INF5120 – Eclipse and MOFScript

Andreas Limyr
Gøran K. Olsen
23.03.06

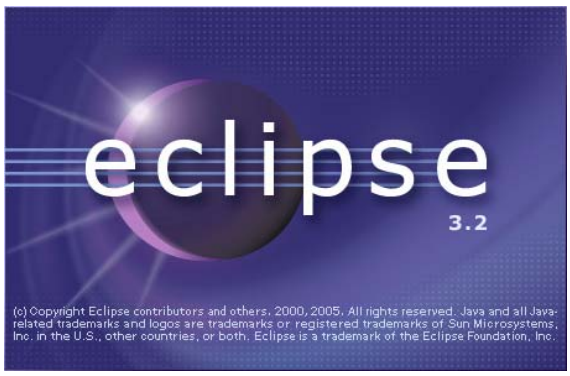
Plan

- Eclipse. What, how and why?
  - What is Eclipse and the Eclipse way of thinking?
  - How can I use Eclipse?
  - Why should I learn Eclipse?
- Some popular Eclipse projects
  - EMF, UML2, GEF, GMF
- MOFScript
  - Introduction and hands-on demonstration

Eclipse

- Eclipse is a kind of universal tool platform - an open extensible IDE for anything and nothing in particular.

www.eclipse.org



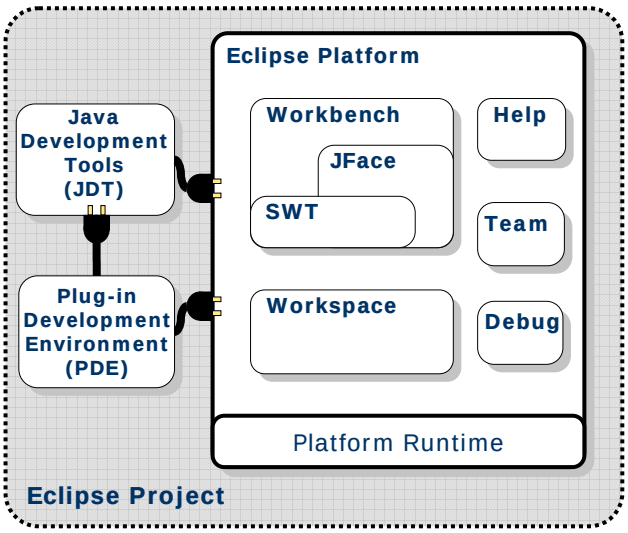
Eclipse

- Platform independent integrated development environment (IDE)
- Open source project
  - Free, originated from IBM
- Highly extensible
  - Plug-ins
- Fast development of tools
  - Many frameworks available

Eclipse SDK

- Eclipse Platform
  - The core framework with runtime, graphics, UI and more
- JDT (Java Development Tools)
  - Adds Java program development capability to the Platform
- PDE (Plug-in Development Environment)
  - Adds Plug-in development capability to the Platform

