

Modelling and consistency check of multi-variant AML models with Eclipse

Sven Kägebein, Yibo Wang
inpro, Universität Hamburg

3. AutomationML Plugfest

14. - 15.10.2015

Motivation

- There is a need for the definition of a constraint-based and multi-variant AutomationML-Model
- From the conceptional point of view, how OCL-Constraints can be integrated in AutomationML, different papers in the past propose a solution e.g. [1]
- The main focus today is on a possible tool support for the configuration and finally the creation of an less-variant AutomationML-Model
- All this work and concepts were created in scope of the EfA-Project which ends this month [2]
- The idea is to bring AutomationML into the Eclipse world, to get use of the various plugin environment (e.g. OCL, EMF and pure::variants)
- The two following exercises mainly are focused on using these tools, to show the possibilities, not to explain the concepts or technical facts behind in detail

[1] - Prinz, J.; Kägebein, S., Hundt, L. Dr.: AutomationML for data exchange within a model and requirement driven functional development process ; Automation 2015. – Kongress: „Automation 2015“; 16 (Baden-Baden)

[2] – „Design method for planning of variable automation systems based on model integration“ - <https://www.hs-owl.de/init/forschung/projekte/b/filteroff/209/single.html>

Content of this workshop

- Deployment of a basic toolset
- Exercise I:
 - Convert an AutomationML-File to a for Eclipse-EMF readable format
 - Using Eclipse-OCL-Plugin for OCL-Constraint-Check
- Exercise II/III:
 - Creation of a variant model for the configuration of an AutomationML-File
 - Creation of a less-invariant AutomationML-File based on the configuration
 - Programmatically way
- Not all files are translated in english, but for the tooling point of view perhapse it's not necessary

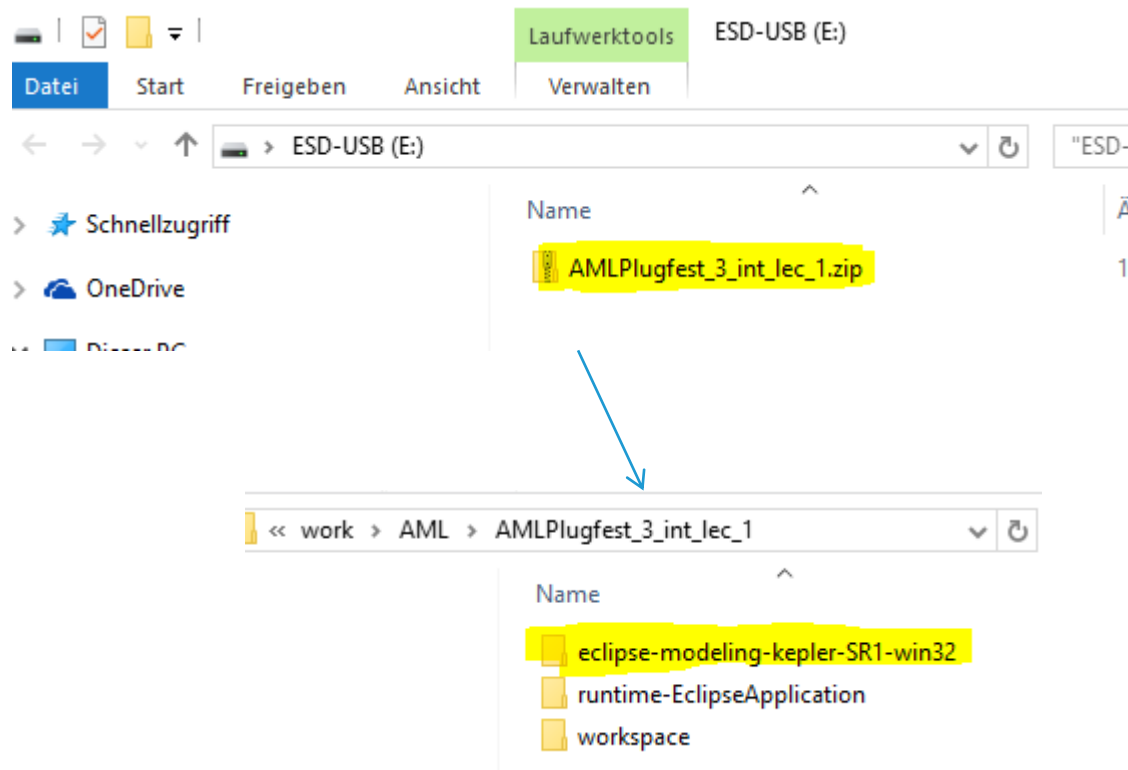




Setup

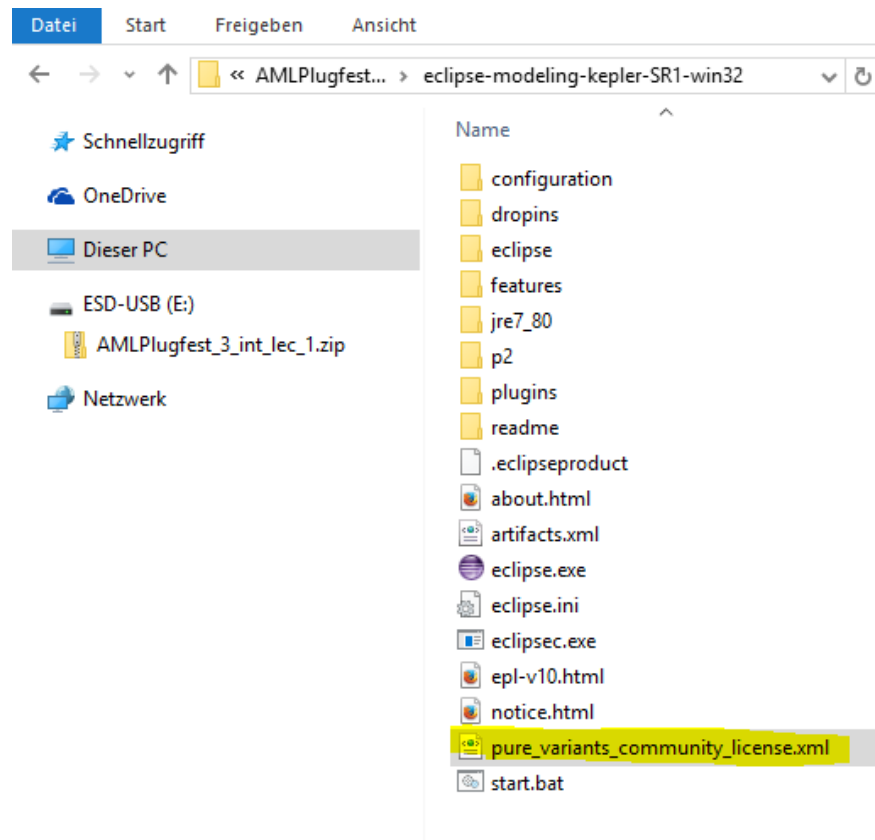
Setup I – Installation (hopefully already done)

- Copy or download zip archive to aim directory
- Extract package

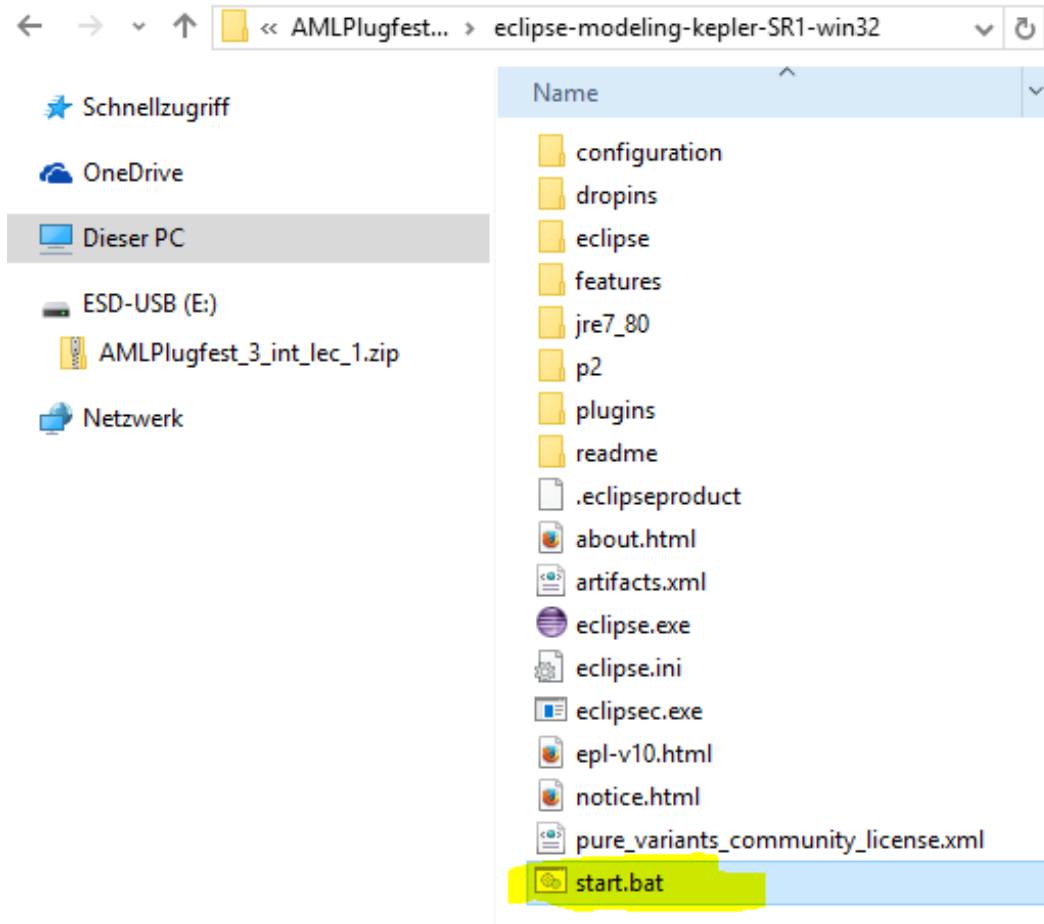


Setup II – Pure::Variants Licence

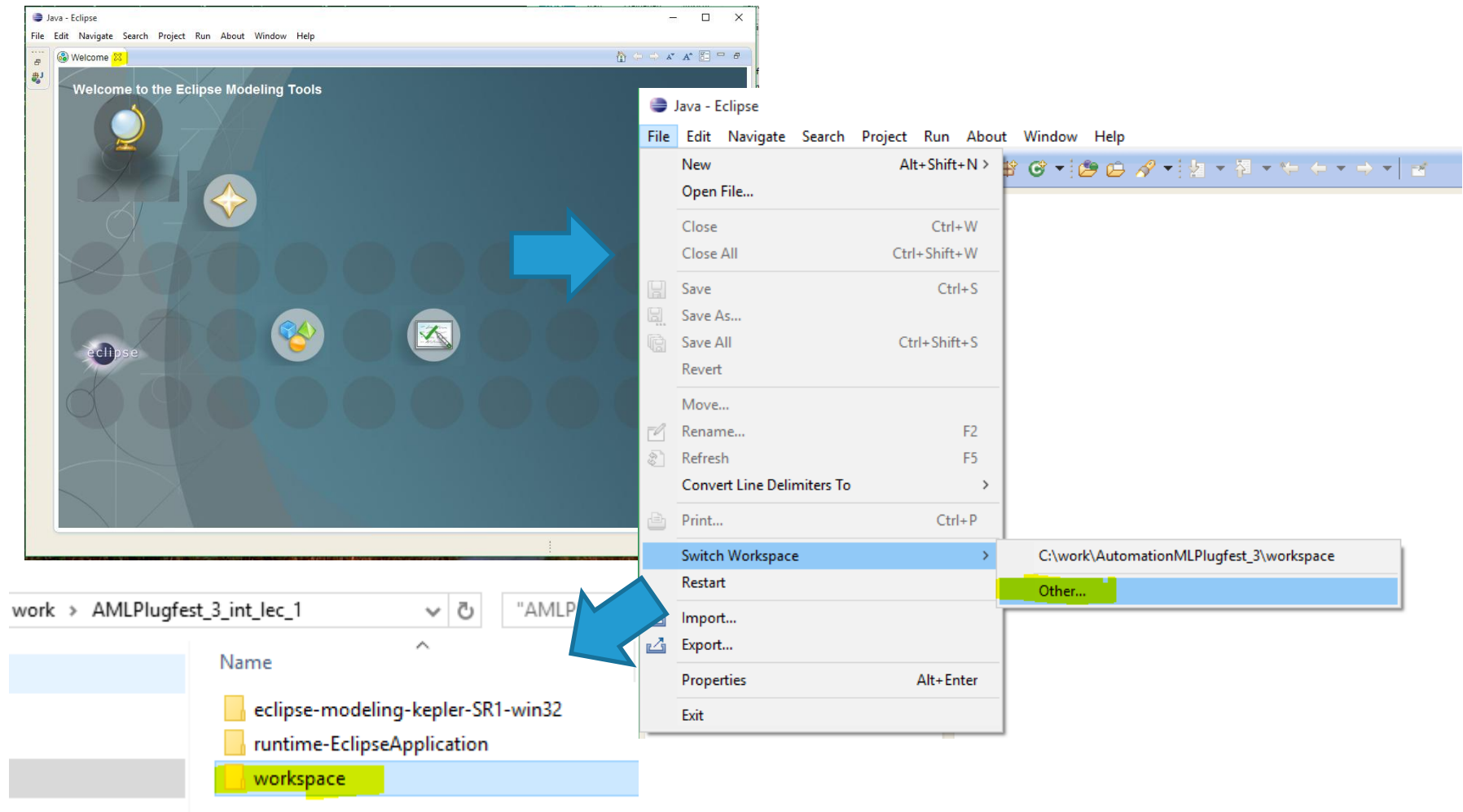
- only during workshop, for later usage please replace the licence with your own one



