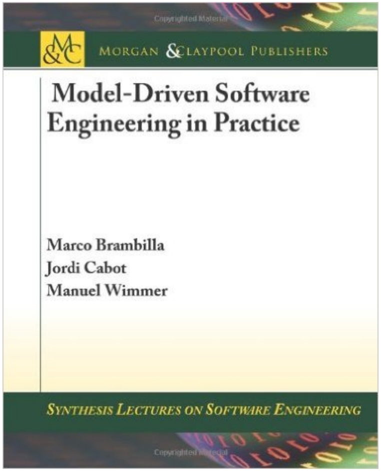
Engenharia de Domínio
Mestrado em Engenharia Informática
PL 6
Model to model transformations
Exercise 3: Mindmap to Requirements

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

Dep. de Engenharia Informática – ISEP

2017/2018

Acknowledgement


Copyrighted Material

MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

Copyrighted Material

These slides are 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 [3].

Outline

- Transformations in MDSE
- ATL: Atlas Transformation Language
- Setup of the ATL Tools
- Exercise 3

Transformations

"Besides models, model transformations represent the other crucial ingredient of MDSE and allow the definition of mappings between different models." [1]

The ability to perform model transformations is one of the advantages of modeling over simply drawing. Models are specified according to a modeling language, in turn defined according to a metamodeling language. The transformation executions are defined based on transformation rules using a specific transformation language.

In MDSE context: **Models + Transformations = Software**

Model-to-model and model-to-text transformations

In model driven development software is derived through a series of **model-to-model transformations** (possibly) ending with a **model-to-text transformation** that produces the final code.

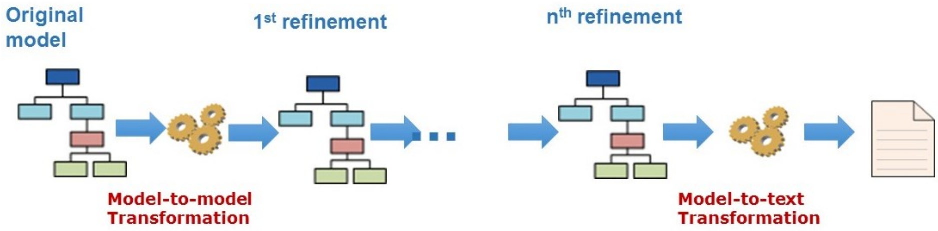
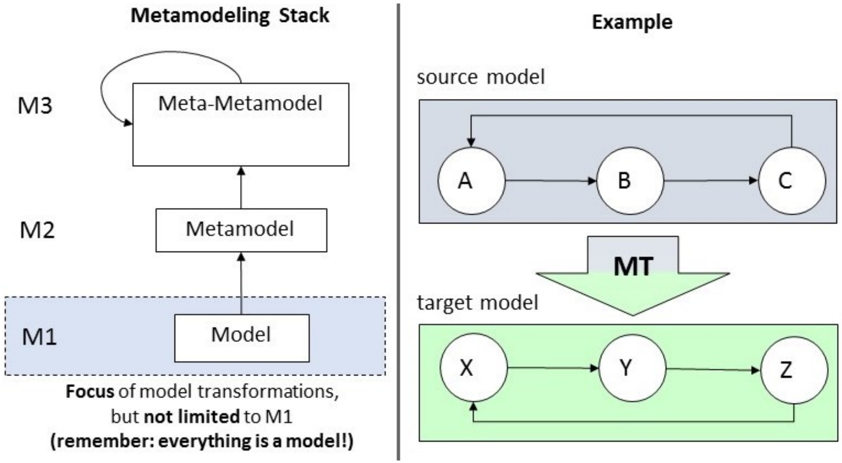


Figure 1: Transformations [1]

Definitions

A **model-to-model (M2M) transformation** is the automatic creation of target models from source models.



Metamodeling Stack
M3
Meta-Metamodel
M2
Metamodel
M1
Model
Focus of model transformations,
but **not limited** to M1
(remember: everything is a model!)
Example
source model
A
B
C
MT
target model
X
Y
Z

Figure 2: Model-to-model (M2M) transformation [1]

Models

“Everything is a model”.

Metamodels, metametamodels, and even transformations are models. Transformations are models of operations upon models.

