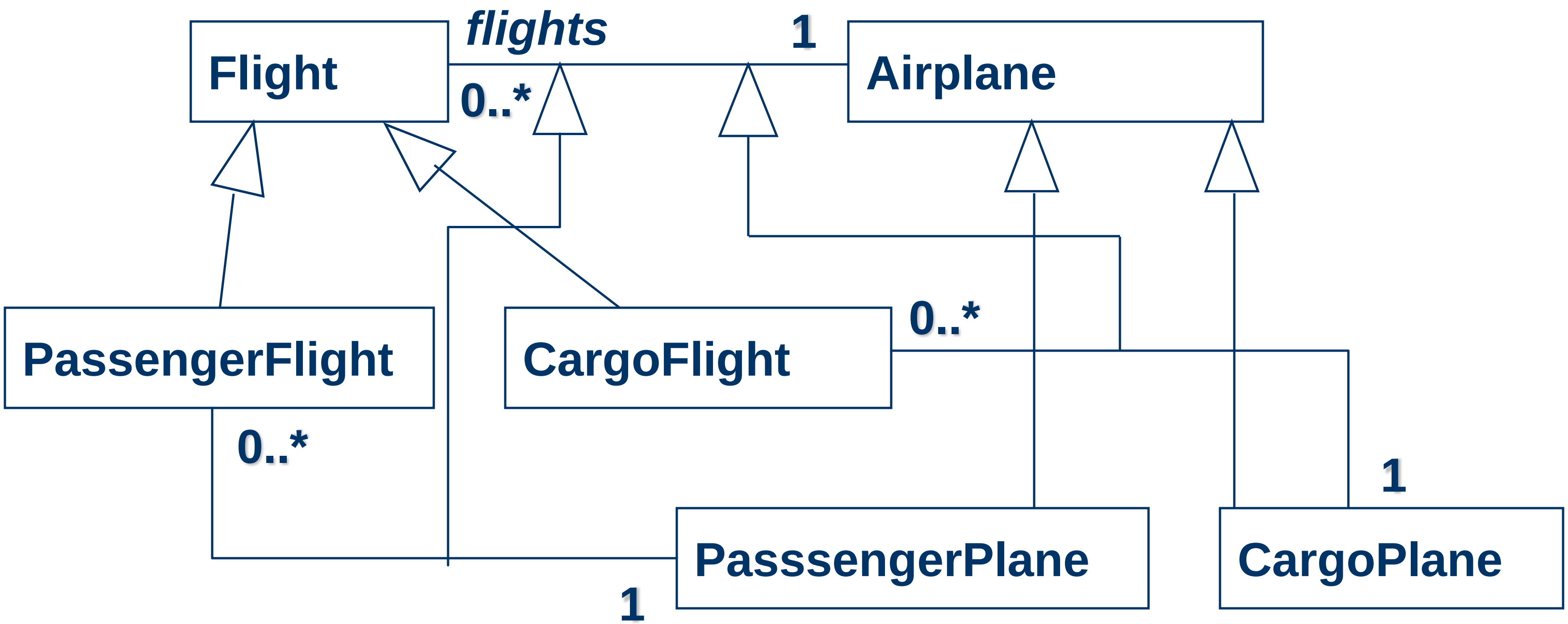
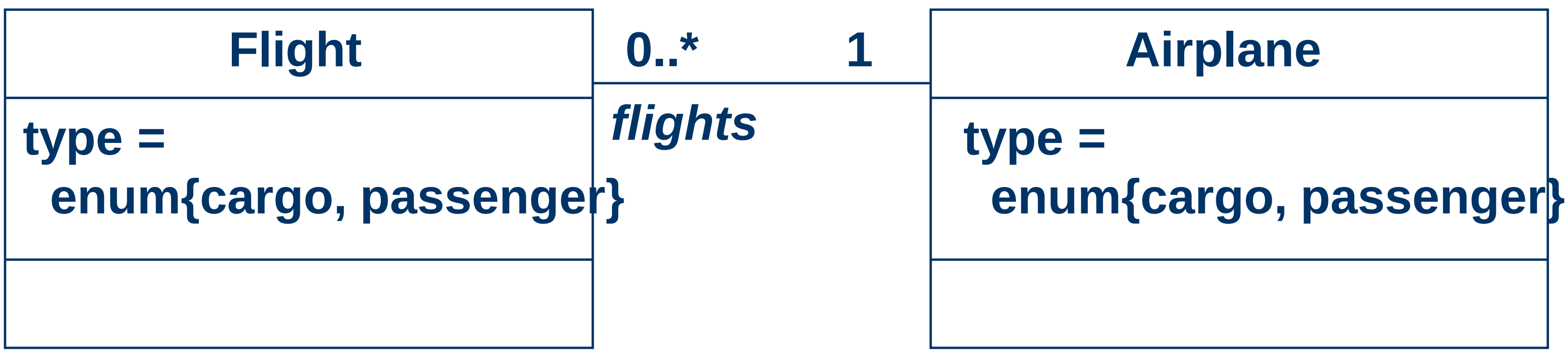