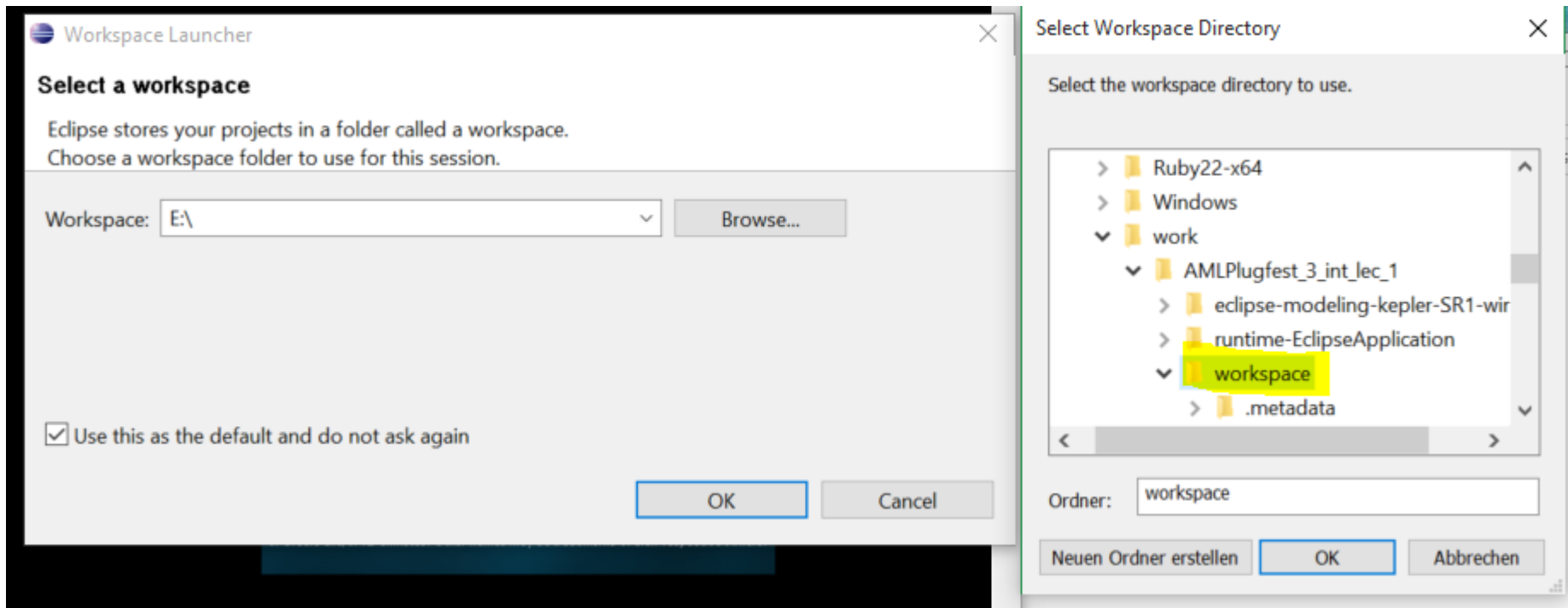
Setup III – Start Eclipse



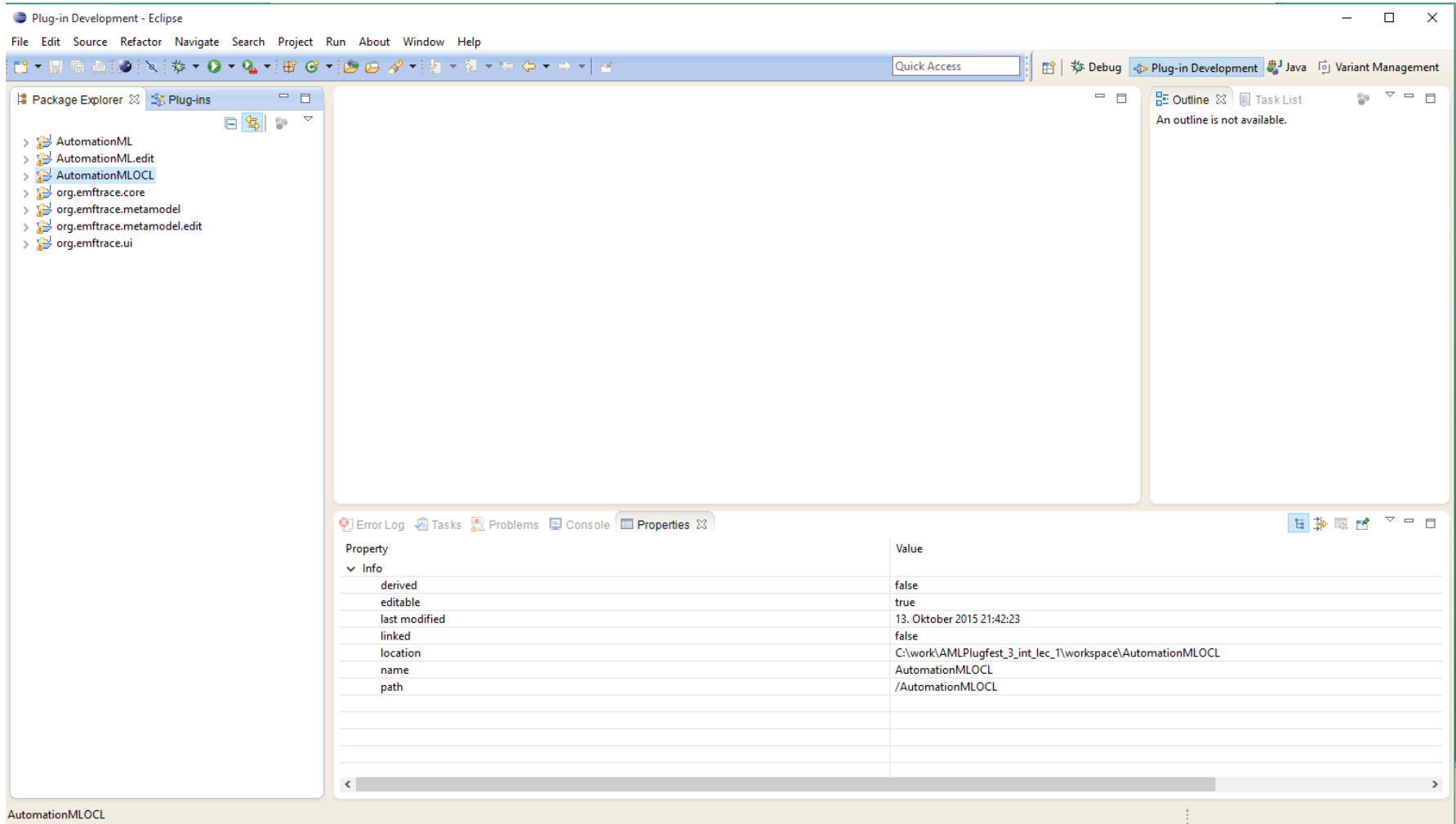
Setup IV – Select Workspace Case I



Setup IV – Select Workspace Case II

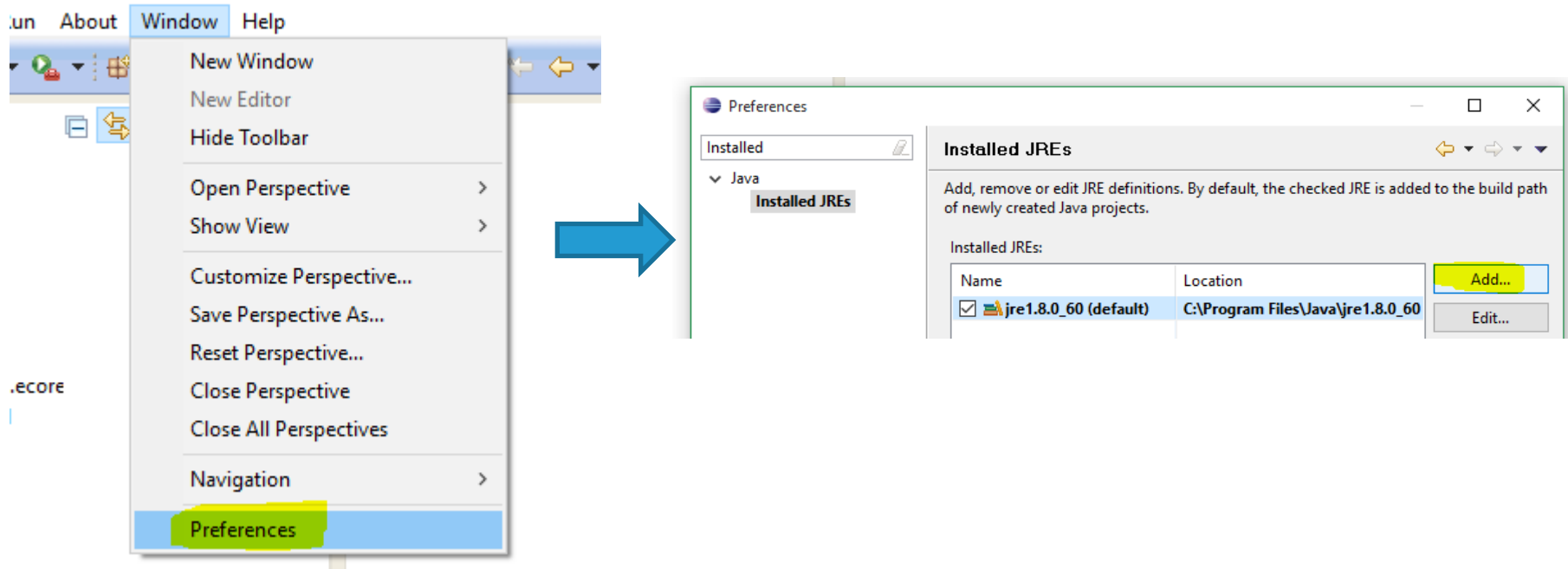


Setup V – Finished

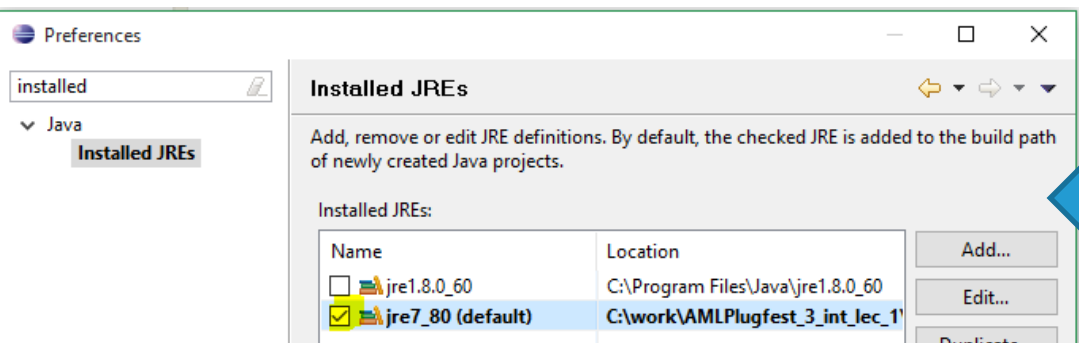
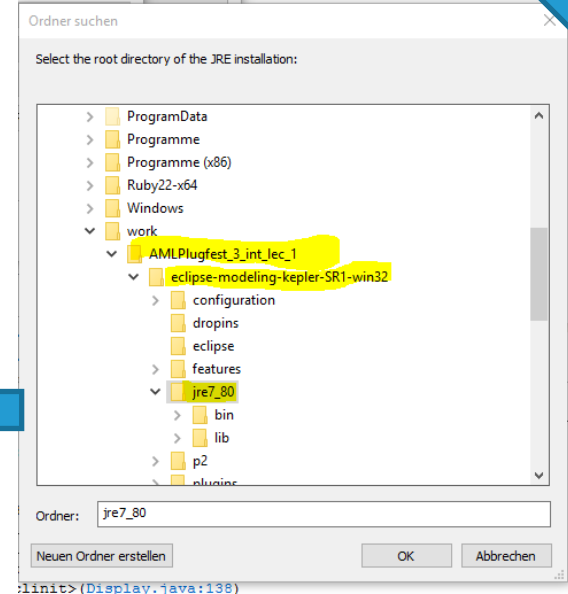
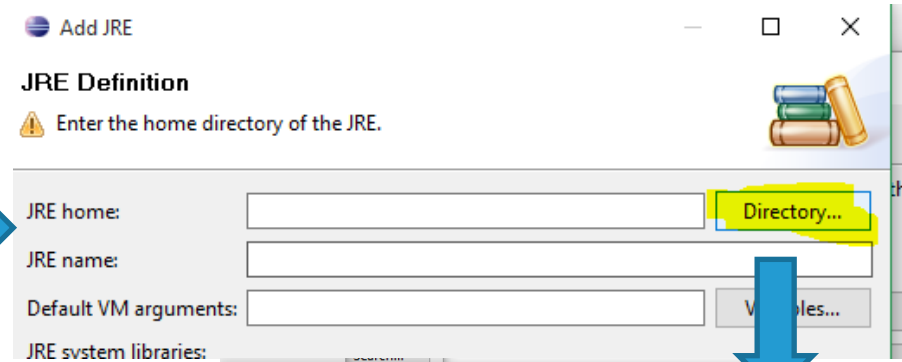
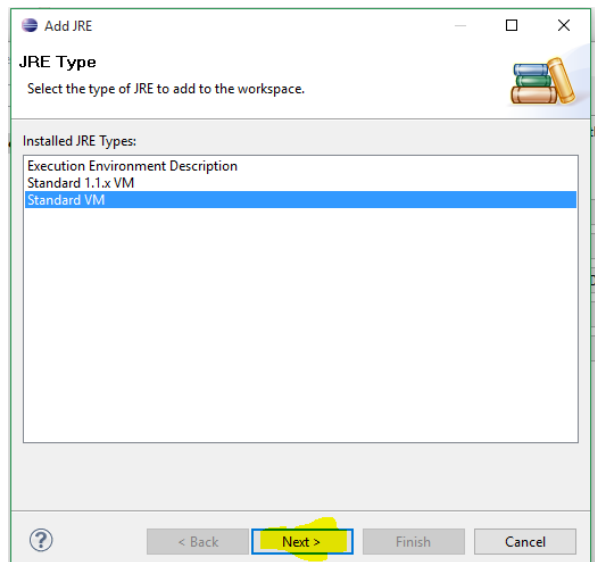