Execution paradigms

There two main execution paradigms for model transformations:

- **Out-place Transformations** - a new model is built from scratch
- **In-place Transformations** - a model is changed by creating, deleting, and updating elements in the input model

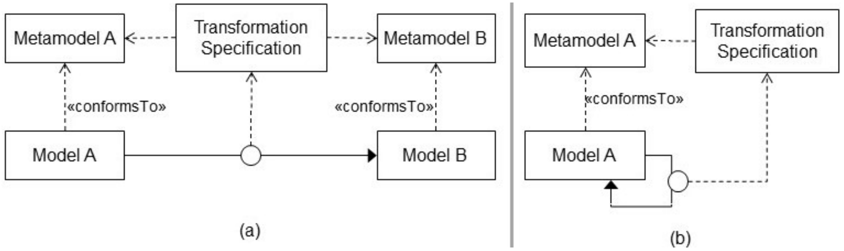


Figure 3: Different kinds of model transformations: (a) out-place vs. (b) in-place [1]

Out-place transformation: a general example

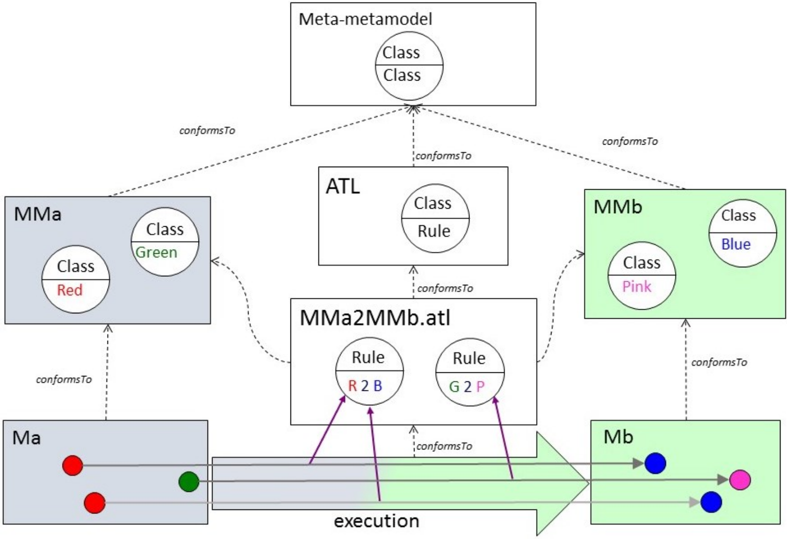


Figure 4: Architecture - an illustrative example [1]

Out-place transformation: a particular example

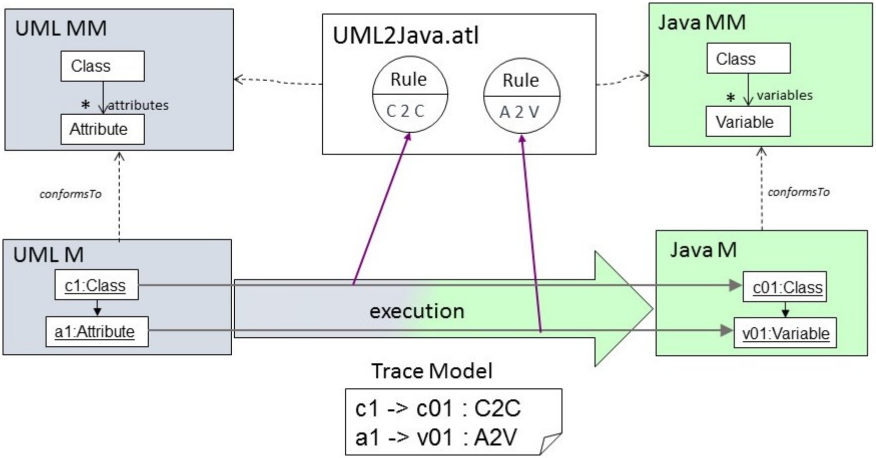


Figure 5: Architecture - a concrete example [1]

ATLAS transformation Language

ATLAS transformation Language (ATL) is a rule-based transformation language for developing out-place transformations which is based on OCL, but with extra language features. Additionaly, it is:

- Widely used in academia and industry
- Supported by a mature tool

ATL overview

- Source models are read-only and target models are write-only
- The language is a declarative-imperative hybrid. The declarative part uses:
  - **Matched rules** with automatic traceability support
  - Side-effect free query language: **OCL**
- The imperative part uses:
  - Called/Lazy rules
  - Action blocks
  - Global variables via Helpers
- Recommended programming style: **declarative**

Rules

- A **declarative rule** specifies
  - A source pattern to be matched in the source models
  - A target pattern to be created in the target models for each match during rule application
  - An optional action block (i.e., a sequence of imperative statements)

- An **imperative rule** is basically a procedure
  - It is called by its name
  - It may take arguments
  - It contains a declarative target pattern and an optional action block

Applying a rule means: **(1)** creating the specified target elements, and **(2)** initializing the properties of the newly created elements.

