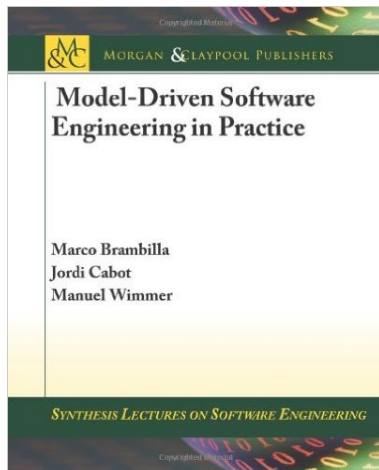


Engenharia de Domínio
Mestrado em Engenharia Informática
PL 8
Model to Text Transformations
Exercise 4: Generating Java Source Code

Alexandre Bragança, Isabel Azevedo
atb@isep.ipp.pt, ifp@isep.ipp.pt

Dep. de Engenharia Informática – ISEP

2017/2018



This lecture is mainly based on the contents of the following book (chapter 8):

"Model-Driven Software Engineering in Practice", Brambilla, Marco; Cabot, Jordi; Wimmer, Morgan & Claypool Publishers, 2012

but also on chapter 6 of [4].

- Model-to-Text Transformations
- Acceleo
- Setup of the Acceleo Tools
- Exercise 4

- Model-to-Text (**M2T**) transformations imply the **derivation of text from models**.
- Model-to-Text transformations have been used for automating several software engineering tasks such as **code-generation** (code representing the system to develop derived from the models, but also other forms of code such as test cases, deployment scripts, etc.), but also **generation of documentation** and task lists, among others. [2].

Ecore-based metamodel and the generated Java code (shown as UML Class Diagram):

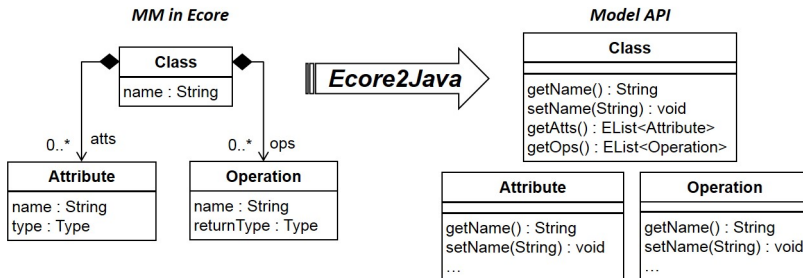


Figure 1: M2T transformation - an example

Phases of code generation:

- Load models
 - Load XML file into memory
- Process models and produce code
 - Process models by traversing the model structure
 - Use model information to produce code
 - Save code into String variable
- Write code
 - Persist String variable to a file using streams

Advantages

- No new languages have to be learned
- No additional tool dependencies

Disadvantages

- Intermingled static/dynamic code
- Non-graspable output structure
- Lack of declarative query language
- Non-graspable output structure

M2T Transformation Languages are template based.

A bunch of template languages for M2T transformation available:

- JET, JET2
- XPAND
- MOFScript
- Acceleo
- XSLT
- ...

Templates are a well-established technique in software engineering.

Many application domains:

- Text processing
- Web Engineering
- ...

E-Mail Text

Dear **Homer Simpson**,
Congratulations! You have won ...

Template Text

Dear «**firstName**» «**lastName**»,
Congratulations! You have won ...

Figure 2: Using templates - an example

The main components of a template-based approach are:

- **Templates**
 - Text fragments and embedded meta-markers
- **Meta-Markers query an additional data source**
 - Have to be interpreted and evaluated in contrast to normal text fragments
 - Declarative model query: Query languages (OCL, XPath, SQL)
 - Imperative model query: Programming languages (Java, C#)
- **Template Engine**
 - Replaces meta-markers with data at runtime and produces output files