Diagram with invariants

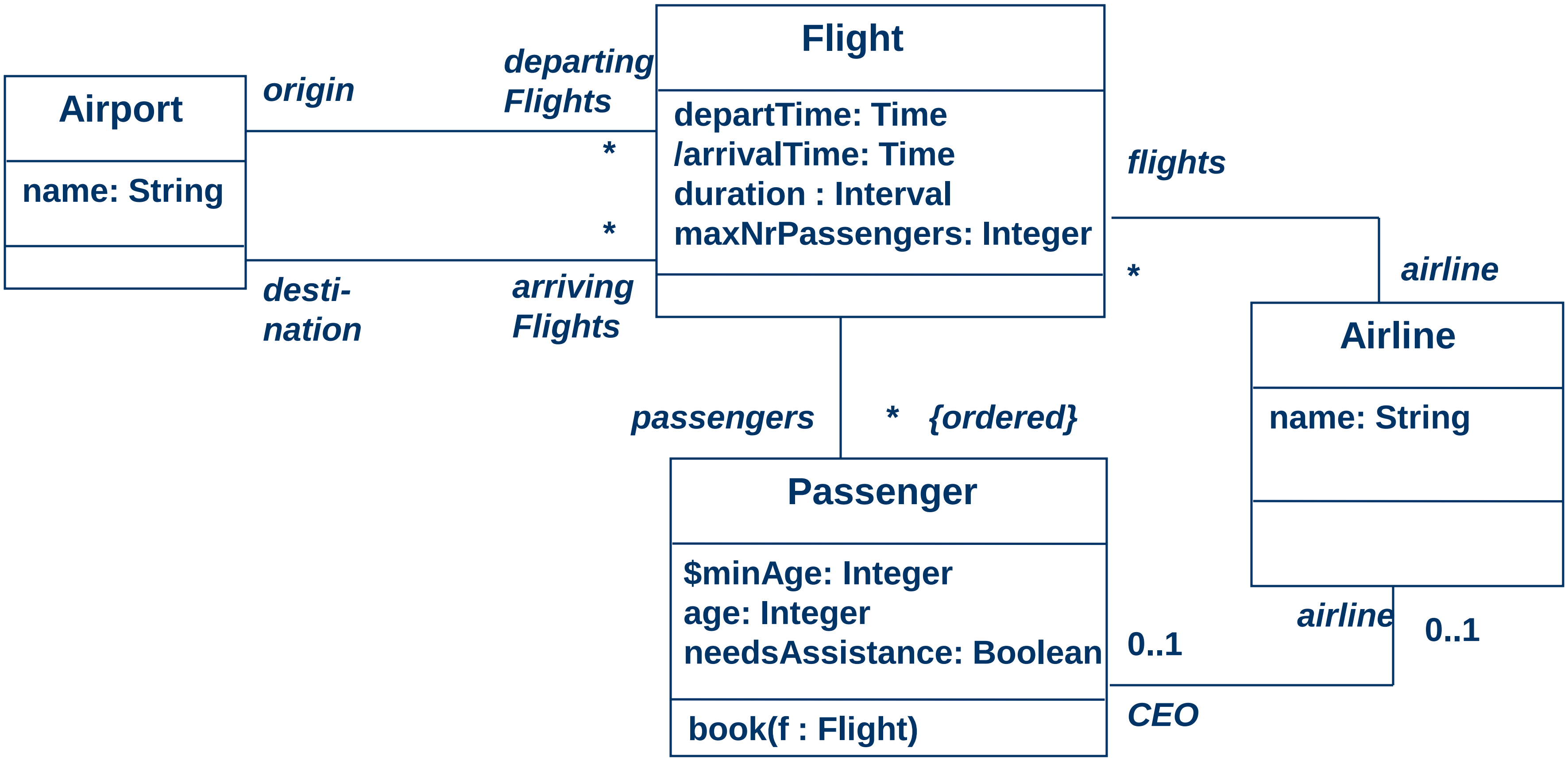


context Flight
inv: type = #cargo implies airplane.type = #cargo
inv: type = #passenger implies airplane.type = #passenger

Definition of constraint

- “A constraint is a restriction on one or more values of (part of) an object-oriented model or system.”