Types of rules

There are two types of rules concerning their application

- **Matched rules** are applied once for each match by the execution engine (a given set of elements may only be matched by one matched rule)
- **Called/Lazy rules** are applied as many times as they are called from other rules

Matched rules

A Matched rule is composed of

- A source pattern
- A target pattern
- An optional action block

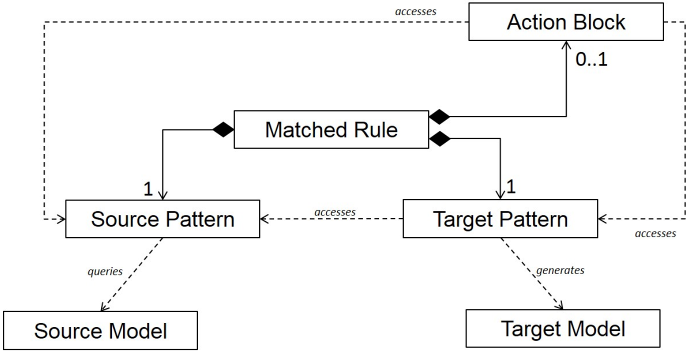


Figure 6: Matched rules - overview

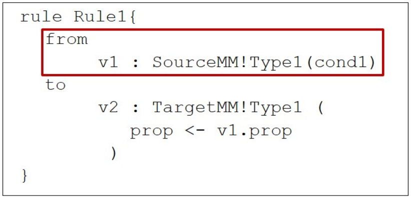
Matched rules: source pattern

The source pattern is composed of

- A set of labeled source pattern elements
- A source pattern element refers to a type contained in the source metamodels
- A guard (Boolean OCL expression) used to filter matches

A match corresponds to a set of elements coming from the source models that

- Fulfill the types specified by the source pattern elements
- Satisfy the guard



```
rule Rule1{
   from
         v1  : SourceMM!Type1(cond1)
   to
         v2  : TargetMM!Type1 (
            prop <- v1.prop
           )
}
```

Figure 7: Source pattern

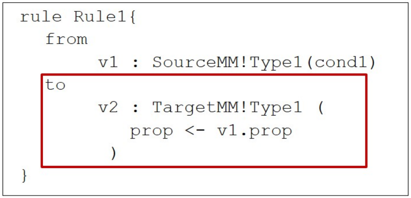
Matched rules: target pattern

The target pattern is composed of

- A set of labeled target pattern elements
- Each target pattern element refers to a type in the target metamodels contains a set of bindings
- A binding initializes a property of a target element using an OCL expression

For each match, the target pattern is applied

- Elements are created in the target models
- Target elements are initialized by executing the bindings



```
rule Rule1{
  from
       v1 : SourceMM!Type1(cond1)
  to
       v2 : TargetMM!Type1 (
          prop <- v1.prop
         )
}
```

Figure 8: Target pattern

Helpers

Syntax

```
helper context Type def : Name(Par1 : Type, ...) : Type = EXP;
```

Global Variable - example

```
helper def: id : Integer = 0;
```

Global Operation - example

```
helper context Integer def : inc() : Integer = self + 1;
```

Calling a Helper

- `thisModule.HelperName(...)` – for global variables/operations without context
- `value.HelperName(...)` – for global variables/operations with context