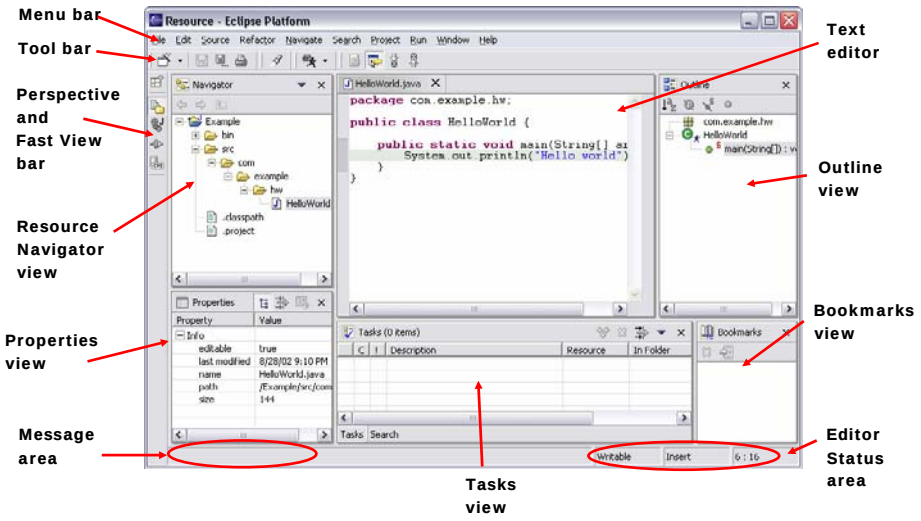
Eclipse SDK



Eclipse terminology

- Some of the most used terms are
  - Workspace – Directory where the work will be stored
  - Workbench – Running instance of Eclipse platform
  - View – Presentation of information
  - Editor – An area to manipulate data
  - Perspective – Composition of views, editors, menus and tool bars
  - Plug-in – Bundles of code and/or data that contribute function to Eclipse
  - Feature – Set of plug-ins
  - Extension point – define points for the platform that other plug-ins can plug into
  - Extension – Contribution of code and/or data according to an extension point

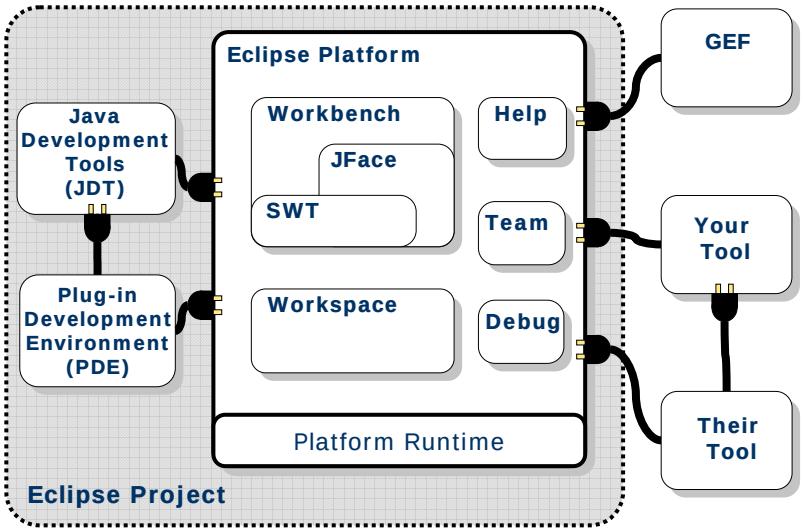
Workbench overview



Plug-ins

- Eclipse is extended with plug-ins
  - Plug-ins can contribute new functions to Eclipse
- Java programming is performed with a plug-in called JDT
- Plug-ins code is mostly written in Java
  - Usually packaged as a JAR (Java ARchive) file
  - Can also be without code, only resources (Help plug-in in HTML)
- It is possible to extend an existing plug-in with another plug-in
  - Extensions and extension points

Eclipse architecture



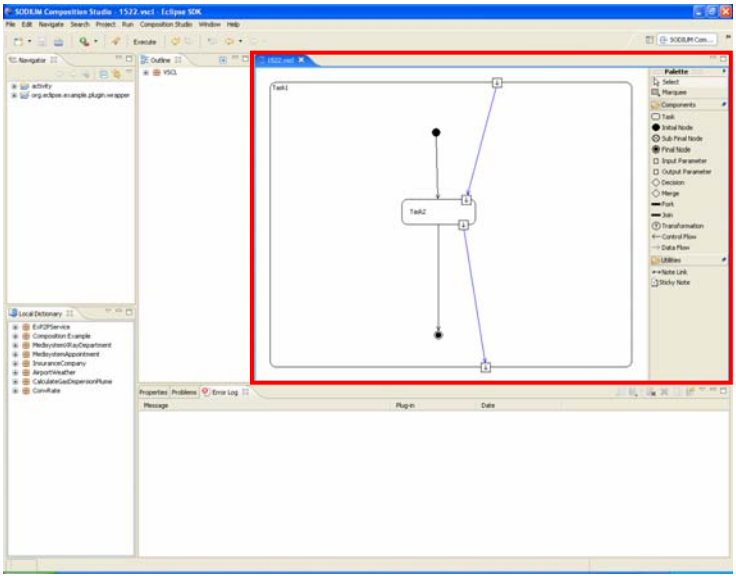
Plug-in installation

- There are two standard ways of installing plug-ins
- Install using the built-in update manager
  - The update manager downloads and installs the plug-ins
- Install manually
  - Acquire the plug-in and place it under the plugins folder in your Eclipse installation (for jar files)
- After installing a plug-in Eclipse needs to restart to register the plug-ins