■ Template-based Approach at a Glance

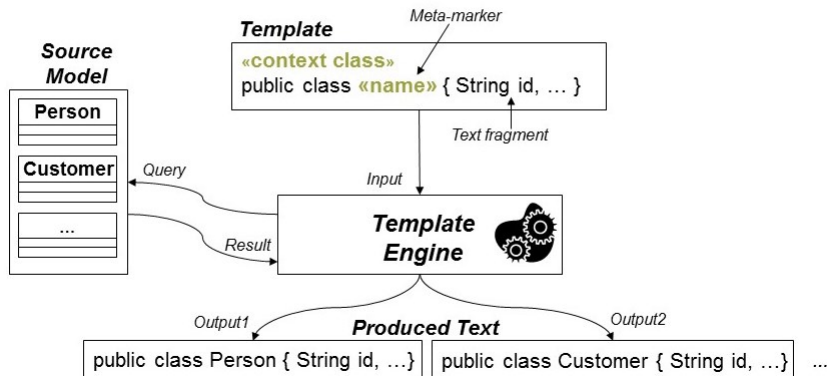


Figure 3: Use of templates by M2T languages

- **Separated static/dynamic code**

- Templates separate static code, i.e., normal text, from dynamic code that is described by meta-markers

- **Explicit output structure**

- Primary structure of the template is the output structure
- Computation logic is embedded in this structure

- **Declarative query language**

- OCL is employed to query the input models

- **Reusable base functionality**

- Support for reading in models, serialize text to files, ...

- **Acceleo is a mature implementation of the OMG M2T transformation standard**
 - Acceleo website: <http://www.eclipse.org/acceleo/>
 - M2T Transformation standard: <http://www.omg.org/spec/MOFM2T>
- **Template-based language**
 - Several meta-markers for useful for code generation available
- **Powerful API supporting**
 - OCL
 - String manipulation functions
 - ...
- **Powerful tooling supporting**
 - Editor, debugger, profiler, traceability between model and code, ...

- **Module concept is provided**
 - Imports the metamodels for the input models
 - Act as container for templates
- **A template is always defined for a particular meta-class**
 - Plus an optional pre-condition to filter instances
 - Templates may call each other
 - Templates may extended each other
 - Templates contain text and provided meta-markers

Templates are the Acceleo structures used to generate code.

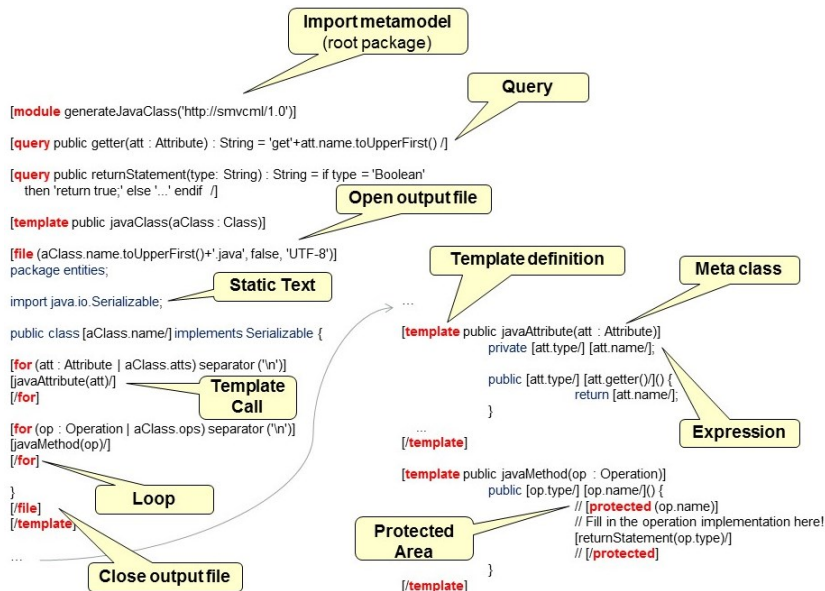
Listing 1: Basic signature of an Acceleo module with one template

```
[module moduleName('http://www.eclipse.org/emf/2002/Ecore')/]
[template public genMyTemplate(aEClass: EClass)]
[/template]
```

Note the visibility specification, the name of the template and its parameters. The parameters are declared following this convention "**name: type**". It is recommended to name them with the following convention "**a<Type>**" (where **Type** is the name of the parameter type). The type of the parameter must be one of the following:

- Types provided by the meta-model (ex: Class, Property, Operation... for UML)
- Default types from OCL like Boolean, String, Integer, OclAny (common super-type of all concepts)