Example 1 – Publication2Book

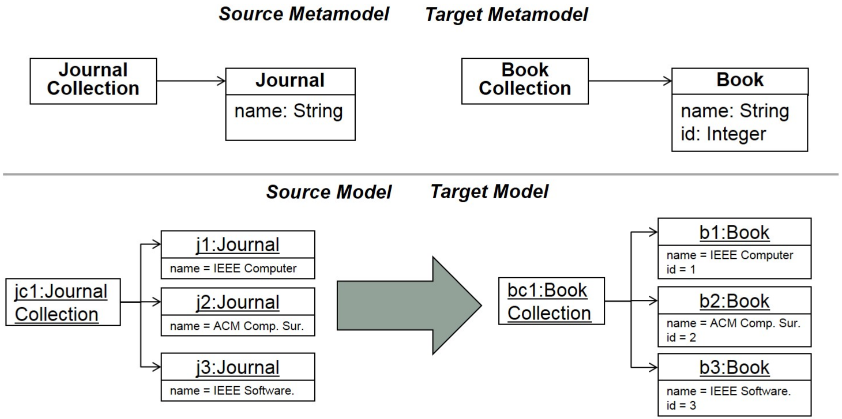
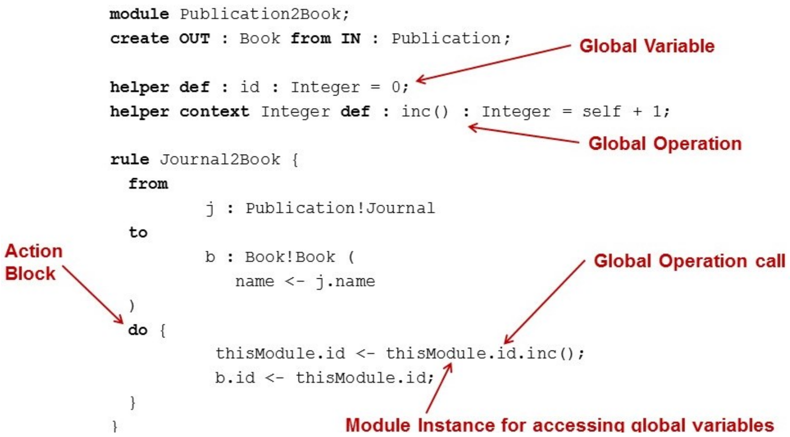


Figure 9: Publication2Book

Example 1 – Publication2Book (ATL)



```
module Publication2Book;
create OUT : Book from IN : Publication;

helper def : id : Integer = 0;
helper context Integer def : inc() : Integer = self + 1;

rule Journal2Book {
  from
          j : Publication!Journal
  to
          b : Book!Book (
             name <- j.name
  )
  do {
           thisModule.id <- thisModule.id.inc();
           b.id <- thisModule.id;
  }
}
```

Example 2 – Person2Customer

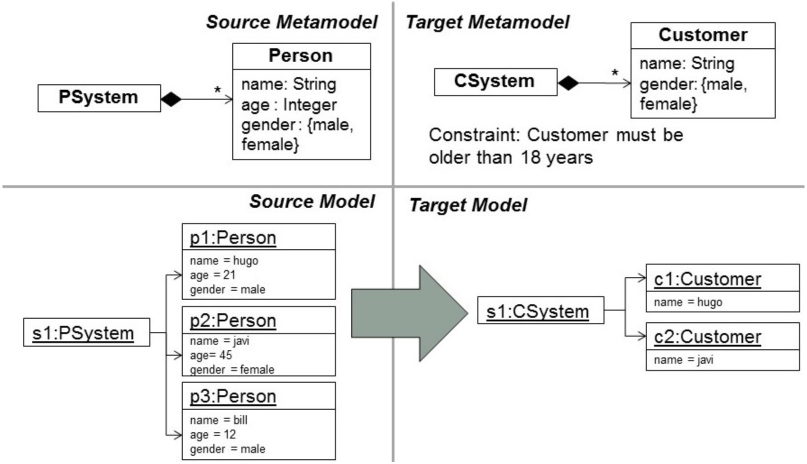
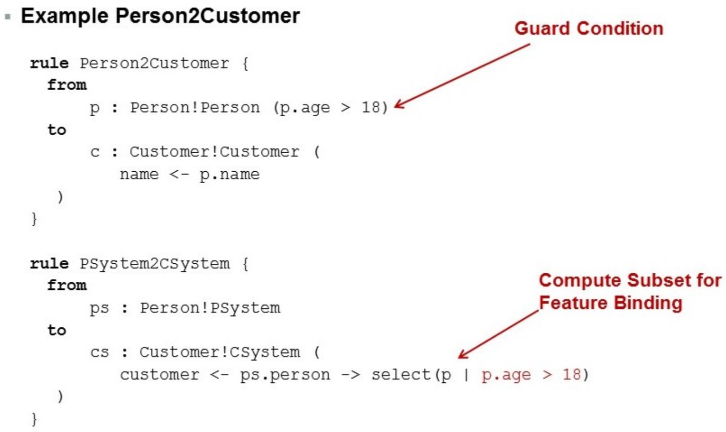


Figure 10: Person2Customer

Example 2 – Person2Customer (ATL)