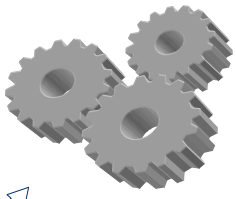
GEF – Graphical Editing Framework

- Support for creating rich graphical editors with use of a MVC (model-view-controller) architecture
- Two plug-ins
  - org.eclipse.draw2d
    - a layout and rendering toolkit for displaying graphics
  - org.eclipse.gef
    - Tools, palette, viewers, controller framework, undo/redo support ++

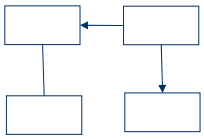
GEF in action



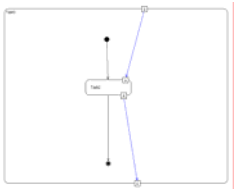
GEFs MVC principle



Controller



Model



View

EMF – Eclipse Modeling Framework

- Unifying Java, XML and (almost) UML
- EMF models are essentially simplified UML Class Diagrams
- EMF generates Java code based on these models
  - Must first create something called a *.genmodel
  - Supports only structural modeling
- Standard serialization is in the form of XMI
- “EMF is MDA on training wheels”

EMF Models and Ecore

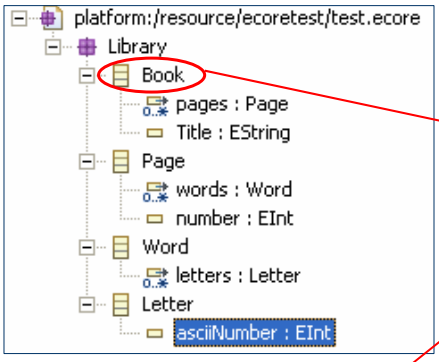
- Ecore is the model used to represent EMF models
- Ecore is also an EMF model and therefore its own metamodel
  - And its own meta-meta-….-model, but never mind
- Available elements are:
  - EClass
  - EAttributes
  - EReference
  - EDataType
  - EEnum, EEnum Literal
  - EPackage
  - EOperation, EParameter

Creating your model

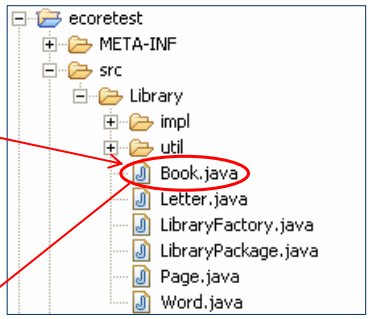
- Can be defined in four ways
  - Java
  - XML Schema
  - Directly manipulate the model (the almost UML way)
  - Export it from a supported tool (RSM)
- Both Java and XML Schema approach builds an EMF model
- Editing the model can be done with EMF's simple tree editor
  - It is also possible to import Rational Rose (.mdl) files
  - Or use the GEF example EDiagram (comes with GEF Examples)
  - Or ECore diagram editor from GMF (not yet finalized)

3 shades of EMF

Ecore model



Generated Java files



Creation of an instance

```
Book book = LibraryFactory.eINSTANCE.createBook();
book.setTitle("How to be a meta role model");
```

Make a model – Part 1: EMF Model

- With the help of wizards it is possible to create your first EMF model
  - Create a new Empty EMF project
    - Empty EMF Project Wizard
  - Create a new folder named model
    - Folder wizard
  - Create an Ecore file
    - Ecore Model wizard
  - Create a new EMF Model based on your Ecore file
    - EMF Model wizard

Make a model – Part 2: Java

- It is possible to create an EMF model from annotated Java
  - Use the
- Example of EMF annotated Java

```
package org.eclipse.example.library;
import java.util.List;
/** * @model */
public interface Library {

        /** * @model */
        String getName();

        /** * @model type="Writer" containment="true" */
        List getWriters();

        /** * @model type="Book" containment="true" */
        List getBooks();
}
```