Setup VI – Java VM for the Eclipse RCPs I

- If a 64-Bit java version is installed, perhaps the EMFTrace-Client will not be started
- To solve it, the default java version has to be adapted
- It's located inside the eclipse-folder



Setup VI – Java VM for the Eclipse RCPs II

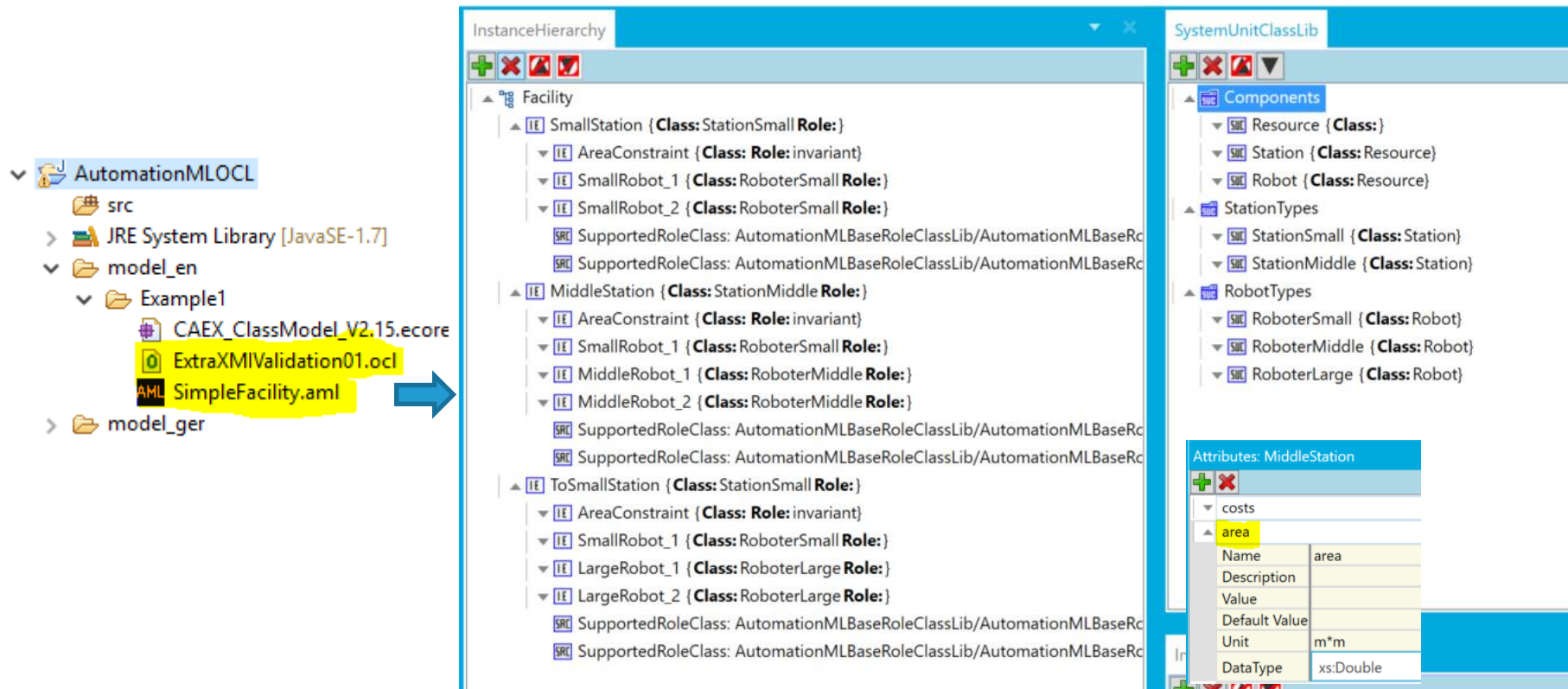


Exercise I

OCL-Constraints with AutomationML and Eclipse

OCL-Constraints with AutomationML and Eclipse I

- AML-File to analyse contains a simple facility structure
- Three different stations with different max. area size contains facilities



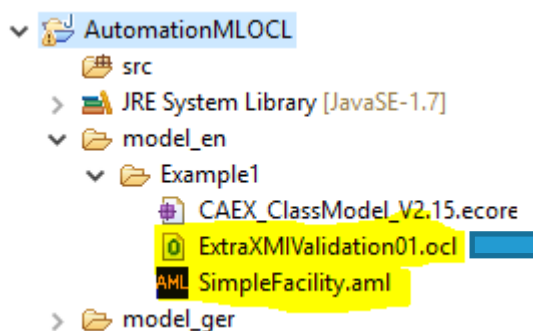
The screenshot shows the Eclipse IDE interface with the following components:

- Project Explorer (Left):** Displays the project structure for 'AutomationMLOCL'. It includes a 'src' folder, 'JRE System Library [JavaSE-1.7]', and a 'model_en' folder containing 'Example1'. Inside 'Example1', there are files for 'CAEX_ClassModel_V2.15.ecore', 'ExtraXMIVValidation01.ocl', and 'SimpleFacility.aml' (highlighted in yellow). A blue arrow points from 'SimpleFacility.aml' to the InstanceHierarchy view.
- InstanceHierarchy View (Center):** Shows a hierarchical tree of the facility structure. The root is 'Facility', which contains three main stations: 'SmallStation', 'MiddleStation', and 'ToSmallStation'. Each station has associated 'AreaConstraint' and 'SmallRobot' or 'MiddleRobot' or 'LargeRobot' elements. The 'ToSmallStation' also includes 'LargeRobot' elements. Each element is associated with a 'SupportedRoleClass' from 'AutomationMLBaseRoleClassLib'.
- SystemUnitClassLib View (Right):** Displays a list of classes and their roles. It includes 'Resource' (Class: Resource), 'Station' (Class: Resource), 'Robot' (Class: Resource), 'StationTypes' (StationSmall, StationMiddle, StationLarge), and 'RobotTypes' (RoboterSmall, RoboterMiddle, RoboterLarge).
- Attributes: MiddleStation (Bottom Right):** A table showing the attributes of the 'MiddleStation' class. The 'area' attribute is highlighted in yellow.

| Attributes: MiddleStation | |
|---------------------------|-----------|
| costs | |
| area | |
| Name | area |
| Description | |
| Value | |
| Default Value | |
| Unit | m*m |
| DataType | xs:Double |

OCL-Constraints with AutomationML and Eclipse II

- The OCL-File checks the amount size of all internal elements
- If it's failed against the max. size of the station the constraint is broken
- To get use of the OCL-Plugin inside Eclipse the AML-File must be converted into Eclipse readable format based on a CAEX-eCore-Metamodel



AutomationMLOCL

src

JRE System Library [JavaSE-1.7]

model_en

Example1

CAEX_ClassModel_V2.15.ecore

ExtraXMIVValidation01.ocl

SimpleFacility.aml

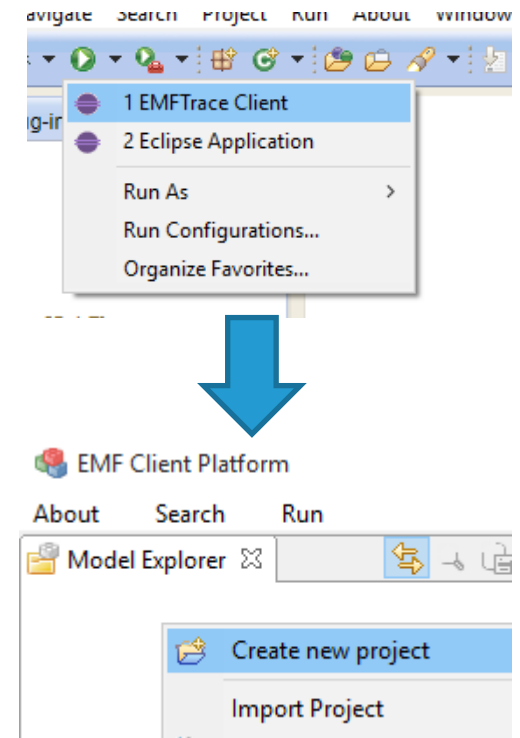
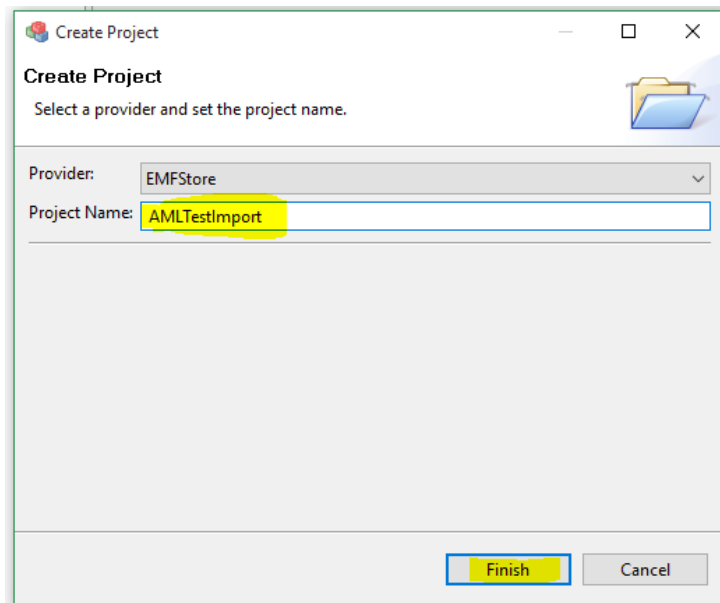
model_ger

```

1  import ecore : 'CAEX_ClassModel_V2.15.ecore'
2
3  package CAEXClassModelIV215
4
5  context InternalElement
6  def: asError(verdict : Boolean) : Boolean = if verdict then true else null endif
7  def: isStationElement(ie : InternalElement) : Boolean = ie.Name.matches('.*Station.*') and not ie.Name.
8  def: isRoleClassResource(ie : InternalElement) : Boolean = ie.SupportedRoleClass.RefRoleClassPath->include
9  def: getSizeOfAnElement(ie : InternalElement): Real =
10 ie.Attribute->select(Name='area').Attribute -> iterate
11 (att; result: Real = 1 |
12   if att.Name='length' or att.Name='width'
13   then result*att.Value.toReal()
14   else result
15   endif
16 )
17
18 inv OCLRule1('Rule 001: The size of a station must bigger than the sum of sizes of its all IE\'s'):
19 asError(self->forAll(isStationElement(self) implies
20   getSizeOfAnElement(self)
21   >
22   self.InternalElement->select(ie | isRoleClassResource(ie))
23   -> collect(ie: InternalElement | getSizeOfAnElement(ie)) -> sum()
24 )
25 )
26
27 endpackage
  
```

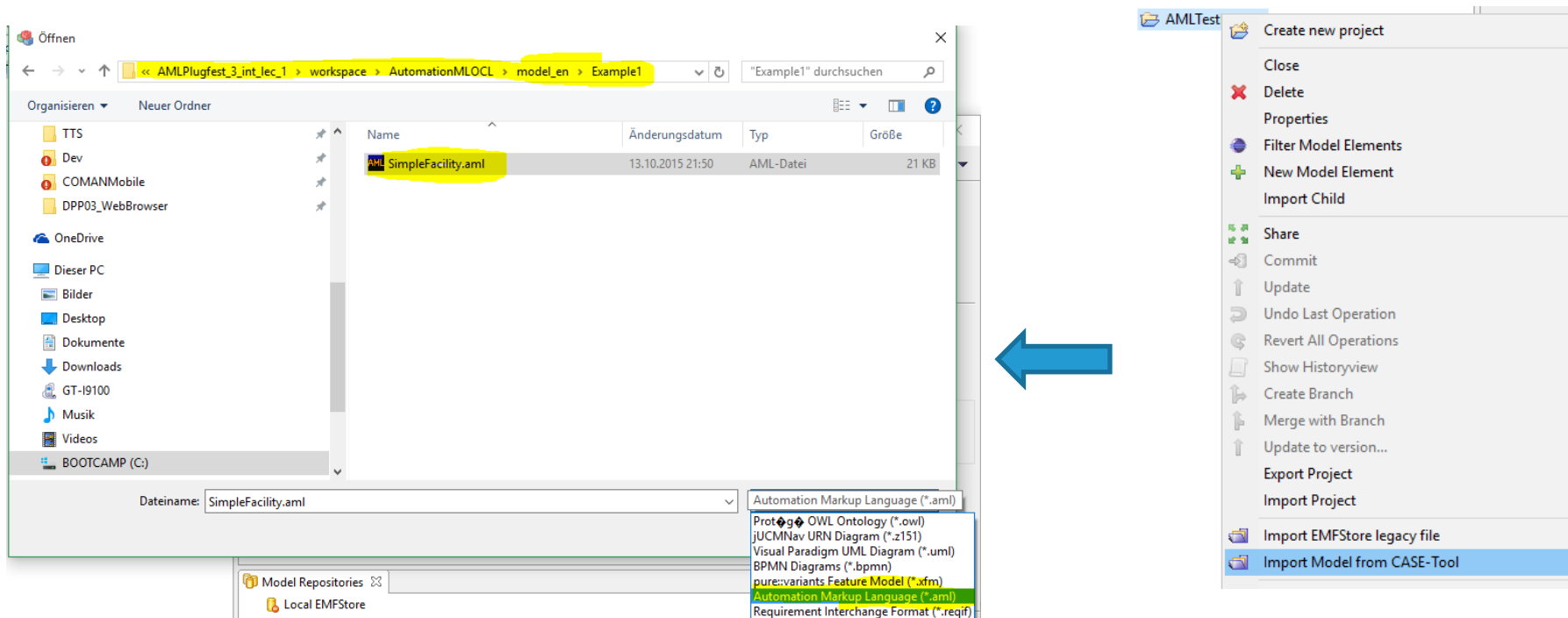
Convert an AML-File to Eclipse readable format I