- **Example Person2Customer**

```
rule Person2Customer {
  from
       p : Person!Person (p.age > 18)
  to
       c : Customer!Customer (
          name <- p.name
    )
}

rule PSystem2CSystem {
  from
       ps : Person!PSystem
  to
       cs : Customer!CSystem (
          customer <- ps.person -> select(p | p.age > 18)
    )
}
```

Guard Condition

Compute Subset for Feature Binding

Example 3

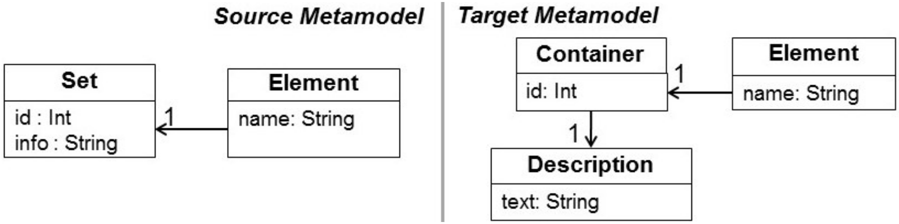
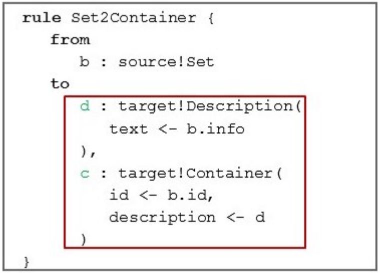
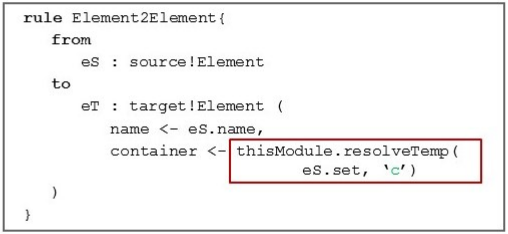


Figure 11: Models overview

Example 3 (ATL)



```
rule Set2Container {
   from
      b : source!Set
   to
      d : target!Description(
         text <- b.info
      ),
      c : target!Container(
         id <- b.id,
         description <- d
      )
}
```



```
rule Element2Element{
   from
      eS : source!Element
   to
      eT : target!Element (
         name <- eS.name,
         container <- thisModule.resolveTemp(
                            eS.set, 'c')
   )
}
```

The `resolveTemp` operation is able to retrieve produced target elements created by a different rule for a given source element.

Transformation execution phases

Module Initialization Phase

- Module variables (attribute helpers) and trace model are initialized
- If an entry point called rule is defined, it is executed in this step

Matching Phase

- Using the source patterns (from) of matched rules, elements are selected in the source model (each match has to be unique)
- Via the target patterns (to) corresponding elements are created in the target model (for each match there are as much target elements created as target patterns are used)
- Traceability information is stored

Target Initialization Phase

- The elements in the target model are initialized based on the bindings (<-)
- The resolveTemp function is evaluated, based on the traceability information
- Imperative code (do) is executed, including possible calls to called rules

Enumerations in ATL implementation

An enumeration is an OclType and has a name. With the OCL specification, referring to an enumeration literal is achieved by specifying the enumeration type (e.g. the name of the enumeration), followed by two double-points and the enumeration value. Consider, as an example, an enumeration named Gender that defines two possible values, male and female. Accessing to the female value of this enumeration type in OCL is achieved as follows: `Gender::female`.

The current ATL implementation differs from the OCL specification in the way an enumeration literal is mentioned. Its name is preceeded by a sharp character without using the enumeration type: `#female`.

ATL Keywords

It is possible to distinguish three kinds of ATL reserved keywords:

- **Constant keywords**: true, false
- **Type keywords**: Bag, Set, OrderedSet, Sequence, Tuple, Integer, Real, Boolean, String, TupleType, **Map**
- **Language keywords**: not, and, or, xor, implies, module, create, from, uses, helper, def, context, rule, using, derived, to, mapsTo, distinct, foreach, in, do, if, then, else, endif, let, library, query, for, div, refining, entrypoint