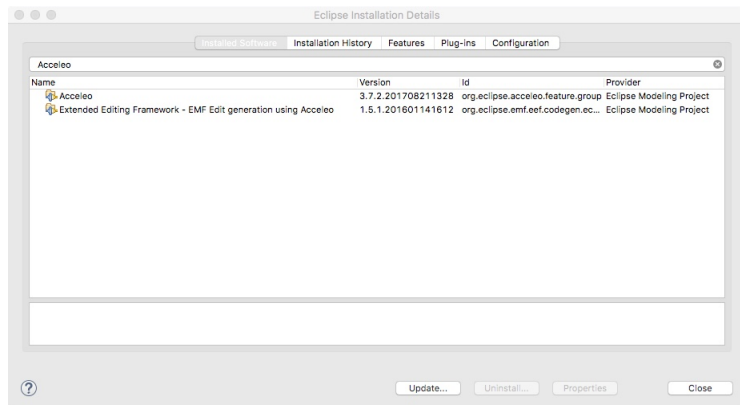
- Several meta-markers (called **tags**) are supported
 - **File** tag: To open and close files in which code is generated
 - **For/If** tag: Control constructs for defining loops and conditions
 - **Query** tag: Reusable helper functions
 - **Expression** tag: Compute values that are embedded in the output
 - **Protected** tag: Define areas that are not overridden by future generation runs



Setup of The Acceleo Tools

Acceleo installation

In Eclipse select "About" and then "Installation Details" and check if the Acceleo plugin is installed. If it is not installed, click on "Help"/"Eclipse Marketplace..." and then search and install **Acceleo**.



How to Create an Acceleo Project (1/8)

- To create an ATL project just select "File"/"New" and then select "Acceleo Project". Fill the fields of the wizard as presented in the Figure. **Do not forget to select the UML metamodel!**

Create a new Acceleo generation project

Create a new Acceleo generation file

This wizard creates new mtl files which can be opened in the Acceleo editor.

Module Files

generate

Parent Folder: org.eclipse.acceleo.module.sample Browse...

Module Name: generate

Metamodel URIs: http://www.eclipse.org/uml2/5.0.0/UML

Template Name: generateElement

Type: Class

☐ Template ☐ Query

☒ Generate documentation

☒ Generate file

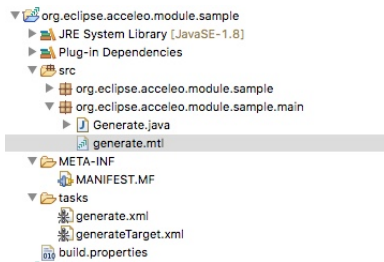
☒ Main template

☐ Initialize Contents

File Browse...

How to Create an Acceleo Project (2/8)

- Your new project should have the following structure.
- The **mtl** file is the most important. It contains the M2T (model-to-text) transformation using the Acceleo syntax.
- The **tasks** folder includes ant build files that can be used to execute our transformations (inside or outside Eclipse).
- The java code is for the plugin project and for running the transformation but we shouldn't need to worry about that for the moment.



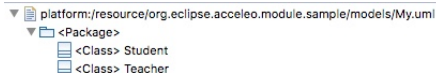
How to Create an Acceleo Project (3/8)

- The **mtl** file should contain a very simple template, like the one presented in the Figure.
- Note line 5 that references the UML metamodel, since in the wizard we selected this metamodel as the source of the transformation.
- The file only includes a template that generates a simple file for each **Class** element that exists in the source model.
- We need a source model (i.e., a UML model) to be able to run this Acceleo transformation.

```
1  [comment encoding = UTF-8 /]
2  /**
3   * The documentation of the module generate.
4   */
5  [module generate('http://www.eclipse.org/uml2/5.0.0/UML')]
6
7
8  /**
9   * The documentation of the template generateElement.
10   * @param aClass
11   */
12  [template public generateElement(aClass : Class)]
13  [comment @main/]
14  [file (aClass.name, false, 'UTF-8')]
15
16  [/file]
17  [/template]
18
```