- For the conversion the EMFTrace client has to be started
- Right click inside the model explorer menu and „Create new Project“



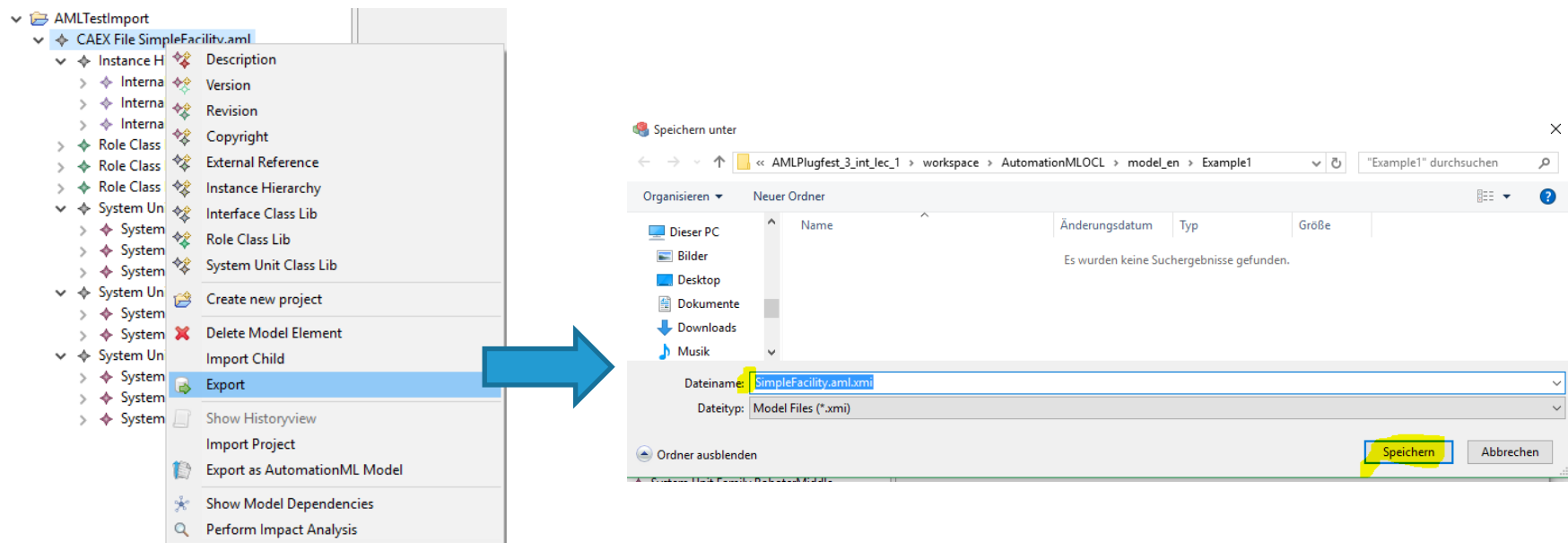
Convert an AML-File to Eclipse readable format II

- Right click on the AMLTestImport-EMFStore-Project and „Import Model from CASE-Tool“
- The SimpleFacility.aml is inside:
 - <<install-folder>>/workspace/AutomationMLOCL/model_en/Example1



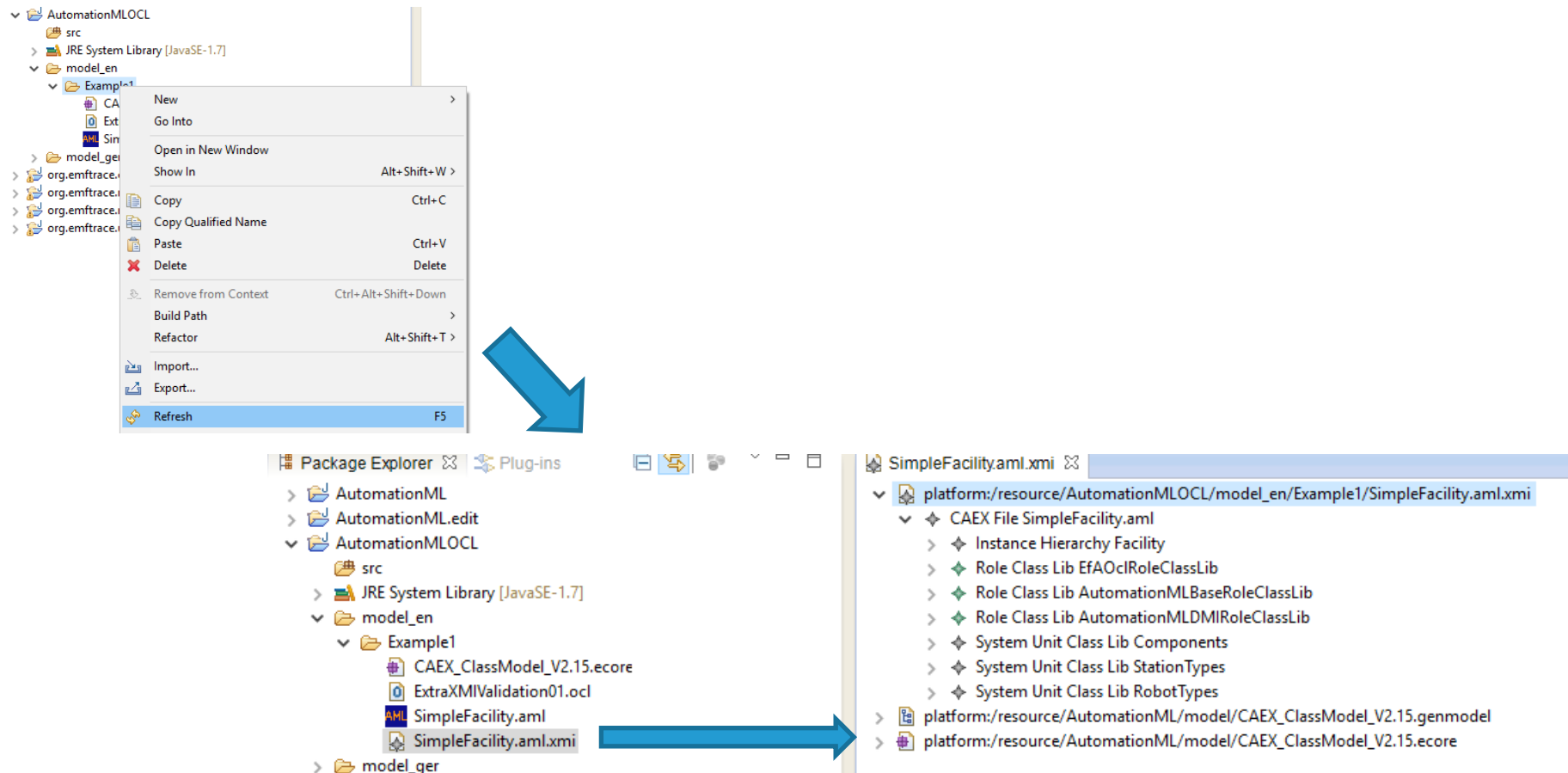
Convert an AML-File to Eclipse readable format III

- The generated file contains all infos from the AML-File
- The export from the EMFStore to an AML-File is also possible
- Next step is to export the emf-model to an XMI-File