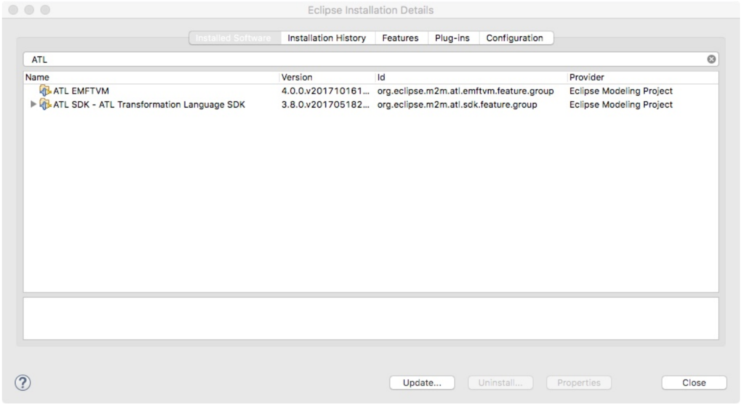
The use of the string "main", which does not belong to the set of language keywords, is restricted.

If required to be used in a different context that their initial one, the keywords can be escaped thanks to double quotes to be considered as any other strings.

Setup of The ATL Tools

ATL installation

In Eclipse select "About" and then "Installation Details" and check if the ATL SDK and the ATL EMFTVM plugins are installed. If they are not installed, click on "Help"/"Install New Software..." and then search and install ATL SDK and ATL EMFTVM.



Name	Version	Id	Provider
ATL EMFTVM	4.0.0.v201710161...	org.eclipse.m2m.atl.emftvm.feature.group	Eclipse Modeling Project
ATL SDK - ATL Transformation Language SDK	3.8.0.v201705182...	org.eclipse.m2m.atl.sdk.feature.group	Eclipse Modeling Project

How to Create an ATL Project

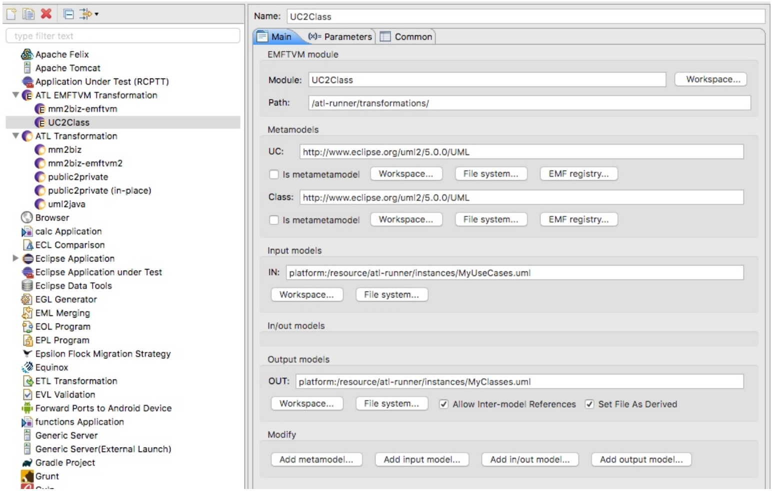
- To create an ATL project just select "File"/"New" and then select "ATL Project". This will create an empty ATL Project.
- You should select the "ATL" perspective when working with ATL Projects.
- Inside this new project create folder for holding the different type of files to be used. For example: metamodels, trasnsformations and instances.
- You can use the project `https://bitbucket.org/mei-isep/atl-runner` as an example on how to organize your folders.

How to Create an ATL Transformation

- To create an ATL transformation just select "File"/"New" and then select "ATL File". Give the file a proper name. This will create an empty ATL file.
- You can see examples of ATL files in `https://bitbucket.org/mei-isep/atl-runner`.
- Do not forget to include the following line at the top of the atl file:
    - `-- @atlcompiler emftvm`

How to Run or Debug an ATL Transformation inside Eclipse

- You need to create a Run Configuration of type "ATL EMFTVM Transformation".
- The following example can be used to Run/Debug the ATL file "UC2Class.atl" included in the project `https://bitbucket.org/mei-isep/atl-runner`.



Name: UC2Class

EMFTVM module
Module: UC2Class
Path: /atl-runner/transformations/

Metamodels
UC: http://www.eclipse.org/uml2/5.0.0/UML
☐ Is metametamodel
Class: http://www.eclipse.org/uml2/5.0.0/UML
☐ Is metametamodel

Input models
IN: platform:/resource/atl-runner/instances/MyUseCases.uml

In/out models

Output models
OUT: platform:/resource/atl-runner/instances/MyClasses.uml
✓ Allow Inter-model References ✓ Set File As Derived

Modify

How to Run an ATL Transformation OUTSIDE Eclipse

- Please follow the instructions in readme.md file of the project `https://bitbucket.org/mei-isep/atl-runner` to run transformations from the command line or using ant.
- Please be aware that in order to to so its necessray a special jar that is located in the "dist" folder of the project.
- Also, be aware that this method is only able to run transformations that do not depend on registered ecore metamodels. That is to say that it is possible to use the method with our own metamodels but not with metamodels that we do not have direct access to their respective ecore files (for instance, the UML2 metamodel).