Example model



Constraint context and self

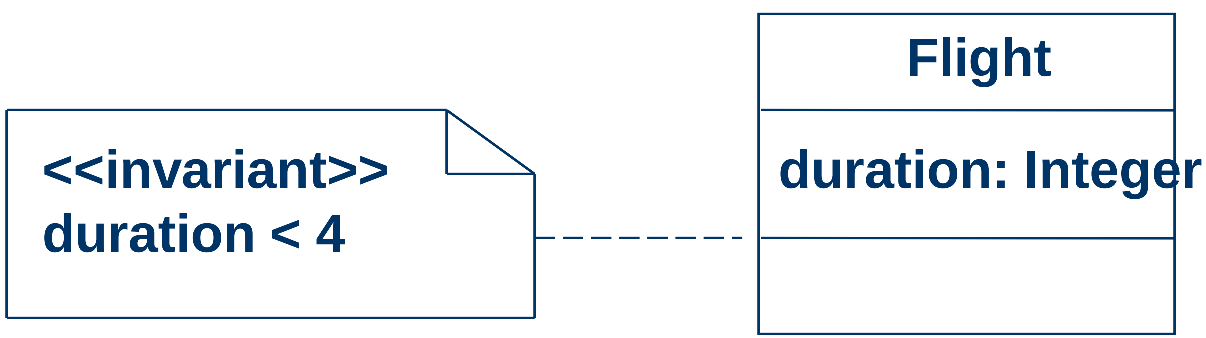
- Every OCL expression is bound to a specific context.
- The context may be denoted within the expression using the keyword ‘self’.

Notation

- Constraints may be denoted within the UML model or in a separate document.
  - the expression:

    context Flight inv: self.duration < 4
  - is identical to:

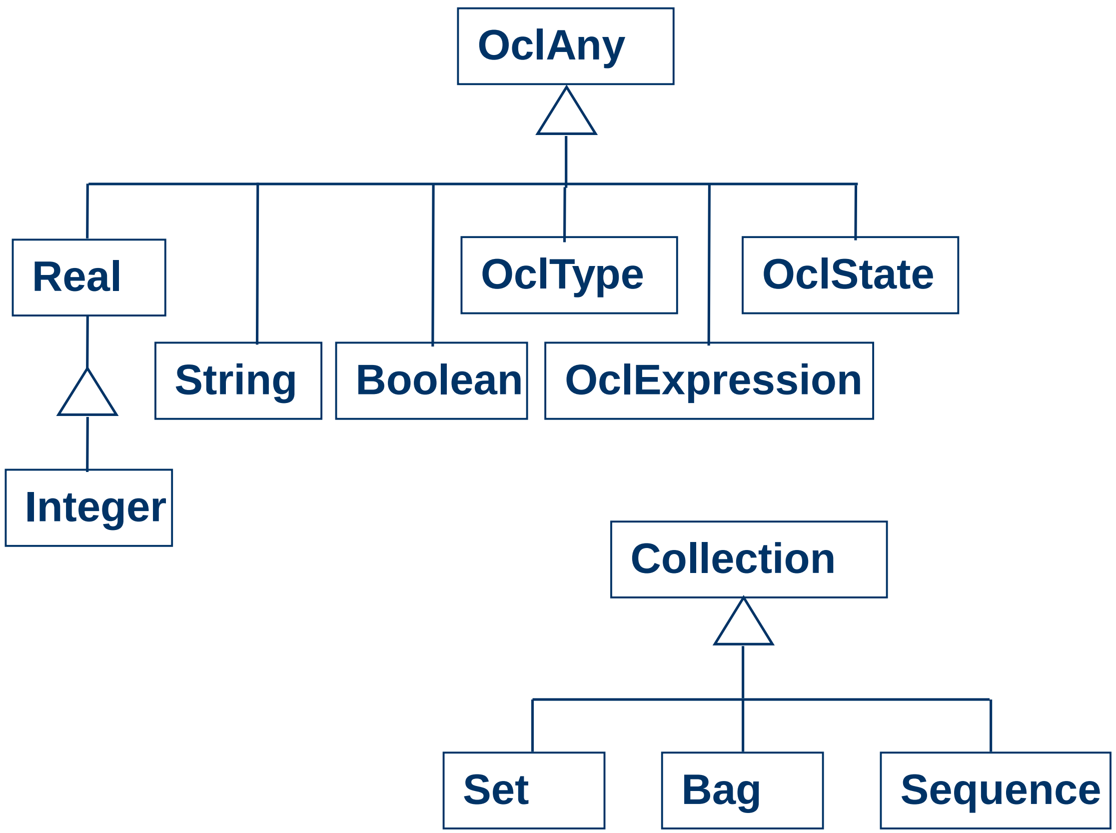
    context Flight inv: duration < 4
  - is identical to:



Elements of an OCL expression

- In an OCL expression these elements may be used:
  - basic types: String, Boolean, Integer, Real.
  - classifiers from the UML model and their features
    - attributes, and class attributes
    - query operations, and class query operations
  - associations from the UML model