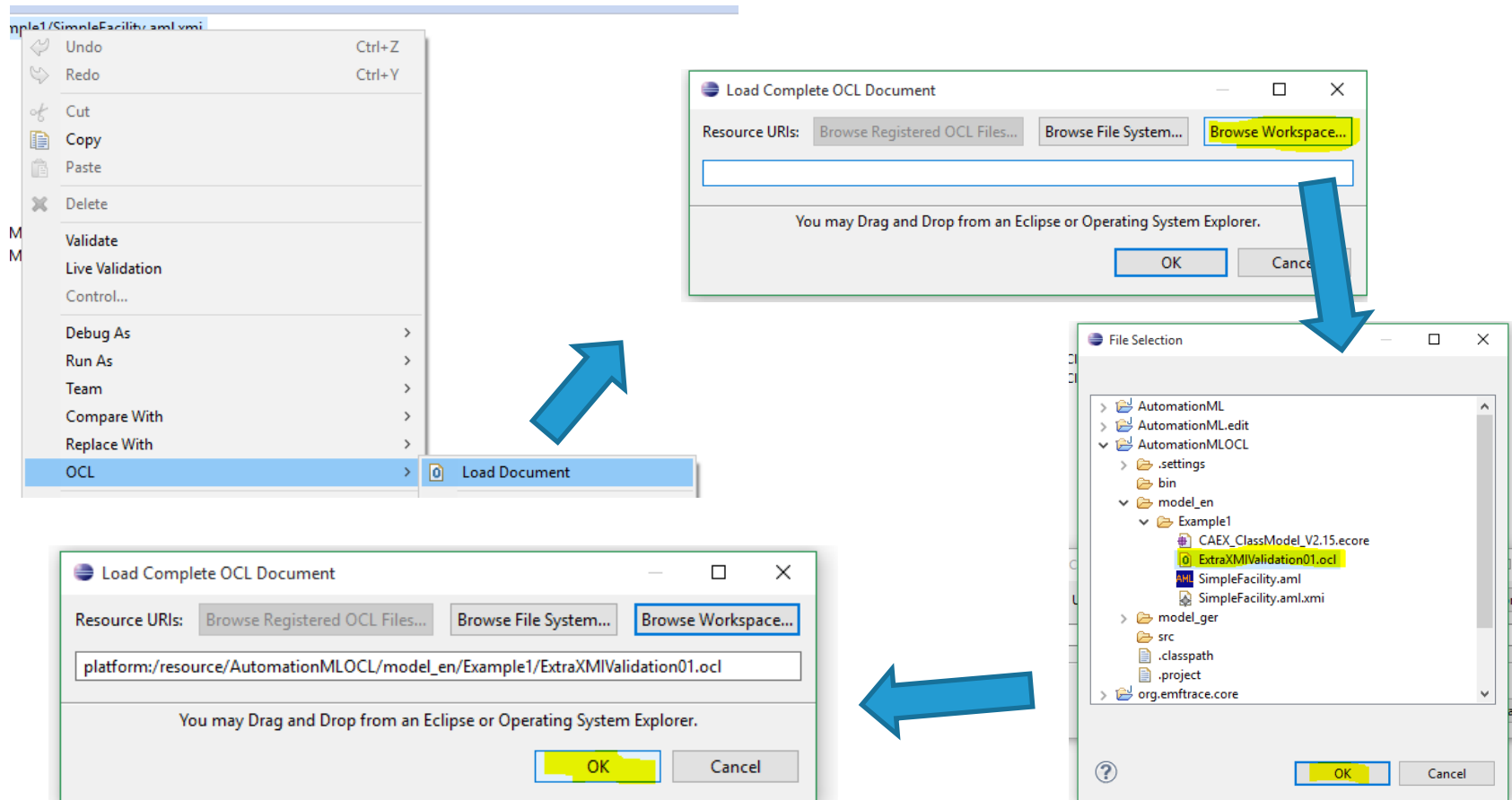
Check OCL-Constraints I

- Refresh the folder inside eclipse and open the SimpleFacility.aml.xml-File



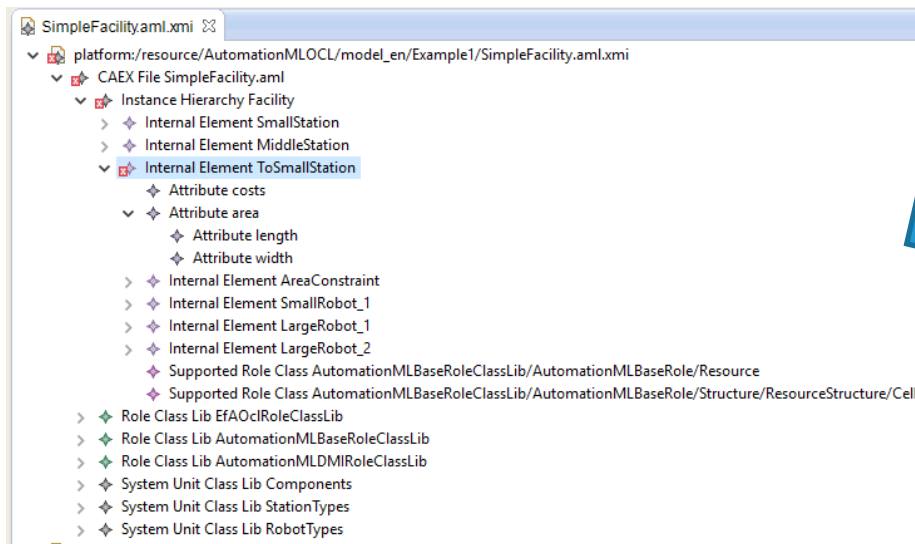
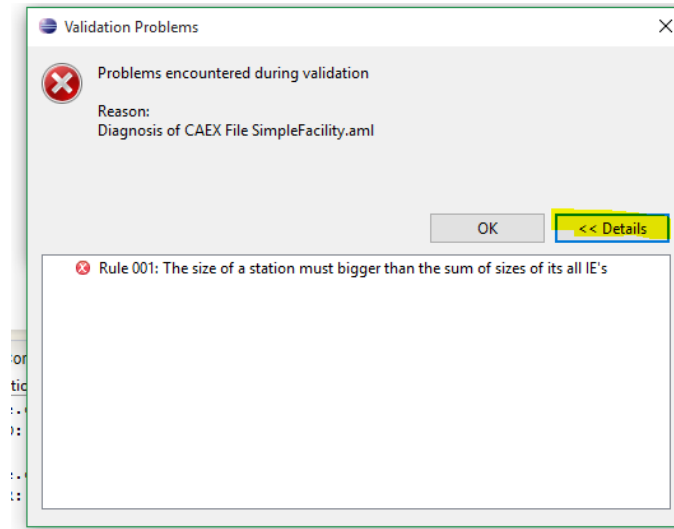
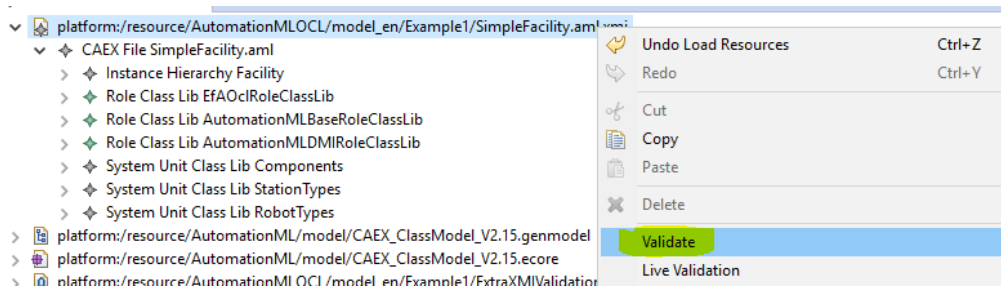
Check OCL-Constraints II

- Right click on platform:/resource.. to load the corresponding OCL-File



Check OCL-Constraints III

- Right click on platform:/resource.. and „Validate“

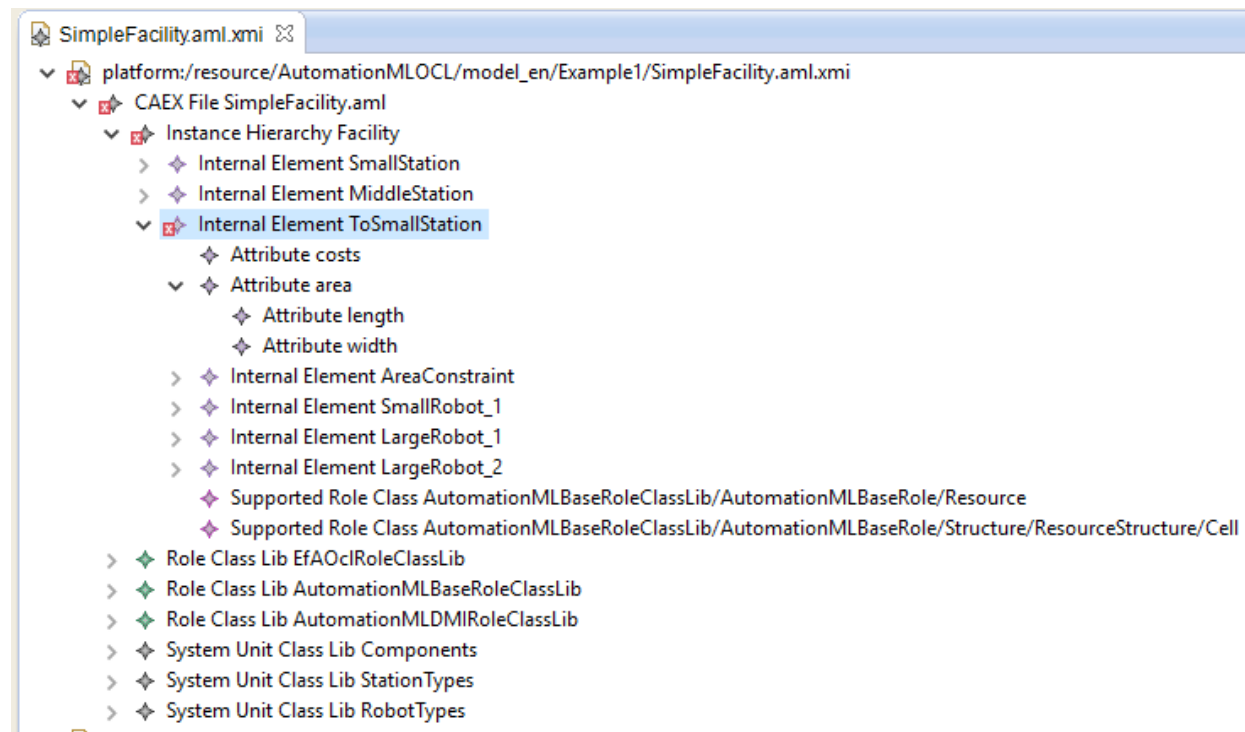


Result

The „ToSmallStation“ is with 16m² too small for two large and one small robot

Check OCL-Constraints - Summary

- After the AML-File is converted into the EMF-Format the OCL-Plugin can be used out-of-the-box
- The export to a AML-File is possible too



Exercise II

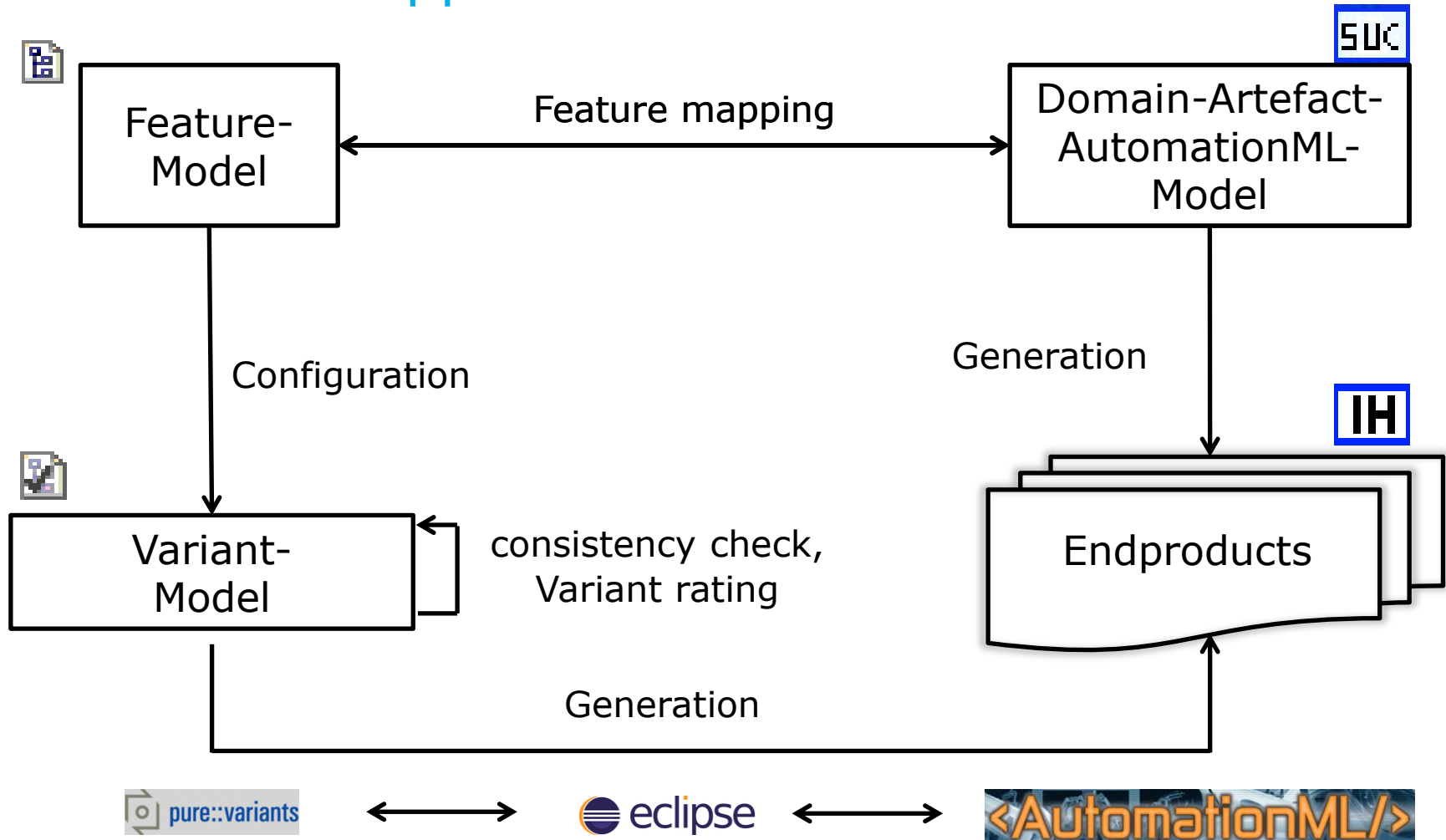
–

Configure a multi-variant
AutomationML-Model and
create a less-variant one via
tooling