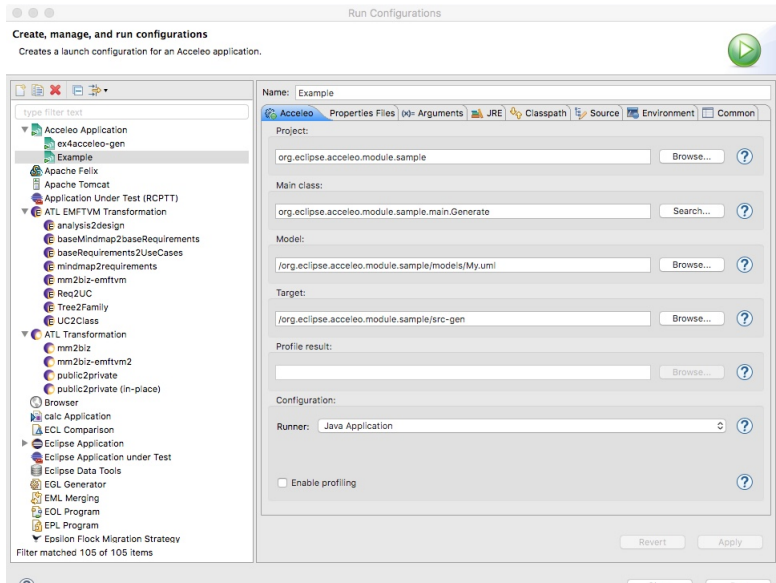
To create a UML source model:

- Create a new folder in the project and call it **models**
- Select "File/New" and then "UML Model". Call it "My.uml".
- Select as "Model Object": Package.
- Add some Class elements to the model (such as the ones presented in the Figure).



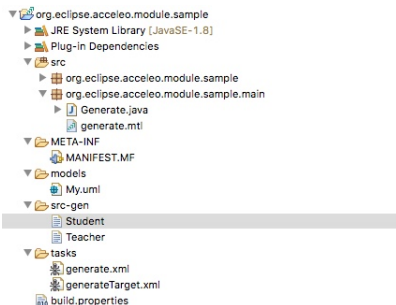
How to Create an Acceleo Project (5/8)

- We can now use the model in a new Acceleo Run Configuration.



How to Create an Acceleo Project (6/8)

- The project should now have some new files...



How to Create an Acceleo Project (7/8)

- We can also use the ant files to execute the transformation.
- In the file **generateTarget.xml** you need to update the properties **MODEL** and **TARGET** to reflect your configuration options.

```
<?xml version="1.0" encoding="UTF-8"?>
1
2
3 <!-- Place this file in the target project and call it with "External Tools > Run As > Ant Build" -
4 <!-- You have to change the MODEL and the TARGET values -->
5
6 <project basedir="." default="generateSample" name="org.eclipse.acceleo.module.sampleSample">
7   <import file=".../org.eclipse.acceleo.module.sample/tasks/generate.xml"/>
8
9   <!-- Change the values of MODEL and TARGET to point to your model and output folder -->
10  <property name="MODEL" value="${basedir}/../models/My.uml"/>
11  <property name="TARGET" value="${basedir}/../src-gen"/>
12
13  <target name="generateSample" description="Generate files in 'TARGET'">
14    <antcall target="generate" >
15      <param name="model" value="${MODEL}"/>
16      <param name="target" value="${TARGET}"/>
17    </antcall>
18  </target>
19 </project>
20
```

Interpreting the screenshot: The image shows the Eclipse IDE interface. The main editor window displays the `generateTarget.xml` file. A context menu is open over the file, with the 'Run As' option selected. The bottom console shows the output of the ant build, indicating a successful execution.

```
<terminated> org.eclipse.acceleo.module.sample generateTarget.xml [Ant Build] /Library/Java/JavaVirtualMachine
Buildfile: /workspaces/odsoft/2017/odsoft-edom-workspace-2017/org.eclipse.acceleo.module.s
generateSample:
generate:
BUILD SUCCESSFUL
Total time: 4 seconds
```

How to Create an Acceleo Project (8/8)

- For more information on Acceleo and the language please refer to the Acceleo Documentation in the Help of Eclipse.