Make a model - Part 3: XML Schema

- It is also possible to import an XML Schema (XSD) to generate a EMF model
- Serialization of the model instances can be conformed to the XML Schema
- You can therefore generate a Java API for reading, writing and manipulating your XML files based on a certain XML Schema

Generate code from ecore model

- From the *.genmodel it is possible to generate code
- Open the genmodel and right click on the root of the file
- Select Generate Model Code
  - It is also possible to generate edit, editor and test code
- The generated model code can be used to create instances of the ecore model

Use generated code: Programmatically

- How to create an instance of a book with one page
- Create a new Class in your source folder
- Create a new Book

```
Book book = LibraryFactory.eINSTANCE.createBook();
book.setTitle("How to be a meta role model");
```

- Create a new Page

```
Page page = LibraryFactory.eINSTANCE.createPage();
page.setNumber(1);
```

- Add page to Book

```
book.getPages.add(page);
```


Use generated code: Editor

- Open the genmodel again

- Select Generate Edit Code
- Select Generate Editor Code

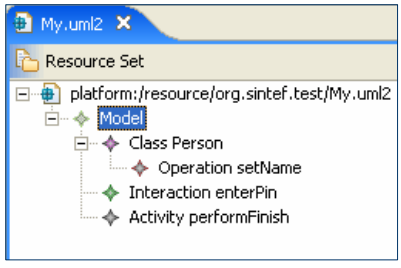
- Run the project as an Eclipse application
- Start wizard
  - Found under "Example EMF Model Creation Wizards" and typically named "name of your ecore package" model
- Can be used to create instances of the ecore model

UML2

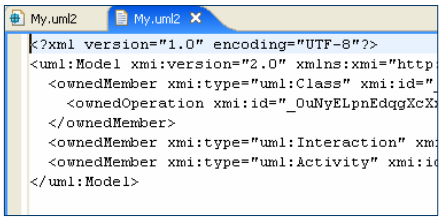
- UML2 is an EMF-based implementation of the UML 2.0 meta-model
- Gives Java API support to manipulate UML 2.0 models
- Comes with a tree editor
  - RSM has a richer graphical diagram support for UML2
  - GMF will provide an example UML2 diagram upon release

UML2 Examples

- Tree editor

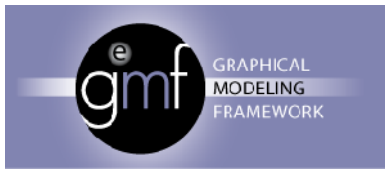


- XMI

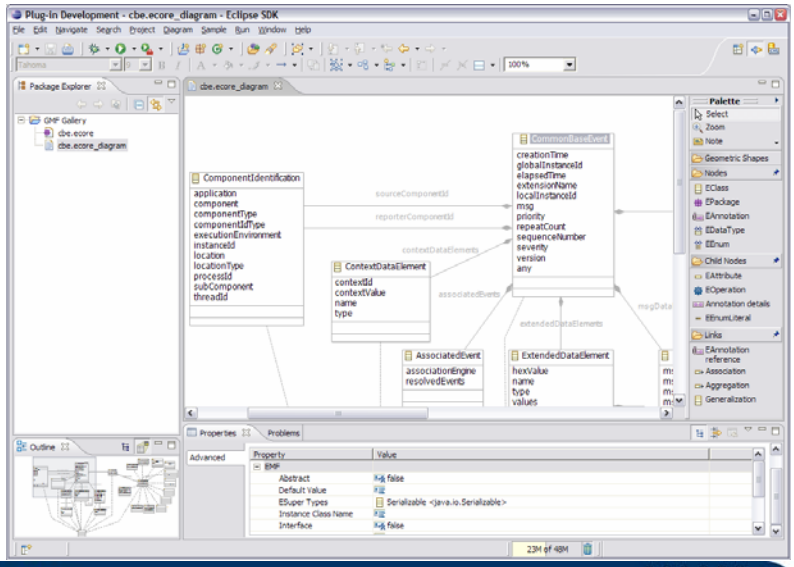


GMF – Graphical Modeling Framework

- Utilizes EMF and GEF to support generation of graphical editors
  - GEF – Graphical Editing Framework
  - EMF – Eclipse Modeling Framework
- Basic idea:
  - Bring your own model
  - Define graphical diagram notation
  - Define your tools
  - Map model to diagram
  - Generate editor