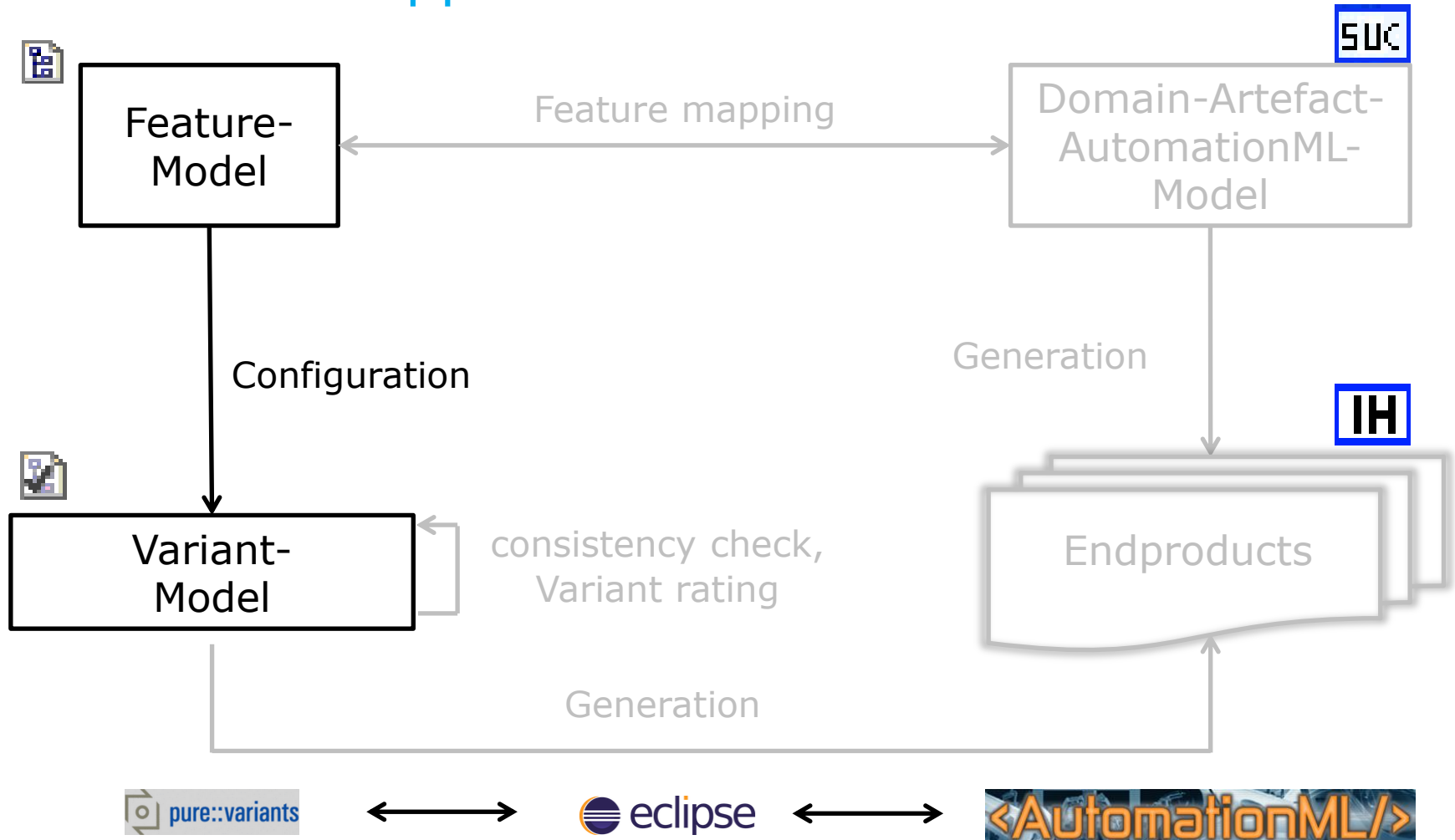
–

Theory

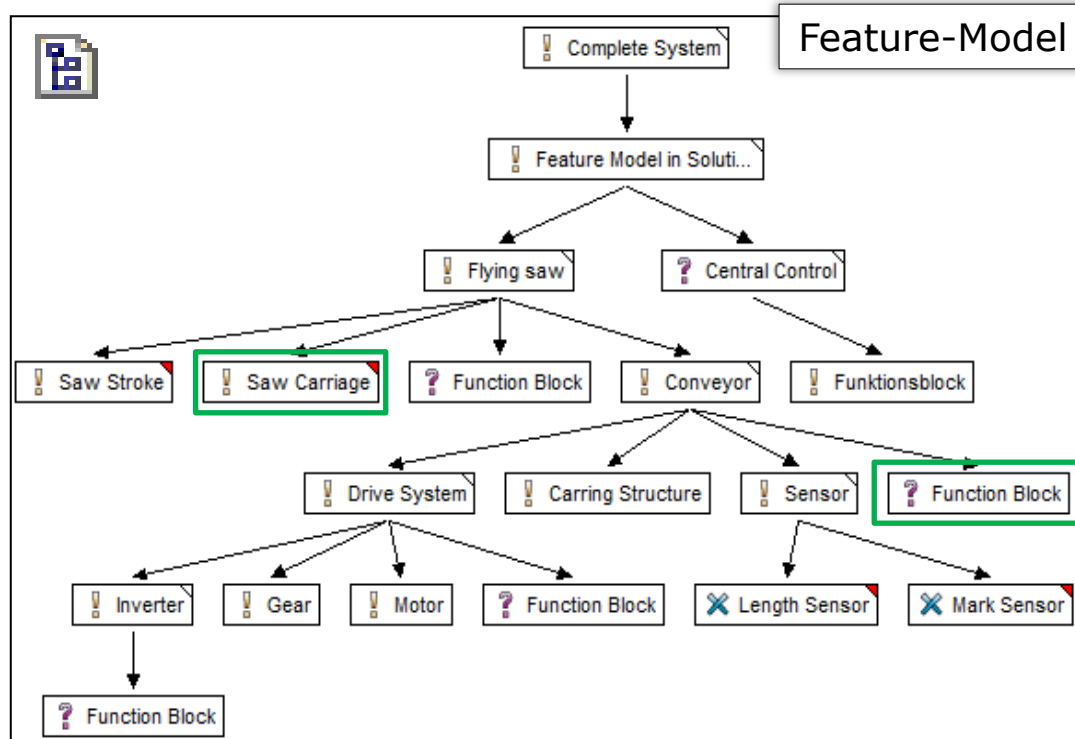
Product-Line Approach I



Product-Line Approach II

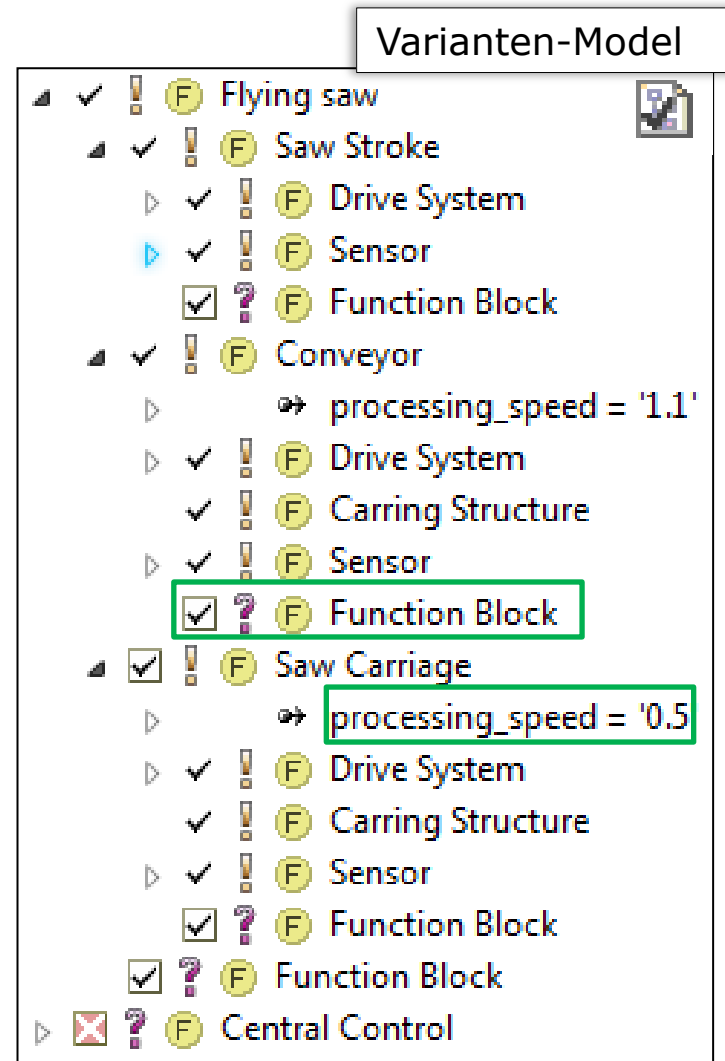


Configuration

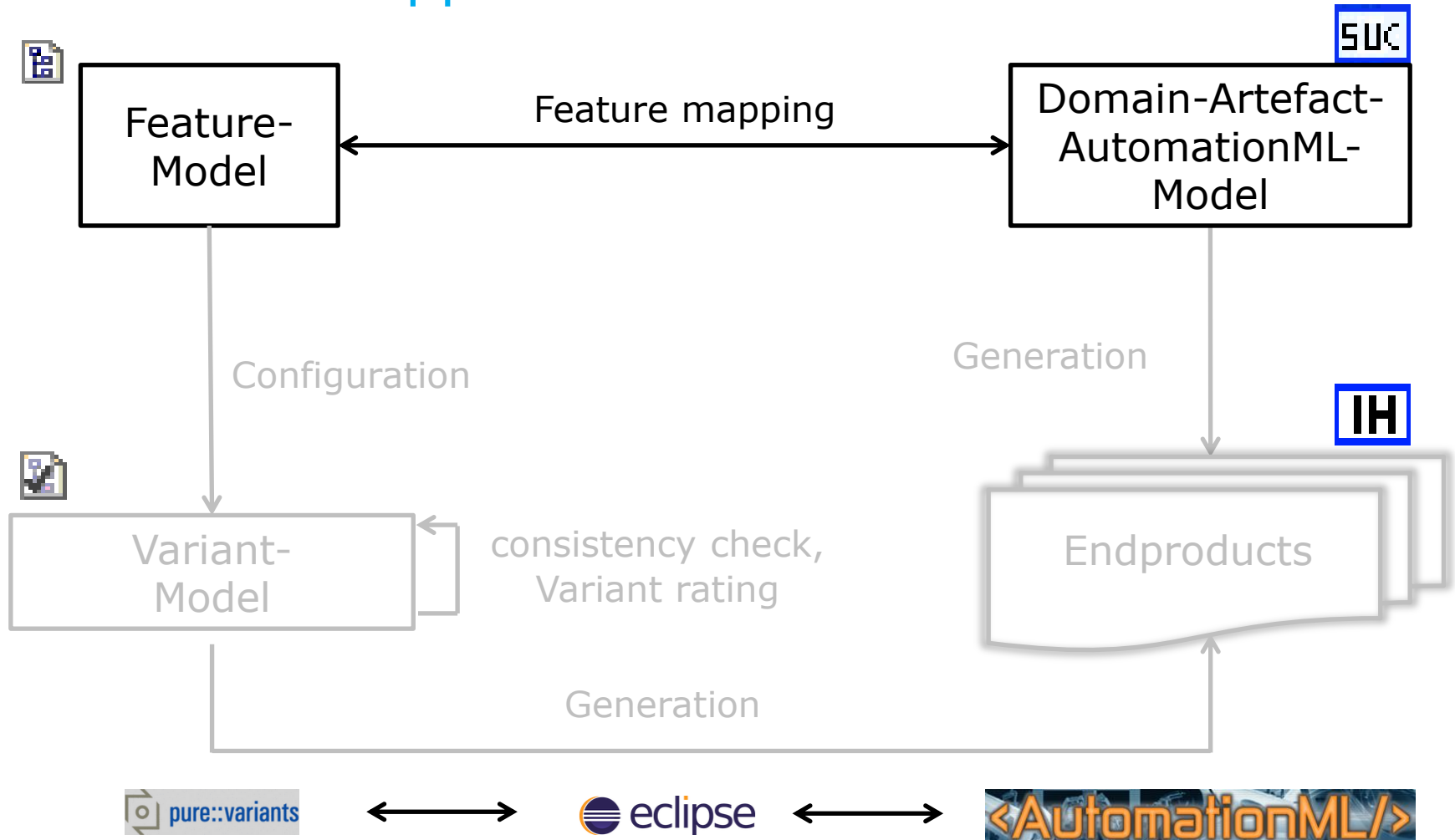


Configuration of a feature:

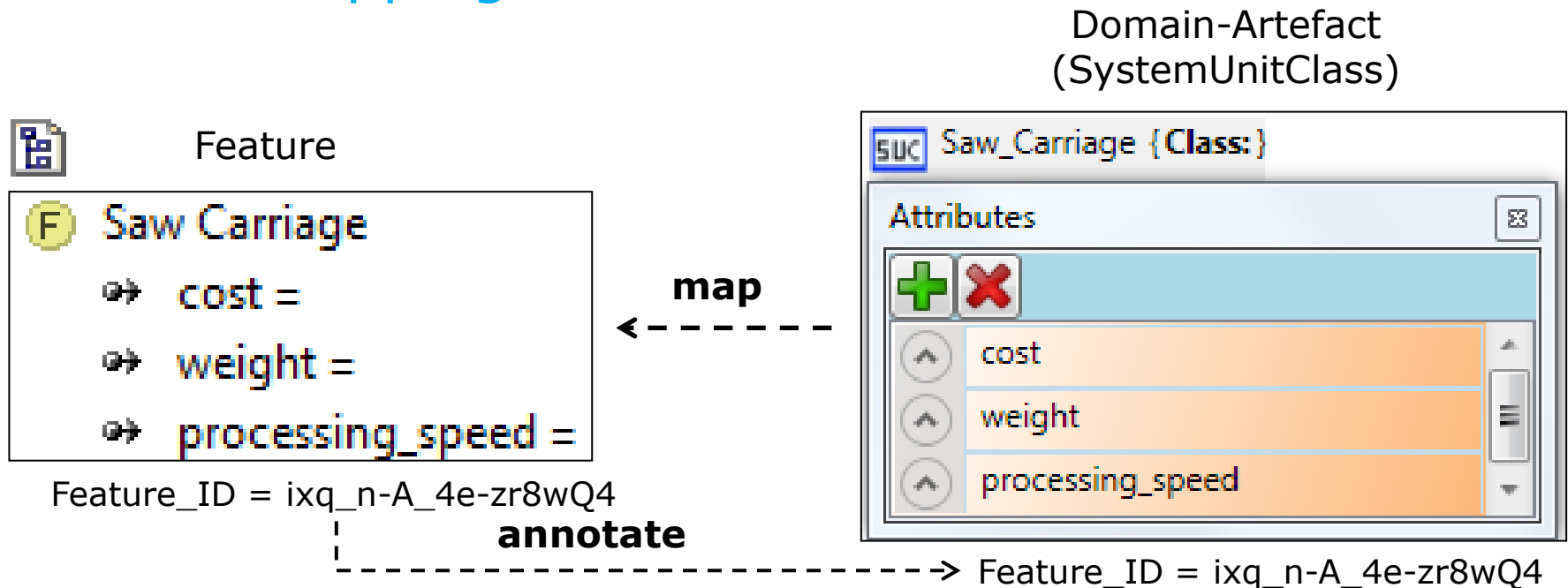
- Selektion and Deselektion
- Setting parameters



Product-Line Approach III



Feature-Mapping I



Mapping realised via **annotation** of
the **feature-ID**

Feature-Mapping II



Feature

Edit Feature

Edit 'Saw Carriage'

Edit general properties...

| General | Relations | Attributes | Restrictions | Constraints |
|--|-------------------|------------|--------------|-------------|
| Unique ID | ixq_n-A_4e-zr8wQ4 | | | |
| Unique Name | Saegeschlitten_SS | | | |
| Visible Name | Saw Carriage | | | |
| Class/Type | ps:feature | | | |
| <input checked="" type="radio"/> Mandatory | | | | |

Domain-Artefakt
(SystemUnitClass)

SystemUnitClassLib

- AD4AS_ContainerTemplateKatalogSystemUnitClassLib
 - System {Class: ContainerTemplate}
 - FliegendeSäge {Class: ContainerTemplate}
 - Antriebssystem {Class: ContainerTemplate}
 - Förderband {Class: ContainerTemplate}
 - Sägeschlitten {Class: ContainerTemplate}
 - newFeatureInterface {Class: FeatureInterface}