OCL types


OclAny
Real
String
Boolean
OclType
OclState
OclExpression
Integer
Collection
Set
Bag
Sequence

Example: OCL basic types

context Airline inv:

name.toLower = ‘klm’

context Passenger inv:

age >= ((9.6 - 3.5)* 3.1).abs implies mature = true

Model classes and attributes

- “Normal” attributes

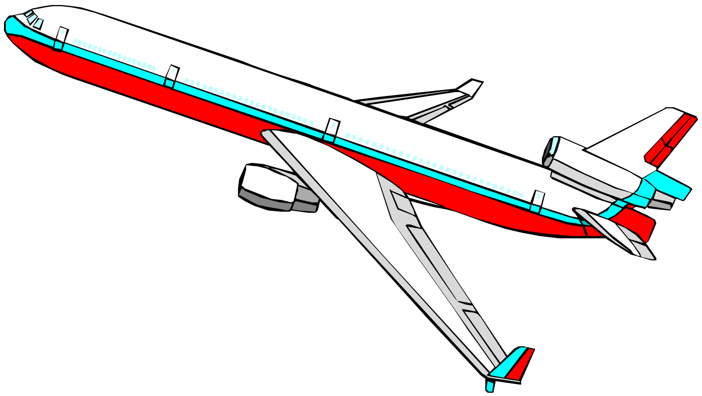
  context Flight inv:

  self.maxNrPassengers <= 1000

- Class attributes

  context Passenger inv:

  age >= Passenger.minAge



Example: query operations

context Flight inv:
self.departTime.difference(self.arrivalTime)
.equals(self.duration)

Time
$midnight: Time month : String day : Integer year : Integer hour : Integer minute : Integer
difference(t:Time):Interval before(t: Time): Boolean plus(d : Interval) : Time

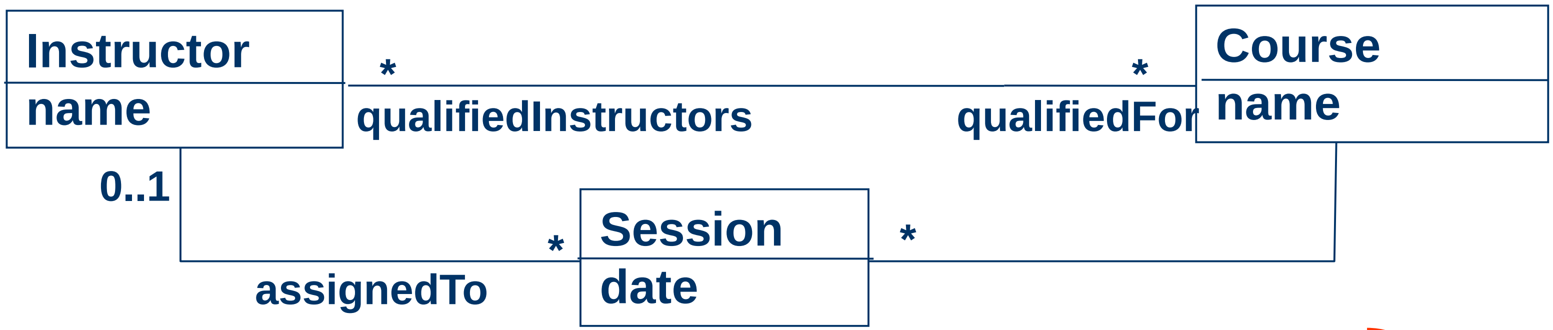
Interval
nrOfDays : Integer nrOfHours : Integer nrOfMinutes : Integer
equals(i:Interval):Boolean $Interval(d, h, m : Integer) : Interval

Example: navigations

- Navigations

```
context Flight
inv: origin <> destination
inv: origin.name = ‘Amsterdam’

context Flight
inv: airline.name = ‘KLM’
```