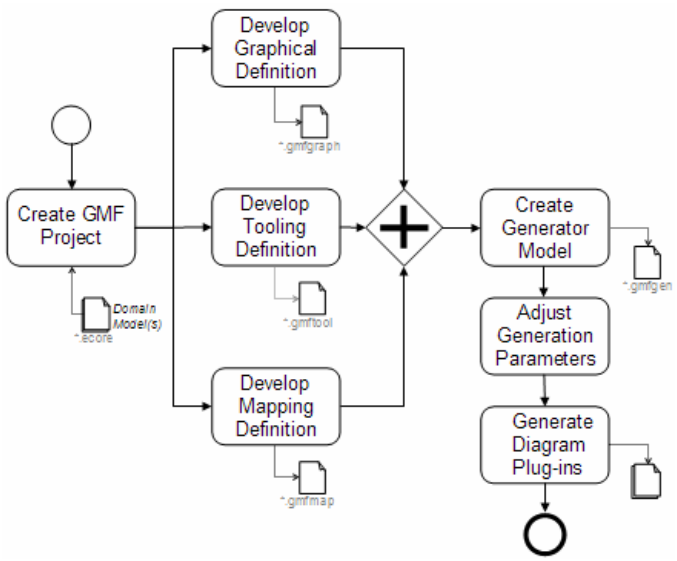
GMF ecore diagram example



Why GMF?

- Fills an architectural gap between EMF and GEF
- Opportunity for the community to reuse high quality standardized components
  - "Compartment" figures
  - Diagram Assistants
  - Common tools
- Complementary to other emerging technologies like Domain Specific Languages (DSL)
  - Enables quick generation of visual design and modeling surfaces in Eclipse
  - Language creation is time consuming, automation is important: Cost, testing, updates

Simplified workflow



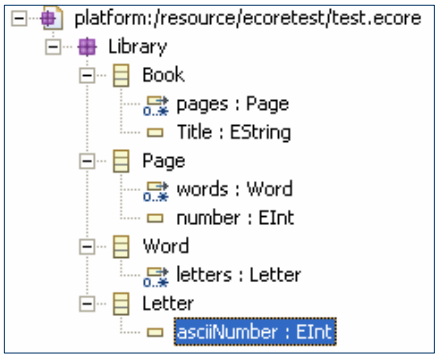
GMF – Quick intro

- Start by creating a new GMFGraph Model
  - GMFGraph is where you define your graphical elements
  - Choose canvas
  - Create a figure gallery with the different shapes you want to use
  - Create a node for each class
  - Create a connection for each reference
- Then create a new GMFTool Model
  - GMFTool is where you define your palette
  - Create creation tools for each of the elements defined in the GMFGraph model

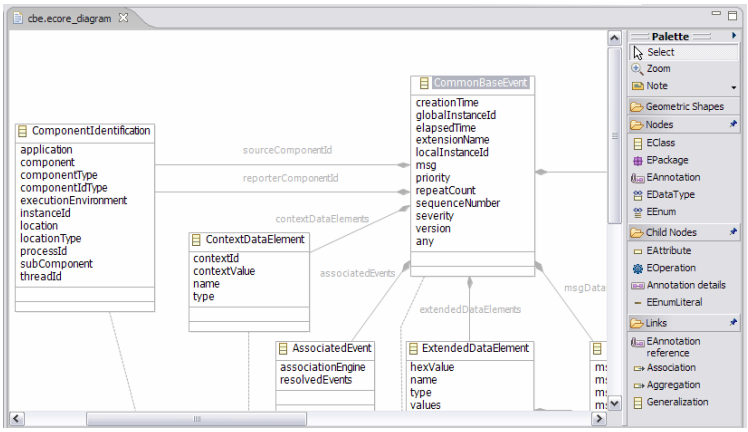
GMF – Quick intro

- Create a GMFMap Model
  - This will define mapping of your ecore model and diagram def.
  - Select mapping
  - Load resources → ecore, gmfmap, gmftool
  - Create canvas mapping
  - Create node mapping
  - Create link mapping
  - Generate genmodel
- Generate diagram code
  - Test diagram…