Exercise 3

Exercise 3

Exercise 3: Domain Engineering Goals

Goals Regarding **Domain Engineering:**

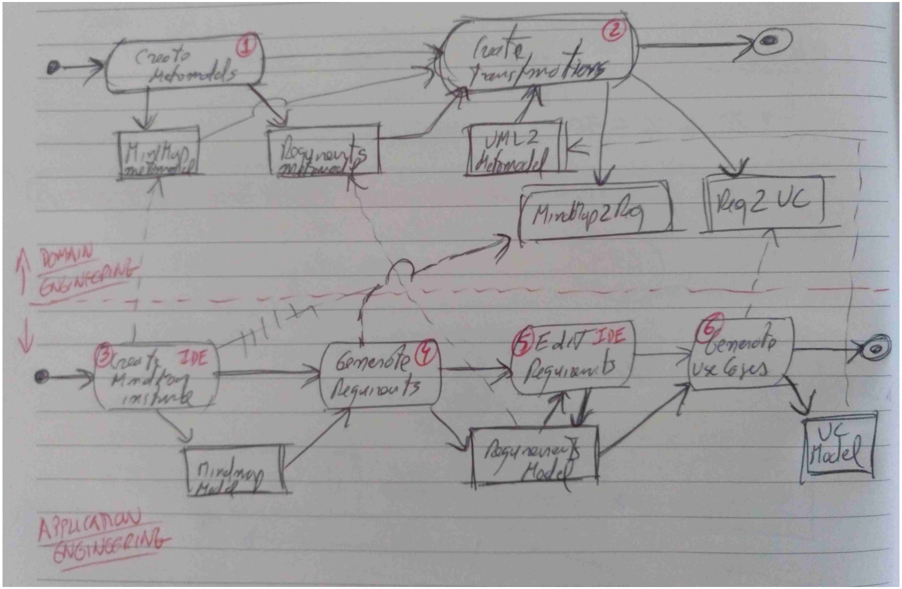
1. **Create Metamodels.** Create the Mindmap and Requirements metamodels (or reuse the ones created in the previous exercise)
2. **Create Transformations.**
   - Create an ATL transformation (ie., an ATL file) to transform instances of mindmap in instances of requirements (i.e., Mindmap2Req.atl).
   - Create an ATL transformation (ie., an ATL file) to create use cases (i..e, UML use cases) from functional requirements in instances of requirements models (i.e., Req2UC). Use the UML2 metamodel for the use cases.

Exercise 3: Application Engineering Goals

Goals Regarding **Application Engineering:**

3. Create a Mindmap model (i.e, instance) as described in the previous exercise (or reuse the one created in the previous exercise).
4. Using the Mindmap2Req transformation generate a requirements model.
5. Edit the previous generated requirements model. Add new functional requirements:
    - Under each RequirementGroup (except the top level "College") add a new Requirement with title "Manage <name of RequirementGroup>"
    - The description should be "CRUD operations for <name of RequirementGroup>"
    - The type of this requirement should be "FUNCTIONAL"
    - All these requirement should relate to a Version "1.0.0"; State should be "NEW"; Priority should be "MEDIUM"; and resolution should be "LATER"
    - For ID you should use a pattern link "R<int>", where <int> is an incrementing integer
6. Using the Req2UC transformation generate a use case model.

Exercise 3: Draft of the Overall Process



Exercise 3: Notes about the Review/Analysis

- Solutions should differ in the way they implement the transformations (**Goal 2**).
  - These differences must be analysed in your review of the exercise.
- **Each review** must also **include a proposal on how to solve the following problem**:
  - There is a manual step (step 5) between the transformations 4 and 6, where the modeler will change/update the model that is produced by step 4.
  - The next time step 4 is executed it will rebuild from scratch the requirements model and, by doing this, **the manual updates of step 5 will be lost**.
  - **Is there a solution to keep the overall process and avoid loosing manual editions in the requirements model?**
- **In each review** compare the solution to the previous problem with the other solution of the team.