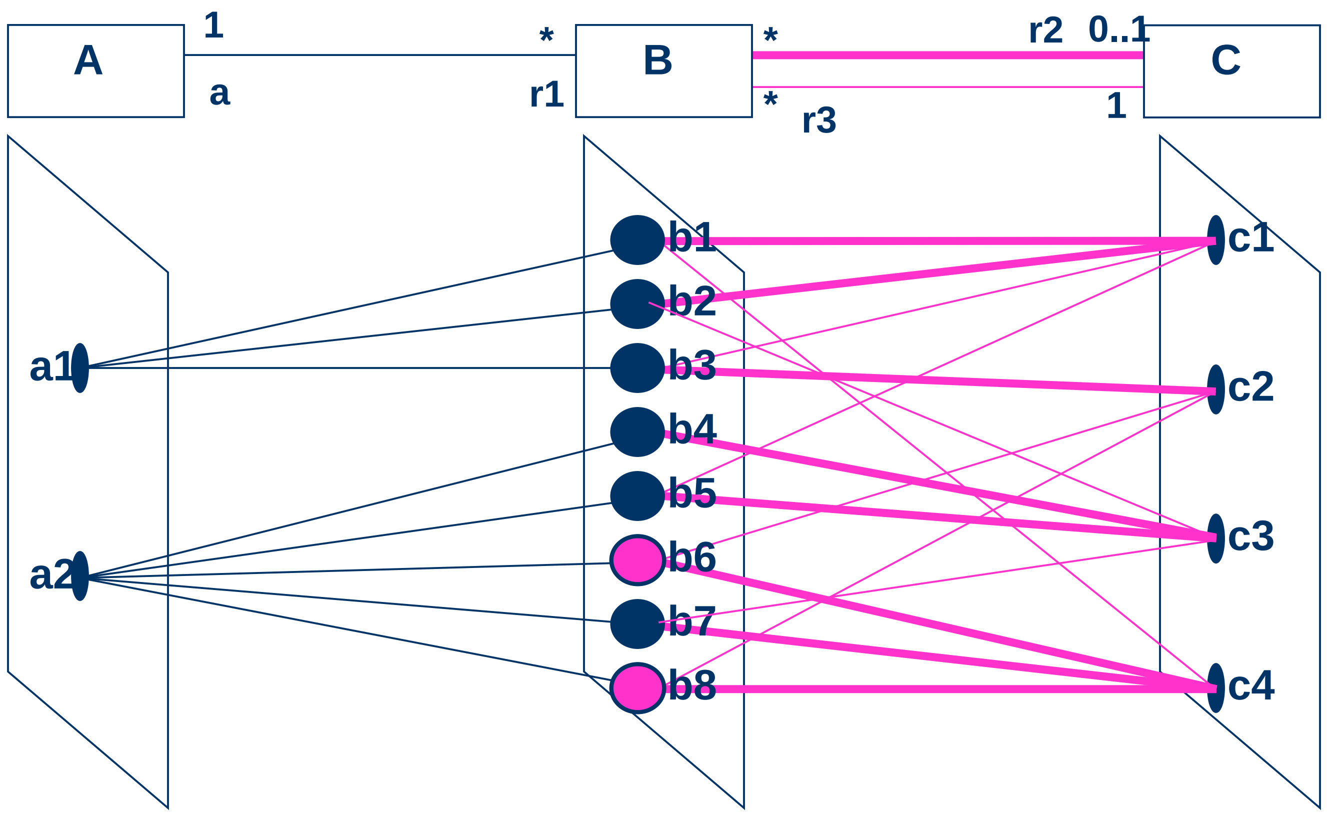
Basic "Navigation" expressions



- i: Instructor, c: Course, s: Session
  - The name of the course:
    - c.name
  - The date of the session:
    - s.date
  - The instructor assigned to the session:
    - s.instructor
  - The course of the session:
    - s.course
  - The name of the course of the session:
    - s.course.name
  - The instructors qualified for the session:
    - s.course.qualifiedInstructors

Let's navigate on a model

Navigation Example



- What does **a1.r1.r2.r3** yield?
- Assuming the B's have a boolean attribute "*black*"; black=false for b6, b8 - what expression refers from **a2** to the set { **b1** }

Association classes

context Person inv:

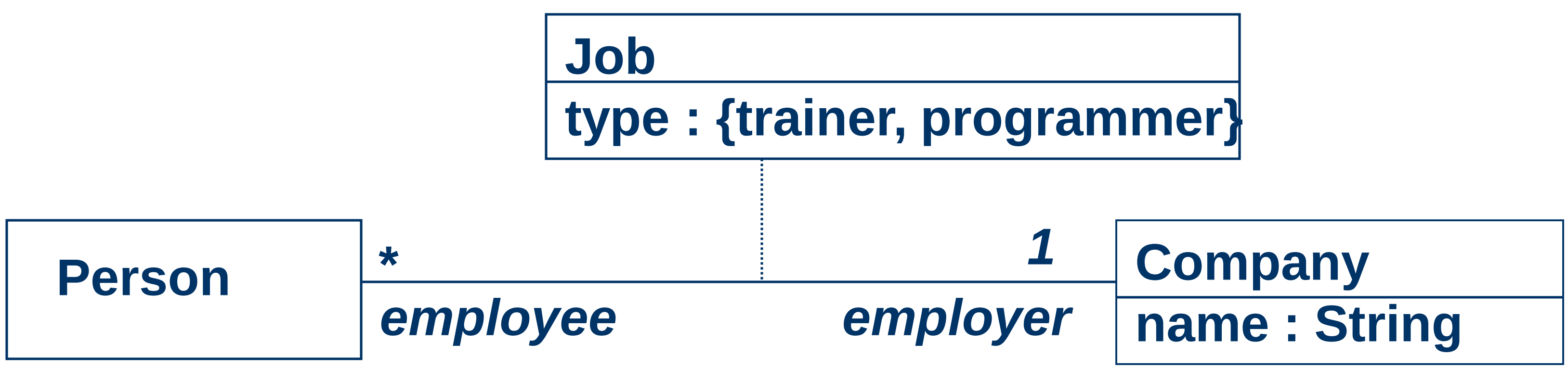
if employer.name = ‘Klasse Objecten’ then