Before GMF – EMF tree editor



After GMF – Graphical Editor



But…

- GMF is not finalized as of yet
- It is possible to download it from eclipse.org
  - A work in progress
- Will not solve all problems, but can help speed up modeling tool creation
- Release is set to be around June/July 2006
  - Along with Eclipse 3.2 ++
  - Part of the Callisto release

Resources

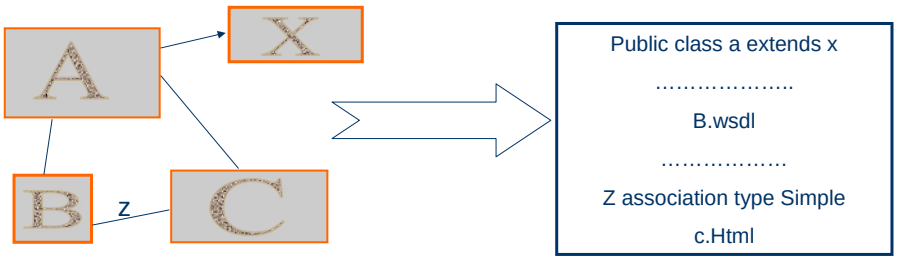
- A good starting point to learn more is
  - www.eclipse.org
- More info on each project can be found at
  - EMF www.eclipse.org/emf
  - UML2 www.eclipse.org/uml2
  - GEF www.eclipse.org/gef
  - GMF www.eclipse.org/gmf
- Don't forget the built in Help function in Eclipse
  - A good place to browse the javadoc for installed features

MOFScript

MOF-Model to text transformation

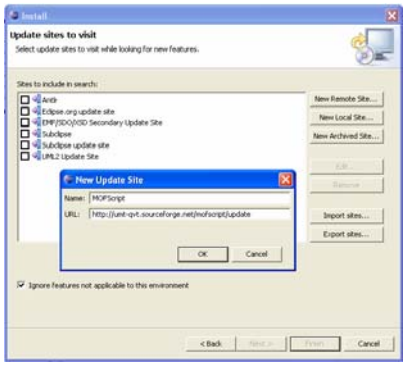
Introduction

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



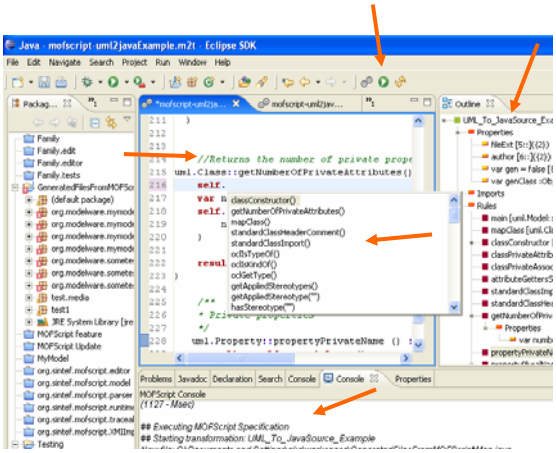
MOFScript installation

- Simply use the update functionality in Eclipse / RSM
  - Eclipse requires EMF
  - RSM only MOFScript
- Location:
  - ***http://download.eclipse.org/technology/gmt/mofscript/update/***



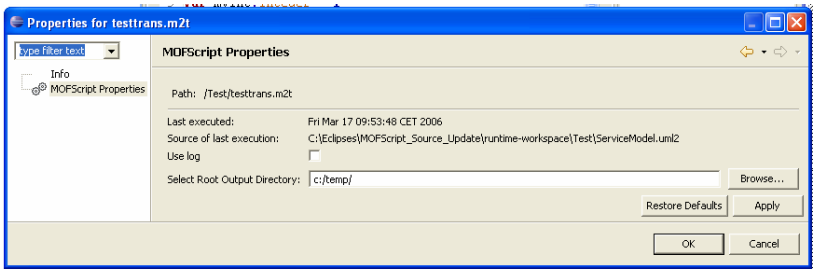
MOFScript a model to text tool

- Provides the means of:
  - Editing, compiling and executing
- Syntax high-lightning
- Content assist (meta-model and rules)
- Outline
- MOFScript Console