Search: Scope: All topics

Contents

- Workbench User Guide
- Java development user guide
- Platform Plug-in Developer Guide
- JDT Plug-In Developer Guide
- Plug-in Development Environment Guide
- Acceleo Documentation**
 - Overview
 - Getting Started
 - Features
 - Reference
 - Language
 - Acceleo Project
 - Operations
 - Stand Alone
 - Migration
 - Text Production Rules
 - Online Resources
 - What's new
 - Legal
- ATL Guide
- CDO Model Repository Documentation
- Dall Java Persistence Tools User Guide
- EclEmma Java Code Coverage
- Eclipse Marketplace User Guide
- EcoreTools User Manual
- EGF Eclipse Generation Factories Guide
- EGit Documentation
- EGradle
- EMF Compare Documentation
- EMF Developer Guide
- EMF Diff/Merge Guide
- EMF Facet Customization Editor
- EMF Facet Documentation
- EMF Parsley Guide
- EMFText User Guide
- Epsilon User Guide
- Extended Editing Framework Guide
- GEF Developer Guide
- GMF Developer Guide
- Graphiti Developer Guide
- JavaScript Development Guide
- JavaServer Faces Tooling User Guide
- JAX-WS Tools User Guide

Acceleo

Code generation based on the OMG's MOF2T standard

[View on Eclipse.org](#) [Eclipse Marketplace](#)

[Install instructions](#) [What's new](#)

Generate anything from any EMF based models

Open Source

Acceleo is an open source code generator developed inside of the Eclipse Foundation. As such, you can use it freely, fork it and even contribute back.

Integrated in Eclipse

Acceleo is deeply integrated in the Eclipse IDE with a full fledged editor with syntax highlighting, real time error detection, quick fixes, refactoring and much more. It also comes with dedicated views to help you navigate within your code generator and quickly access code generation design patterns.

Incremental Generation

Whether you are considering it or not, you will one day have to modify manually your generated code and you want to keep your modification even if you are regenerating your code. Acceleo lets you define protected areas in which you can safely modify the generated.

Stand Alone

Acceleo does not lock you inside of the Eclipse environment, as such you can build and run your generator easily out of Eclipse. Acceleo provides a brand new Maven integration.

Versatile

Code generators are often limited to a set of technologies. With its template based approach, Acceleo can generate code for any kind of languages. If you can write it, Acceleo can generate it.

Jump in with [the tutorial](#) and [the list of all the features](#).

Exercise 4

Context for this Exercise:

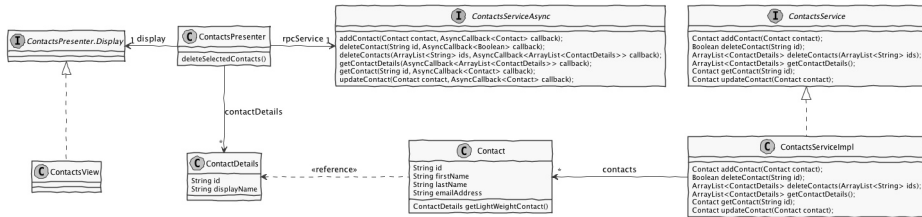
- In this exercise we will continue from where we left in exercise 3.
- In exercise 3 we end up with a UML Use Case model that reflected the functional requirements of our applications.

In exercise 4 we will:

- Complement the use case model with more information. We will call it **analysis model**.
- Generate Java code for implementing the functional requirements by using two approaches:
 - 1 Generate a new **design model** from the analysis model and then generate the implementation from this new design model.
 - First an ATL transformation (UML analysis to UML design)
 - Then an Aceleo transformation (UML design to Java Code)
 - 2 Generate the implementation (i.e., java code) directly from the analysis model.
 - This will use an Aceleo transformation (UML to Java Code).
- The target will be new functionality in the GWT sample application: College Management System.

Exercise 4: Goal

Whatever the approach of the previous slide the goal is to have generated Java code similar to the one presented in the Figure for **each of the CRUD Use Cases**.



The Context:

- From exercise 3, the final transformation, produces a Use Case Model.
- This Use Case Model specifies the functionality of an application. For instance, we can have a use case called "Manage Contact", that represents the CRUD¹ operations over the entity "Contact".

The Problems:

- Although this is the **intention** of the model, there is nothing that unambiguously states that and we also do not know the details of the "Contact" entity. Therefore, we need to:
 - State that some Use Cases represent CRUD functionality;
 - Specify the details of entities.