    job.type = #trainer

else

    job.type = #programmer

endif



Three subtypes to Collection

- Set:
  - arrivingFlights(from the context Airport)
- Bag:
  - arrivingFlights.duration (from the context Airport)
- Sequence:
  - passengers (from the context Flight)

Collection operations

- OCL has a great number of predefined operations on the collections types.

- Syntax:

collection->operation

The collect operation

- Syntax:
  collection->collect(elem : T | expr)
  collection->collect(elem | expr)
  collection->collect(expr)
- Shorthand:
  collection.expr

- The *collect* operation results in the collection of the values resulting evaluating *expr* for all elements in the *collection*

The select operation

- Syntax:
  collection->select(elem : T | expression)
  collection->select(elem | expression)
  collection->select(expression)

- The *select* operation results in the subset of all elements for which *expression* is true

The forAll operation

- Syntax:

  collection->forAll(elem : T | expr)

  collection->forAll(elem | expr)

  collection->forAll(expr)

- The *forAll* operation results in true if *expr* is true for all elements of the collection

$\forall$

The exists operation

- Syntax:

  collection->exists(elem : T | expr)

  collection->exists(elem | expr)

  collection->exists(expr)

- The *exists* operation results in true if there is at least one element in the collection for which the expression *expr* is true.

$\exists$

Example: exists operation

```
context Airport inv:
self.departingFlights ->
                exists(departTime.hour < 6)
```

Other collection operations

- *isEmpty*: true if collection has no elements
- *notEmpty*: true if collection has at least one element
- *size*: number of elements in collection
- *count(elem)*: number of occurences of elem in collection
- *includes(elem)*: true if elem is in collection
- *excludes(elem)*: true if elem is not in collection
- *includesAll(coll)*: true if all elements of coll are in collection

Iterate example

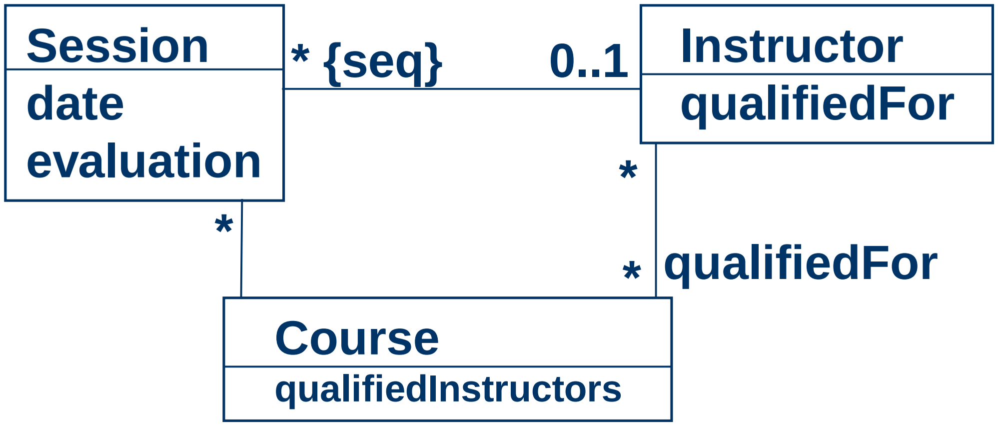
- Example iterate:

```
context Airline inv:
flights->select(maxNrPassengers > 150)->notEmpty
```

- Is identical to:

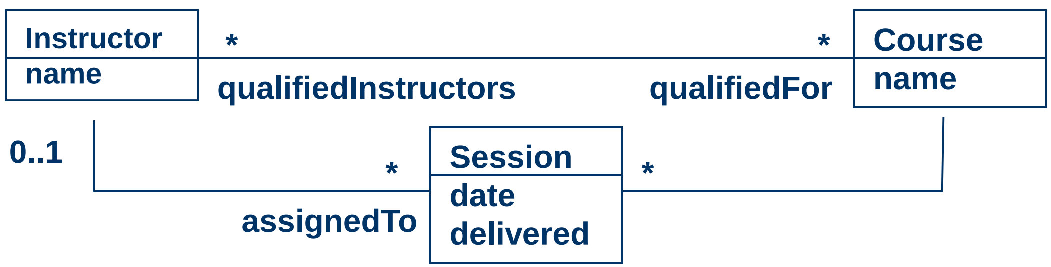
```
context Airline inv:
flights->iterate(f : Flight; answer : Set(Flight) = Set{ } |
if f.maxNrPassengers > 150 then
   answer->including(f)
else answer endif )->notEmpty
```