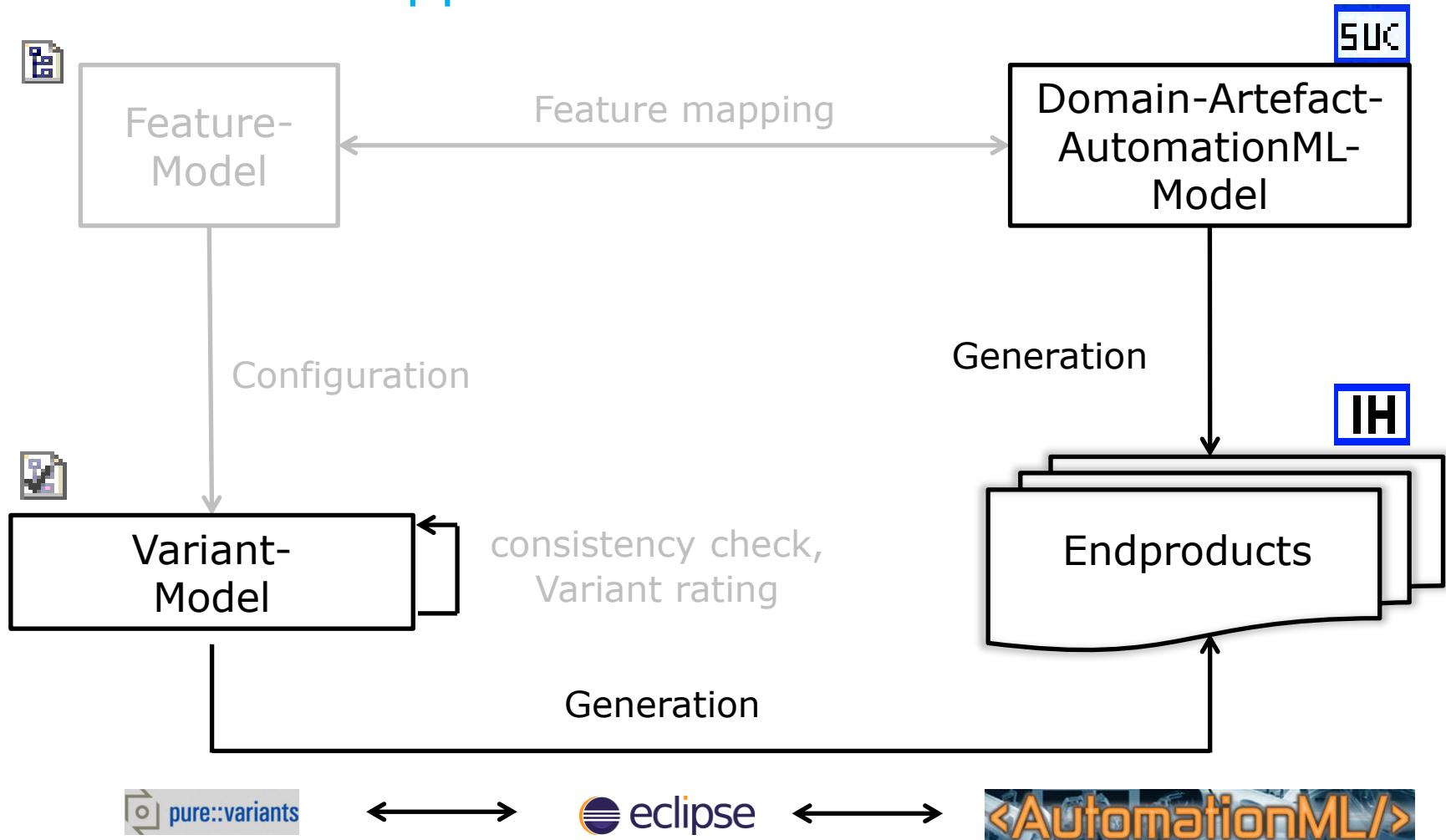
| Attributes: newFeatureInterface | |
|---------------------------------|-------------------|
| refURI | |
| featureIDs | |
| Name | featureIDs |
| Description | |
| Value | ixq_n-A_4e-zr8wQ4 |
| Default Value | |
| Unit | |
| DataType | xs:complexType |

Feature_ID = ixq_n-A_4e-zr8wQ4

-----> Feature_ID = ixq_n-A_4e-zr8wQ4

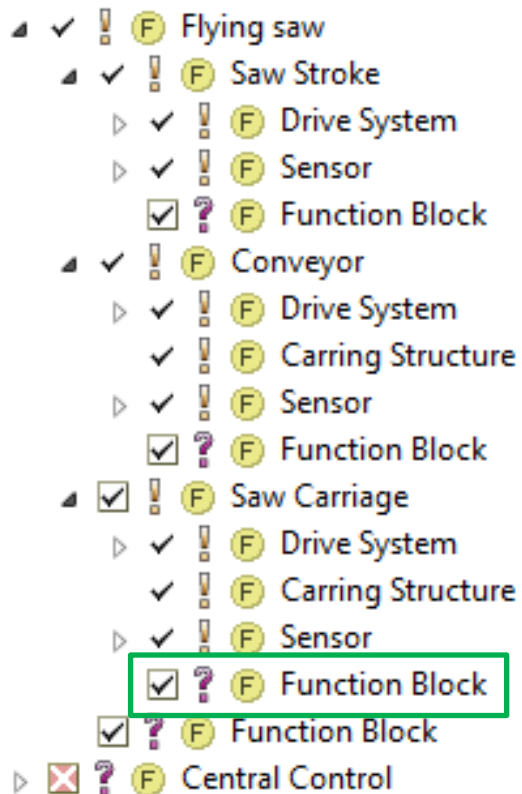
Mapping realised via **annotation** of
the **feature-ID**

Product-Line Approach III

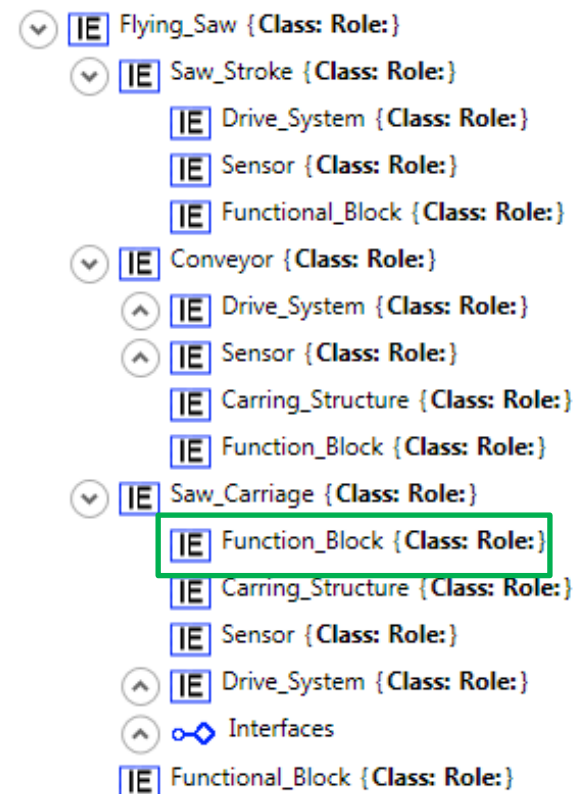


Generation

Varianten-Model



Productionline (InstanceHierarchy)



Systemhierarchie depends of the variant-configuration

Exercise II

–

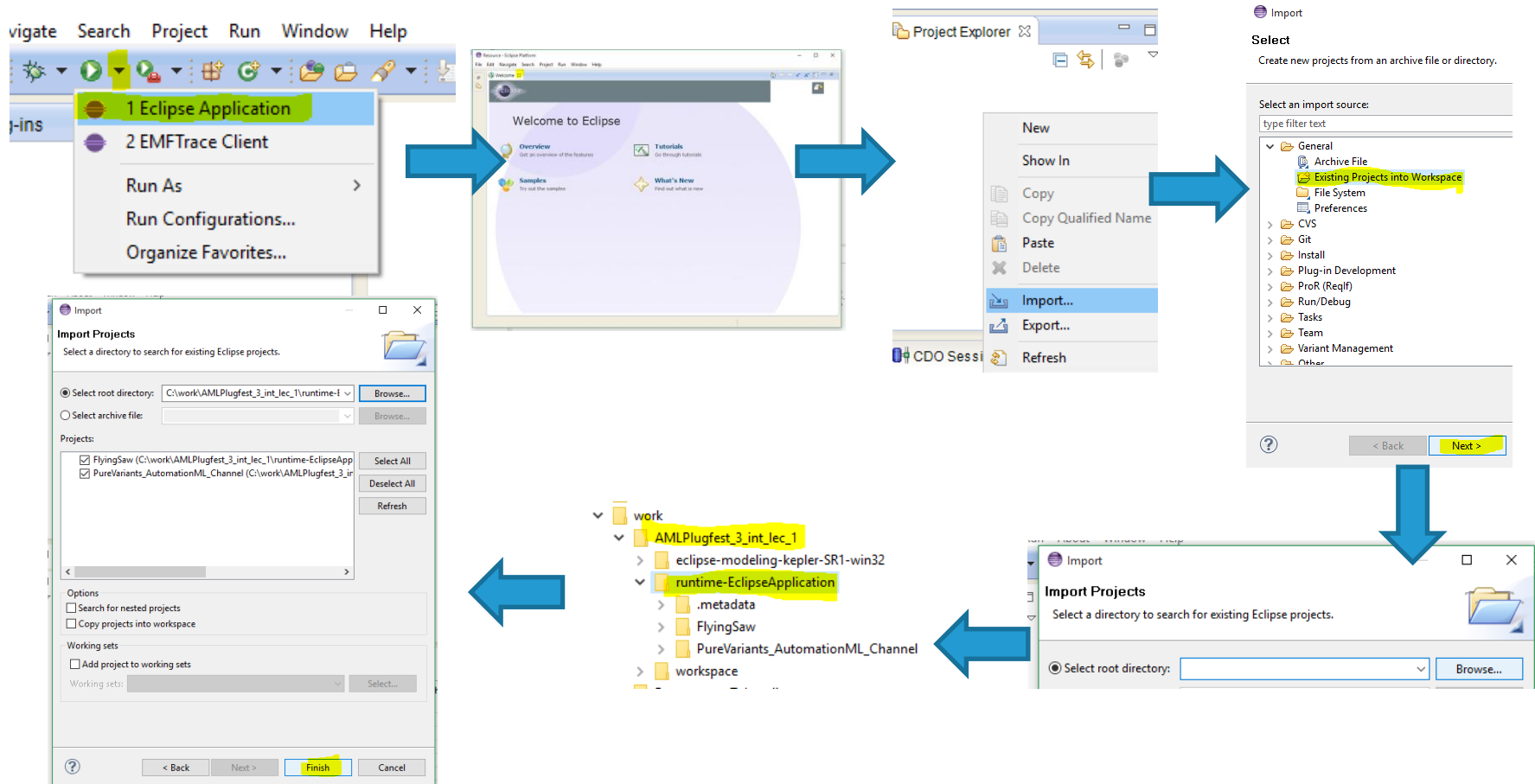
Configure a multi-variant
AutomationML-Model and
create a less-variant one via
tooling

–

Let's try

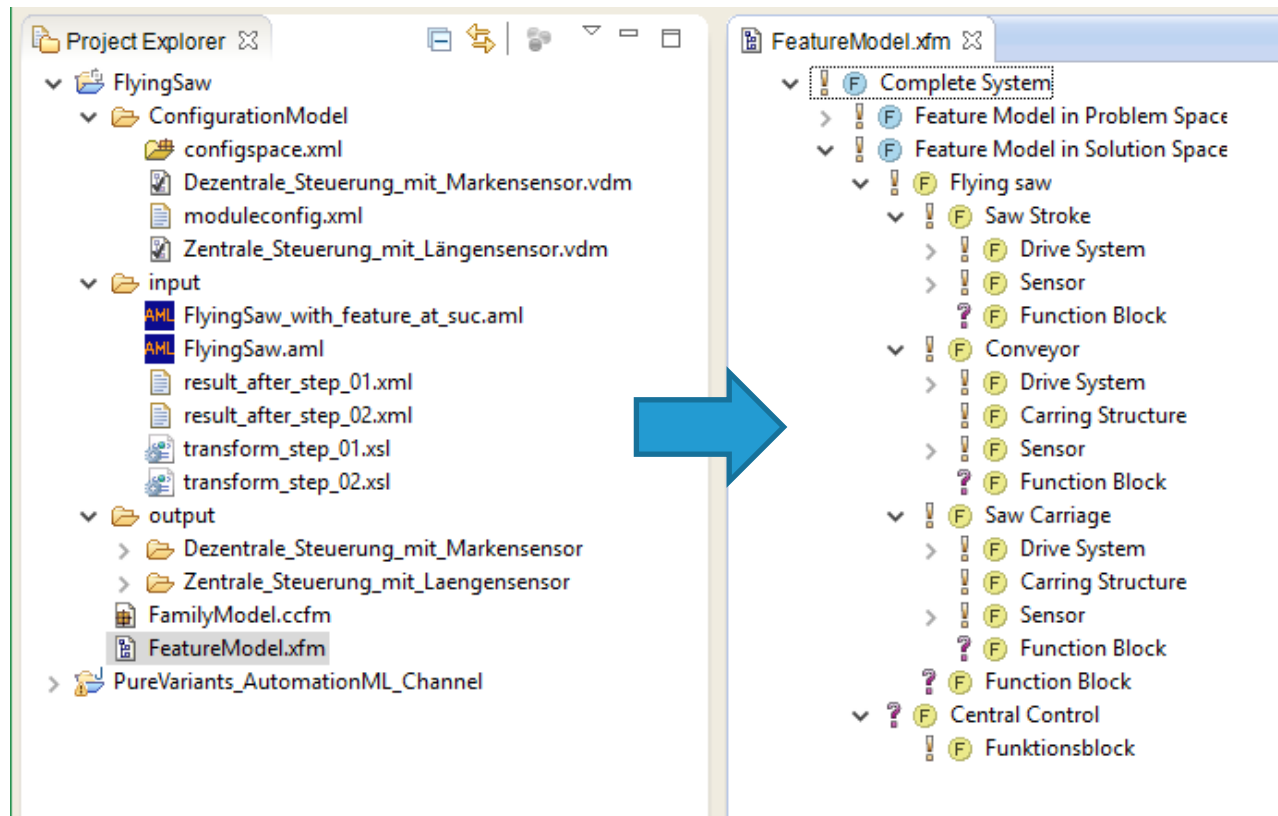
Configure multi-variant AutomationML-Model I