Exercise 3: Notes about Step 2
Mindmap to Requirements Transformation - Overview

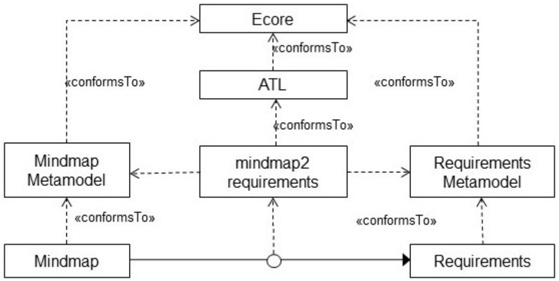


Figure 12: A M2M transformation - overview

Exercise 3: Notes about Step 2
Mindmap to Requirements Transformation - Requirements

The following three requirements have to be fulfilled by the transformation from Mindmap to Requirements:

- **Requirement 1**: The Map (Mindmap metamodel) is to be used as the input to the mapping that create a Model instance (Requirements metamodel)
- **Requirement 2**: For each root Topic, a RequirementGroup is to be created and its ID should start with "G" followed by a integer number. A counter variable can be used. Examples: G1, G2, G3...
- **Requirement 3**: All other Topic elements are to be transformed into a RequiremenGroup. Its ID should continue the same numeration of the root RequirementGroups. For instance, if the last RequirementGroup was "G4", then the ID should be "G5". The hierarchy of these RequirementGroups should follow be similar to the one of the respective Topics.

Note: The project `https://bitbucket.org/mei-isep/atl-runner` already contains both metamodels (mindmap and requirements), one instance of mindmap and the transformation. There is also an example of how to run the transformation in the Ant file build.xml.

Exercise 3: Notes about Step 2
Mindmap to Requirements Transformation - How to Develop

How to Develop and Execute this transformation

- This transformation should be possible to **execute outside eclipse**
  - Please follow the instructions and see the example provided in the following project: `https://bitbucket.org/mei-isep/atl-runner`

Exercise 3: Notes about Step 2
Requirements to Use Cases Transformation - Overview

- For this transformation the input metamodel is **Requirements**.
- The output metamodel needs to be a **UML metamodel** so that we can generate the Use Cases.
- EMF includes a UML metamodel that is named **UML2**
  - To access this metamodel in eclipse (as well as all other registered metamodels) go to the "Navigate" menu and select "Open EPackage".
  - In the new window type "UML2". Select the entry "http://www.eclipse.org/uml2/5.0.0/UML".
  - A new window will open that will enable you to inspect the metamodel of UML.
- To manually create an instance (i.e., model) of UML in eclipse:
  - Select "File/New..."and the enter "uml". Select "UML Model" and complete the wizard.
  - You should be able to edit an UML model in a way similar to any other EMF model.
- You can also create use case diagrams from the UML model by using papyrus:
  - Right click in the UML model file in Package Explorer and select "New/Other..." and then "Papyrus Model".
  - Create a use case diagram in a similar way to how it was done in exercise 1.
  - If you select the "Papyrus" perspective, you can use the "Model Explorer" to select existing elements in the model and drag them to the use case diagram.

Exercise 3: Notes about Step 2
Requirements to Use Cases Transformation - Requirements

The following requirements have to be fulfilled by the transformation from Requirements to Use Cases:

- **Requirement 1**: The Model from Requirements should originate a Model in UML.
- **Requirement 2**: Every RequirementGroup should originate a Component in UML. The structure/hierarchy of Components should be the same as the one of RequirementGroup.
- **Requirement 3**: Every functional Requirement that is child of a RequirementGroup should originate a owned use case in the corresponding Component.
- **Requirement 4**: Every functional Requirement that is child of another Requirement should originate a owned use case in the corresponding "parent" Component and the Use case corresponding to its Requirement parent should have an include relationship to it.

Exercise 3: Notes about Step 2
Requirements to Use Cases Transformation - How to Develop

How to Develop and Execute this transformation

- This transformation <u>can only be execute inside eclipse</u>
    - In the project `https://bitbucket.org/mei-isep/atl-runner` you can find an example of a transformation that uses UML as input and also as output.

Further Reading

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

Eclipse community
ATL/Tutorials - Create a simple ATL transformation.
June 2013,
https://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation.

Gronback, Richard C.
Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.
2009,
Addison-Wesley Professional.

Wagelaar, Dennis
ATL/User Guide - The ATL Language.
15 March 2015,
http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language.