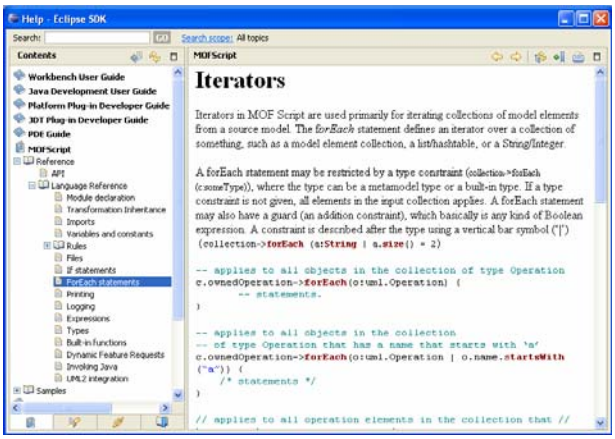
MOFScript properties

- Right-click transformation file to choose output directory
  - Do not use root generation directory from window-preferences



MOFScript help

- Use the help functionality provided



Structure of a MOFScript Transformation

```
texttransformation myTransformation (in uml: "UML2" ) {
    //(in uml:"http://www.eclipse.org/uml2/1.0.0/UML")
    property ext =".java"
    var number:Integer = 10

    //Context of the rule is uml.Model
    uml.Model::main(){
    //Statements
    }

    uml.Package::mapPackage(){
    //Statements
    }

    //Context of the rule is uml.Class
    uml.Class::makeJavaFile(){
    //Statements
    }
}
```

The context of the rule decides what rules / operations that are available in content assist. (According to the meta-model and the rules with matching context)

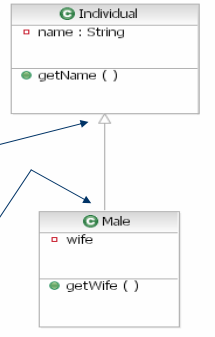
Uml2Java Example

```
//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;
}
```

Generalization 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()
```



Class name: Male
Attribute name: wife
Operation name: getWife
Generalization target name: Individual
Generalization source name: Male

Accessing UML meta-model elements

- uml.Package:
  - self.name
  - self.hasStereotype("Service")
  - self.getAppliedStereotypes()
  - selfOwnedMember->forEach(c:uml.Class)
- uml.Class:
  - self.name
  - self.hasStereotype("Controller")
  - self.visibility
  - self.ownedAttributes->forEach (p:uml.Property)
  - self.ownedOperations->forEach (o:uml.Operation)
  - self.generalization->forEach (g: uml.Generalization)
- uml.Operation:
  - self.visibility
  - self.ownedParameters->forEach (param: uml.Parameter)

This is not a complete list, but an example to illustrate the similarities

More advanced features

- Dynamic feature requests
- Invoking Java
- Inheritance super and sub transformations
- The MOFScript Meta-model repository

Dynamic feature request

- In some cases, a meta-model contains features that conflicts with the keywords in MOFScript.
- In these cases, a special construct can be used to gain access to that feature, the ***'getFeature("feature name")'*** operation.
- Using this operation, the conflicting features can be access without compilation errors.

Invoking Java (Black box operations)

- MOFScript has built-in support for invoking external Java code, which enables the integration of external *(black box)* operations from within MOFScript. This is done with the *java* operation.
- The syntax is as follows:

```
Java (String className, String methodName,
      List/Something parameters, String classpath)
```

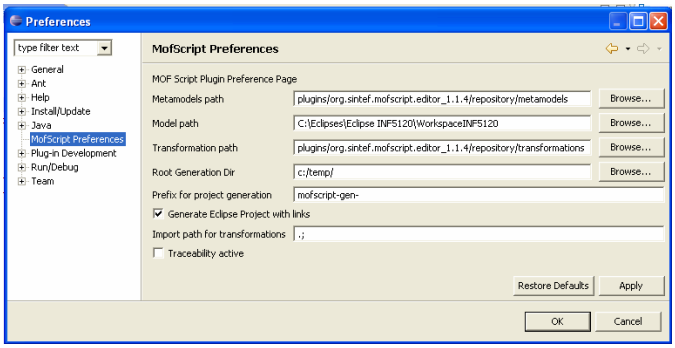
- The method invoked may be static or class scope.
- If it is non-static, the class must have a default constructor.
- The parameters may be *null*, a single parameter (e.g. a String, an integer etc) or a List of parameters if the method takes several parameters.

MOFScript meta-model repository

- Two repositories
  - Global built-in from Eclipse/EMF
  - File-based which can be configured by the user
- Adding your own meta-model (e.g wsdl) is simply done by copying the model into your directory: eclipse/plugins/org.sintef.mofscript.editor/repository/metamodels
- Then run a transformations based on your own meta-model
- You will also automatically gain access to a meta-model "aware" content assist

There might be a problem

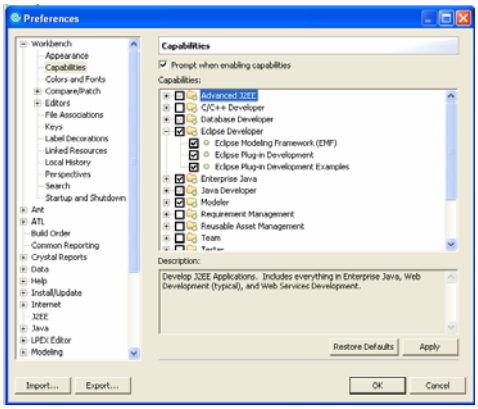
- If the meta-model does not appear after writing “in mdl:” check (window->preferences->MOFScript), and remove the @update before plugin/org.sintef.mofscript.editor/repository…



Creating an ecore model in RSM

- Create a Uml class diagram
- File -> Export -> Ecore model
- This model can be used in the meta-model repository
- The different elements will be accessible in MOFScript content assist

Enabling EMF capabilities in RSM



- Window, Preferences
- Workbench, Capabilities
- Choose Eclipse Developer

Demonstration and examples

Example 1 Uml2Text

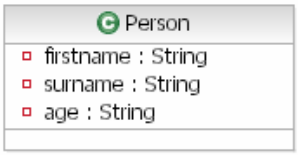
```
texttransformation myTransformation (in uml: http://www.eclipse.org/uml2/1.0.0/UML ){
    uml.Model::main(){
        stdout.println ( "My example“ )
        self.ownedElement->forEach(c:uml.Class){
            c.doClass()
        }
    }
    uml.Class::doClass(){
        <%**********************************\n%>
        stdout.println( "Class name: “ + self.name )

        self.ownedAttribute->forEach(p:uml.Property){
            stdout.println( "\nAttribute: “ +p.name)
            stdout.println( " Visibility: “ +p.visibility
             stdout.println( "Datatype: “ + p.type.name)
        }
        self.ownedOperation->forEach(o:uml.Operation){
              stdout.println( "Operation: “ + o.name)
              o.ownedParameter->forEach(param:uml.Parameter){
                  stdout.println( "Parameter Name: “ + param.name)
                  stdout.println( "Parameter Type: “ + param.type.name )
              }
         }
    } //doClass
} //Transformation
```


Example 2: Uml2Html

Generates a simple html representation of the model
Class and Attributes

Your turn !

- Given a class “Person” with 3 attributes: firstname, surname and age
- Write a transformation rule with context type Class (from UML metamodel)
- The rule shall generate a Java toString method that returns the concatenation of the attributes
  - public String toString{
    ……..



Vision of the future

- If the transformations are made just enough fine-grained and domain independent, it makes them reusable
- Libraries containing hundreds of reusable transformations
- Reuse as it is, by composition, by specializing or opportunistic
- The more domain specific we make the transformation the more code could potentially be generated, but reusability decreases