OCL — Navigation Details



- An association end with cardinality maximum > 1 yields a set or sequence
  - anInstructor.Session yields a sequence
  - anInstructor.qualifiedFor yields a set
- An association end with cardinality maximum of 1 yields an object or a set (with zero or one elements)
  - aSession.Instructor yields an object
  - aSession.Instructor->isEmpty yields a Boolean

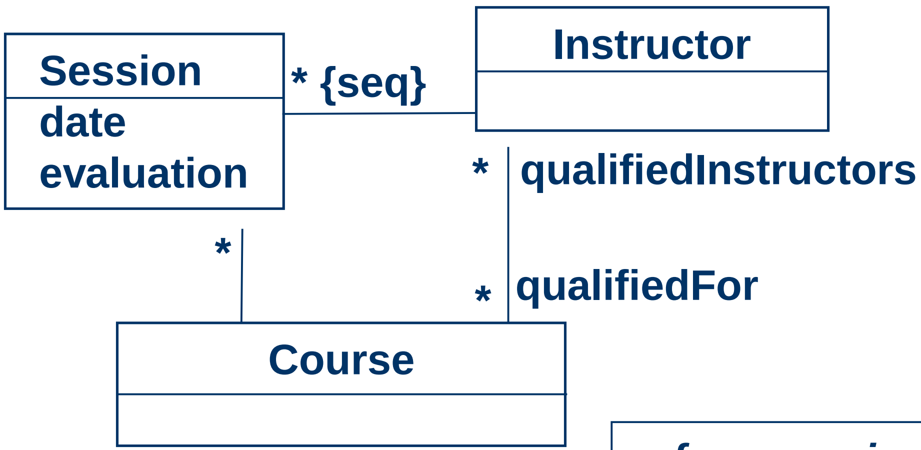
Collections use



Let's navigate on a model

- i: Instructor
- The courses an instructor is qualified to teach
  - Course.allInstances ->**select** (c | c.qualifiedInstructors ->**includes** (i))
- Sessions delivered by an instructor who is no longer qualified to teach it
  - Session.allInstances ->**select** (s | s.delivered **and**
    s.course.qualifiedInstructors ->**excludes** (s.instructor))
- The last can be simplified significantly with "convenience" attributes
  - Session.allInstances **->select** (s | s.teacherNotQualified)

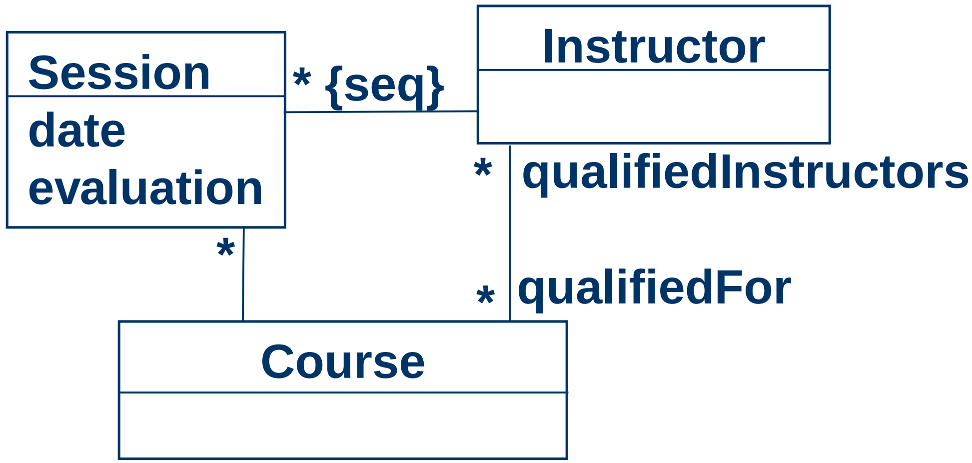
Another Invariant Formalized



-- for every instructor ...
Instructor::invariant
-- for any course
Course.allInstances->forAll (c |
-- if the evaluation bad
Session->select(Course=c)->forAll(s |
s.evaluation = bad
-- then instructor is disqualified for course
implies qualifiedFor ->excludes (c)))

Always combine formal and narrative descriptions

Same Invariant on Course



-- for every course ...
Course::invariant
-- for all sessions
Session->forAll (s |
-- if the evaluation is bad
s.evaluation = bad implies
-- then the instructor is not a qualified instructor
qualifiedInstructors->excludes(s.Instructor))