The Solution:

- **Problem:** State that some Use Cases represent CRUD functionality.
 - **Solution:** Use annotations to mark such use cases.
- **Problem:** Specify the details of entities.
 - **Solution:** Use a domain model to model entities.

¹See https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

Notice:

- This is an academic exercise!
- We do not state that all use cases represent CRUD functionality.
- The presented approaches are only to illustrate the used technologies and the MDE approach.
- Other approaches could have been taken.

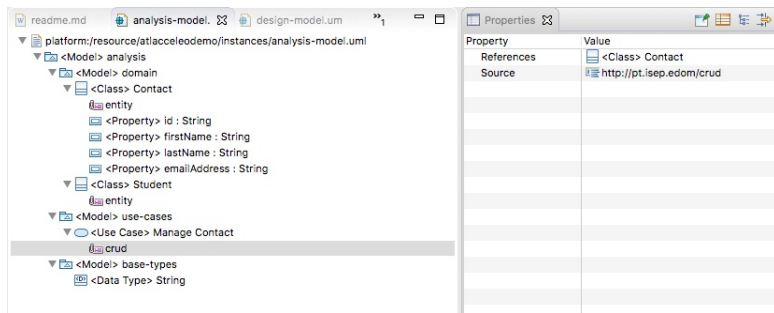
Exercise 4: Startup Sample Project

- For this exercise there is a sample project that you should use as a starting point.
- Please, use the following url to get the project:
`https://bitbucket.org/mei-isep/atlaceleodemo`
- This is an Eclipse project that you can import into Eclipse.
- This project contains a partial solution for the overall exercise.

Exercise 4: Startup Sample Project / Analysis Model

In this figure we illustrate an analysis model that should be the starting point for this exercise. It includes:

- **use-cases** This is the use case model that resulted from the previous exercise.
- **domain** This is a domain model that should contain the specification of the entities of the application that are "referenced" by the use cases. This domain model must be edited manually!
- **base-types** This include some base types that are required for the domain model.



Exercise 4: Startup Sample Project / Analysis Model

Use cases that represent CRUD operations should be annotated. The annotation should:

- Have as **Source**: <http://pt.isep.edom/crud>
- Have as **References**: the entity from the domain model that is the focus of the CRUD operations of the use case

The screenshot shows the Eclipse IDE with the 'analysis-model.uml' file open. The left-hand 'Project Explorer' view displays a hierarchical tree of the model's contents. The tree structure is as follows:

- platform:/resource/atilaccedemo/instances/analysis-model.uml
 - <Model> analysis
 - <Model> domain
 - <Class> Contact
 - entity
 - id : String
 - firstName : String
 - lastName : String
 - emailAddress : String
 - <Class> Student
 - entity
 - <Model> use-cases
 - <Use Case> Manage Contact
 - crud
 - <Model> base-types
 - String

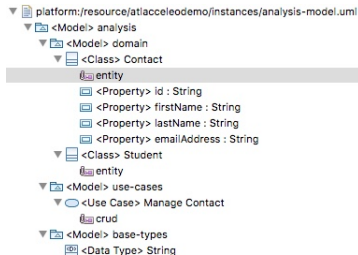
The 'crud' element under the 'Manage Contact' use case is currently selected. On the right-hand side, the 'Properties' view is open, showing a table with two columns: 'Property' and 'Value'. The table contains the following entries:

Property	Value
References	<Class> Contact
Source	http://pt.isep.edom/crud

Exercise 4: Startup Sample Project / Analysis Model

Elements of the Domain Model that represent "Entities" should be annotated. The annotation should:

- Have as **Source**: <http://pt.isep.edom/entity>



Property	Value
References	
Source	http://pt.isep.edom/entity

Exercise 4: Startup Sample Project / Analysis Model to Design Model

In the project you will find a very incomplete ATL transformation that generates a design model from the analysis model described previously. Right click on the file "analysis2design2.launch" and select "Run As".

```
1 |:= @atlcompiler emftvm
2
3 -- @path Analysis=http://www.eclipse.org/uml2/5.0.0/UML
4 -- @path Design=http://www.eclipse.org/uml2/5.0.0/UML
5
6 module Analysis2Design;
7 create OUT : Design from IN : Analysis;
8
9 @helper context Analysis!Model def : referencedInCRUDUseCase() : Set( Analysis!Class ) = Analysis!Class.allInstances()->select( c |
10 not c.eAnnotations->select(a | a.source = 'http://pt.isep.edom/entity')->asSet()->isEmpty() -- must be annotated as entity
11 and Analysis!UseCase.allInstances()->select( uc |
12 uc.eAnnotations->select( anot |
13 anot.source = 'http://pt.isep.edom/crud' and anot.references->includes(c)
14 )->asSet()->notEmpty()
15 )->asSet()->notEmpty() )->asSet();
16
17 @rule Model2Model {
18 from
19 sm: Analysis!Model ( sm.name = 'domain' )
20 to
21 tm: Design!Model (
22 name <- 'design-model'
23 , packagedElement <- sm.referencedInCRUDUseCase()
24 )
25 }
26
27 @rule Entity2Class {
28 from
29 sm : Analysis!Class (
30 not sm.eAnnotations->select(a | a.source = 'http://pt.isep.edom/entity')->asSet()->isEmpty() -- must be annotated as entity
31 and Analysis!UseCase.allInstances()->select(
32 uc | uc.eAnnotations->select(
33 a | a.source = 'http://pt.isep.edom/crud' and a.references->includes(sm)
34 )->asSet()->notEmpty()
35 )->asSet()->notEmpty() -- must be referenced by a CRUD use case
36 )
37 to
38 tm : Design!Class (
39 name <- sm.name
40 ),
41 tmDetails : Design!Class (
42 name <- sm.name + 'Details'
43 ),
44 tmPresenter : Design!Class (
45 name <- sm.name + 'SPresenter'
46 )
47 }
48 }
```

Exercise 4: Startup Sample Project / Design Model to Java Code

In the project you will find a very incomplete Aceleo transformation that generates java code from the previous design model. You can use "tasks-nogen/design2code.xml" to run this Aceleo transformation with Ant. You can also create a Run Configuration as described in slide 24.

```
1 [comment encoding = UTF-8 /]
2 [module generate('http://www.eclipse.org/uml2/5.0.0/UML')]
3
4 [template public generateElement(aClass : Class)]
5 [comment @main /]
6 [file (aClass.name+'.java', false, 'UTF-8')]
7 // This is a test
8 public class [aClass.name/] {
9
10     [for (p: Property | aClass.attribute) separator('\n')]
11     private [p.type.name/] [p.name/];
12     [/for]
13
14     [for (o: Operation | aClass.ownedOperation) separator('\n')]
15     public [if (o.type->isEmpty())void[else] [o.type.name/][if] [o.name/]] {
16         // [protected(o.name)]
17         // TODO should be implemented
18         // [/protected]
19     }
20     [/for]
21 }
22 [/file]
23 [/template]
```

Exercise 4: Requirements

- Generate Java code for implementing the functional requirements by using two approaches:
 - ① **Base Solution:** Generate a new **design model** from the analysis model and then generate the implementation (i.e., **java code**) from this new design model.
 - First an ATL transformation (UML analysis to UML design)
 - Then an Aceleo transformation (UML design to Java Code)
 - ② **Alternative Solution:** Generate the implementation (i.e., **java code**) directly from the analysis model (i.e., do not generate an intermediate design model).
 - This will use an Aceleo transformation (UML to Java Code).
- The target will be new functionality in the GWT sample application: College Management System. Both solutions should try to generate code **as much as possible similar** to the one in the GWT CMS application and **in accordance with the diagram from slide 30**. It is not required to generate complete functional code but the code should be correct.
- **Review** Your review should focus on how both solutions compare.



Eclipse community

Acceleo/Getting Started.

[https://wiki.eclipse.org/Acceleo/Getting_Started.](https://wiki.eclipse.org/Acceleo/Getting_Started)



Brambilla, Marco; Cabot, Jordi; Wimmer, Manuel

Model-Driven Software Engineering in Practice.

2012,

Morgan & Claypool Publishers.



Eclipse community

MOF Model to Text Transformation Language, v1.0.

2006,

[http://www.omg.org/spec/MOFM2T/1.0/PDF/.](http://www.omg.org/spec/MOFM2T/1.0/PDF/)



Gronback, Richard C.

Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.

2009,

Addison-Wesley Professional.