Operation Specification

Client
balance

SeminarSystem
pay(client:Client, amount: Money)

operation SeminarSystem::pay (**in** client:Client, **out** amount: Money)

-- When you pay off an invoice

pre -- *Provided the payment amount is not negative and does not*

-- exceed amount owed

client .balance >= amount **and** amount>=0

post -- *The balance is reduced by the amount of the payment*

client.balance@**pre** = client.balance + amount

let, new: Convenient Names, New Objects

Any specification can introduce local names using **let** … **in** ...

```
operation SeminarSystem::scheduleCourse
    (client: Client, date: Date, course: Course)
  let ( availableInstructors =
      instructors ->select (qualifiedFor(course) and availableOn(date)) )
   in (  -- the name "availableInstructors" can be used in pre or post
    pre         availableInstructors ->notEmpty
    post        -- some instructor from available instructors is assigned …
   )
```

Actions often result in the creation of a **new** object

```
let (s = Session.new) in ( -- s is a new member of Session type
   s.client = client and s.date = date and s.course = course
   and ….
)
```

OCL — Misc.

- Special words
  - **@pre** designates a value at the start of an operation

    total = total@**pre** + amount
  - **self** designates the object itself

    self.total = **self**.total@**pre** + amount
  - **result** designates the returned object (if any)

    result = total
- Comments
  - -- Two hyphens start a comment that goes through the end of line

OCL Tools

- Cybernetics
  - ww.cybernetic.org
- University of Dresden
  - www-st.inf.tu-dresden.de/ocl/
- Boldsoft
  - www.boldsoft.com
- ICON computing
  - www.iconcomp.com
- Royal Dutch Navy
- Others … …

Conclusions and Tips

- OCL invariants allow you to
  - model more precisely
  - stay implementation independent
- OCL pre- and postconditions allow you to
  - specify contracts (design by contract)
  - precisely specify interfaces of components
- OCL usage tips
  - keep constraints simple
  - always combine natural language with OCL
  - use a tool to check your OCL

UML og OCL

- Skriver OCL som tilleggsdokumentasjon til modeller
- Skriver OCL i Constraints
- (Verktøy)problem: hvordan bruke aktivt
  - forfining
  - konsistens
  - kodegenerering

OclAny

x,y:OclAny; T is a OclType	
x = y	x and y are the same object
x < > y	not (x=y)
x.oclIsNew	True if x is a new instance
x.oclType	The type of x
x.isKindOf(T)	True if T is a supertype (transitive) of the type of x
x.isTypeOf(T)	True if T is equal to the type of x
x.asType(T)	Results in x, but of type T.

OclType and operators

T is a OclType	
T.new	Create a new instance of type T
T.allInstances	All of the instances of type T

- Logical operators in Boolean expressions
  - and, or, xor, not, implies

Collection (1)

c,c2 : Collection(T); x,e:T; P:T$\rightarrow$ Boolean;	
f, f2: T $\rightarrow$ Object	
c->size	Number of elements
c->sum	Sum of elements (elements must support addition)
c->count(e)	Number of times e is in c
c->isEmpty	c->size = 0
c->notEmpty	not c->isEmpty

Collection (2)

c->includes(e)	True if e is in c
c->includesAll(c2)	True if c2 in c
c->excludes(e)	True if e not in c
c->exludesAll(c2)	True if none in c2 is in c
c->exists(P)	True if an e makes P true
c->forAll(P)	True if P true for all e in c
c->isUnique(f)	True if f evaluates to different value for all e in c
c->sortedBy(f)	Sequence sorted by f
c->iterate(x;e=f;f2)	Iterate x over c and apply f2, initialise e to f

Collection subtypes (1)

Applies to set and bag
set, bag: Collection; e,x:T; P: T→Boolean;
f, f2: T→Object

set->union(set2)
set->union(bag)
set = set2
set->intersection(set2)
set->intersection(bag)
set – set2
set->including(e)

Collection subtypes (2)

set->excluding(e)	
set->symmetricDifference(set2)	
	The set of elements in set or set2, but not in both
set->select(x|P)	All elements for which P is valid
set->select(P)	Same as set->select(self|P)
set->reject(x|P)	Same as set->select(x|not P)
set->reject(P)	Same as set->select(self|not P)
set->collect(x|f)	The bag of elements which results from applying f to every member of set
set->asSequence	
set->asBag	

Sequence

seq->append(e)	seq followed by e
seq->prepend(e)	e followed by seq
seq->subSequence(lower, upper)	Subsequence in range [lower, upper]
seq->at(i)	Element at position i
seq->first	seq->at(1)
seq->last	seq->at(seq->size)