- Start the Eclipse Application and import the corresponding projects



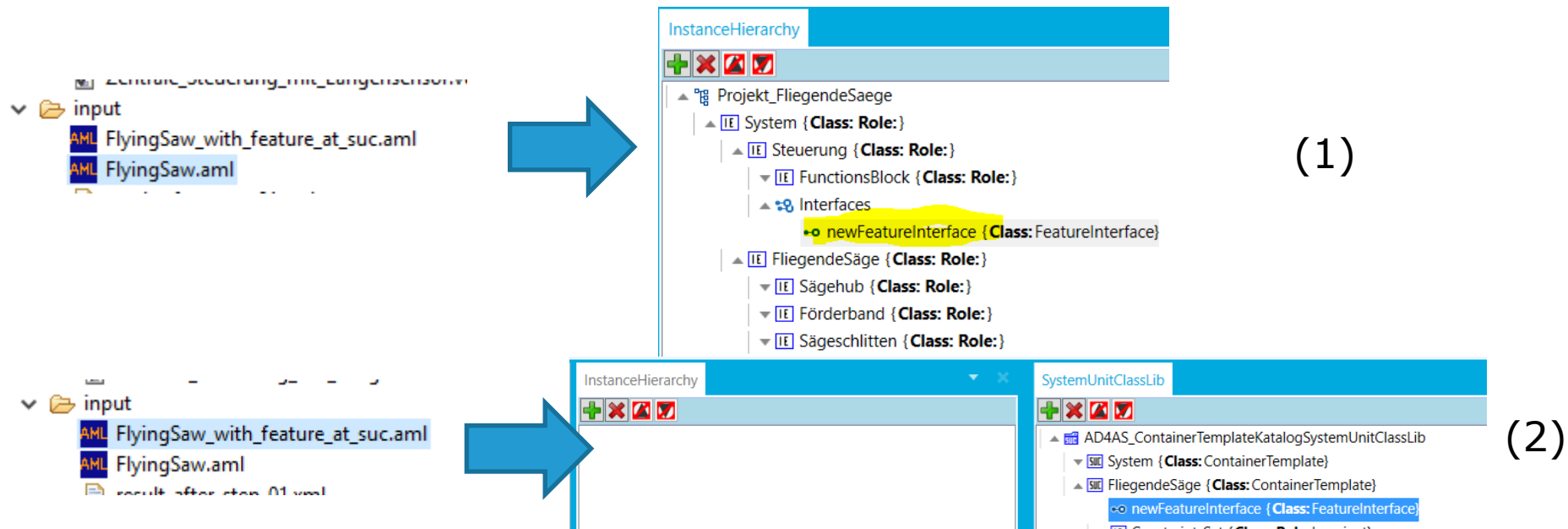
Configure multi-variant AutomationML-Model II

- The first relevant project is the flying saw
- Open the FeatureModel.xfm, to see the feature model



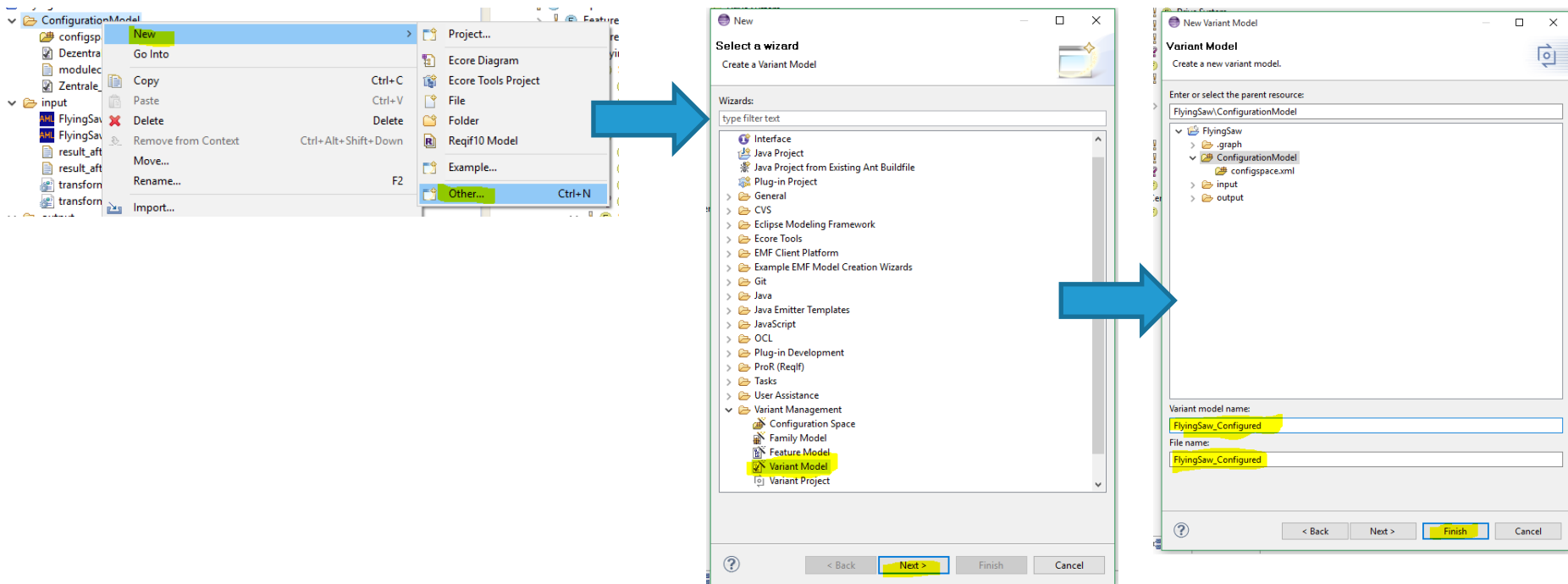
Configure multi-variant AutomationML-Model III

- The AML-File FlyingSaw.aml contains an 150%-InstanceHierarchy with all possible configurations (1)
- Normally the corresponding SUC contains the links to the feature (2)
- In the first step we will use the 150%-InstanceHierarchy



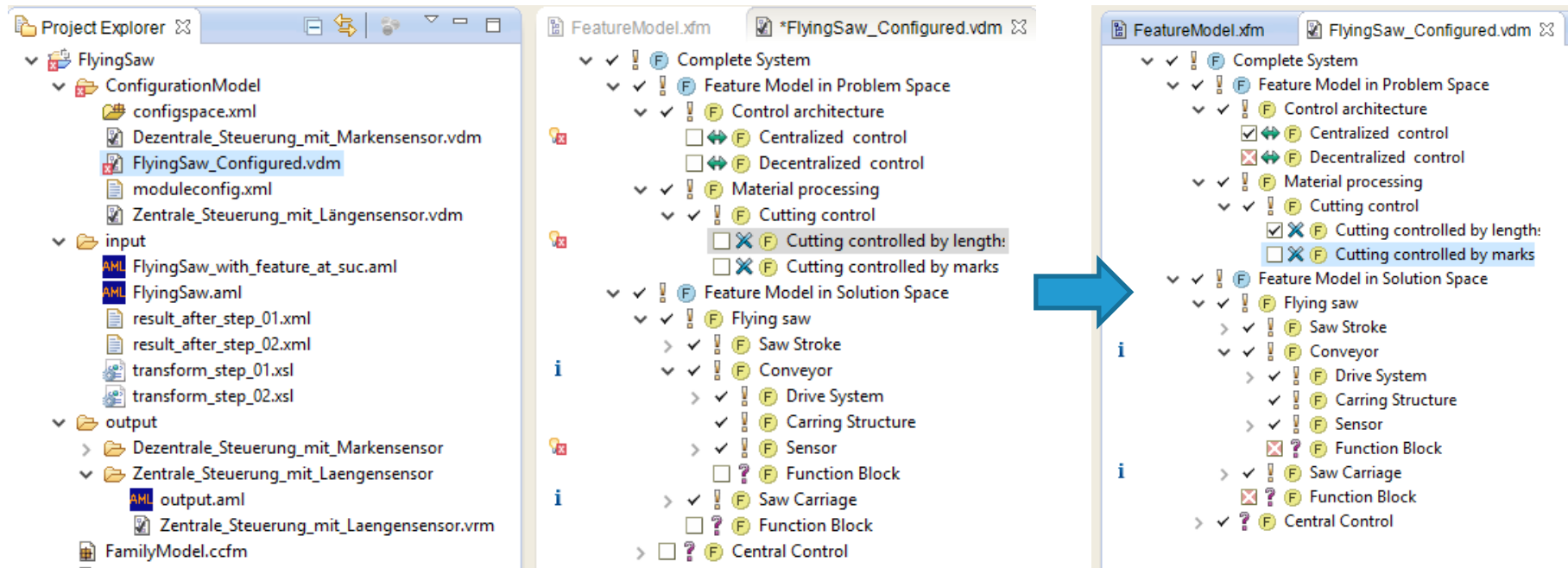
Configure multi-variant AutomationML-Model IV

- Create a variant model via right click on the „configuration model“
- Enter a „Variant model name“ and Finish



Configure multi-variant AutomationML-Model V

- To finalize the variant model configure the control architecture and the cutting control and save (File→Save)



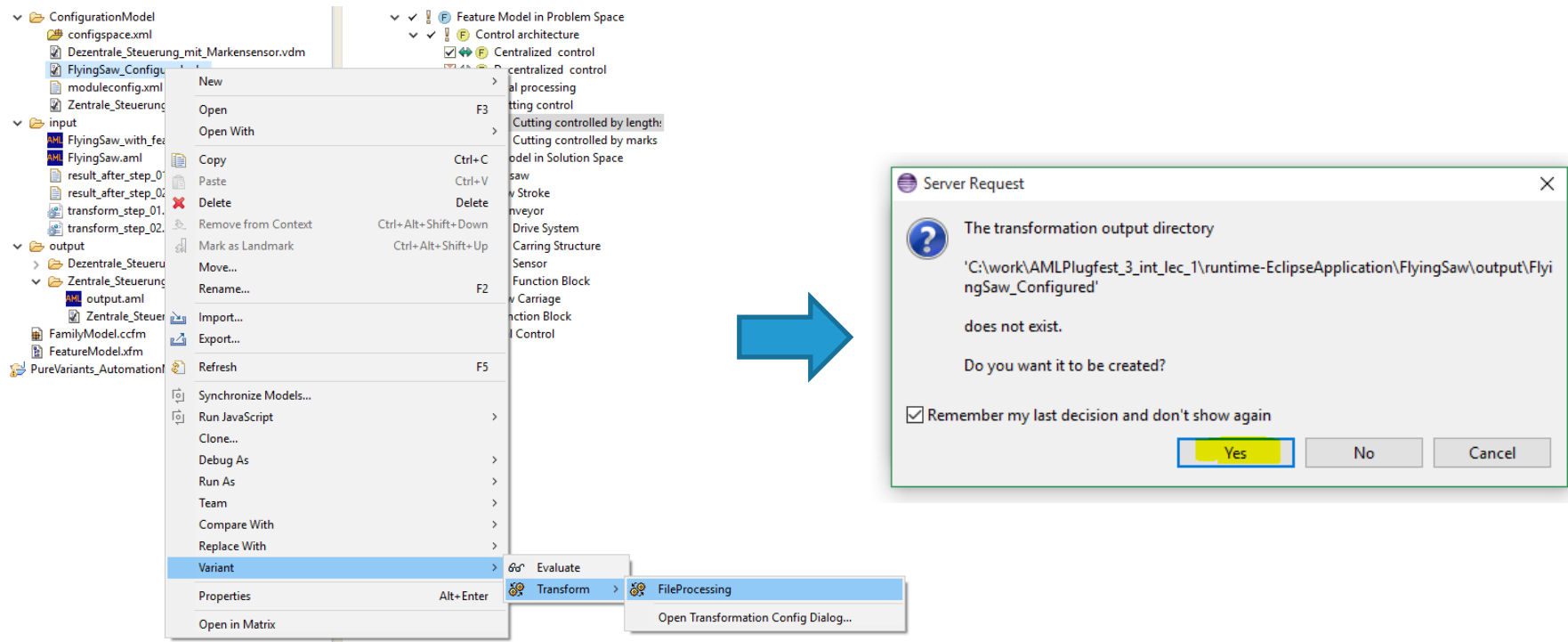
The screenshot displays the AutomationML software interface for configuring a multi-variant model. The interface is divided into three main panes:

- Project Explorer:** Shows a tree structure of files and folders. The 'FlyingSaw' folder is expanded, showing sub-folders like 'ConfigurationModel' and 'input'. The 'FlyingSaw_Configured.vdm' file is highlighted.
- FeatureModel.xfm:** Shows a hierarchical tree of features. The 'Complete System' feature is expanded, showing sub-features like 'Feature Model in Problem Space', 'Control architecture', 'Material processing', and 'Cutting control'. The 'Cutting controlled by length' and 'Cutting controlled by marks' options are visible.
- FlyingSaw_Configured.vdm:** Shows the resulting configuration. The 'Feature Model in Solution Space' is expanded, showing sub-features like 'Flying saw', 'Saw Stroke', 'Conveyor', 'Drive System', 'Carring Structure', 'Sensor', 'Function Block', 'Saw Carriage', and 'Central Control'.

A large blue arrow points from the 'FeatureModel.xfm' pane to the 'FlyingSaw_Configured.vdm' pane, indicating the transformation of the feature model into the configured model.

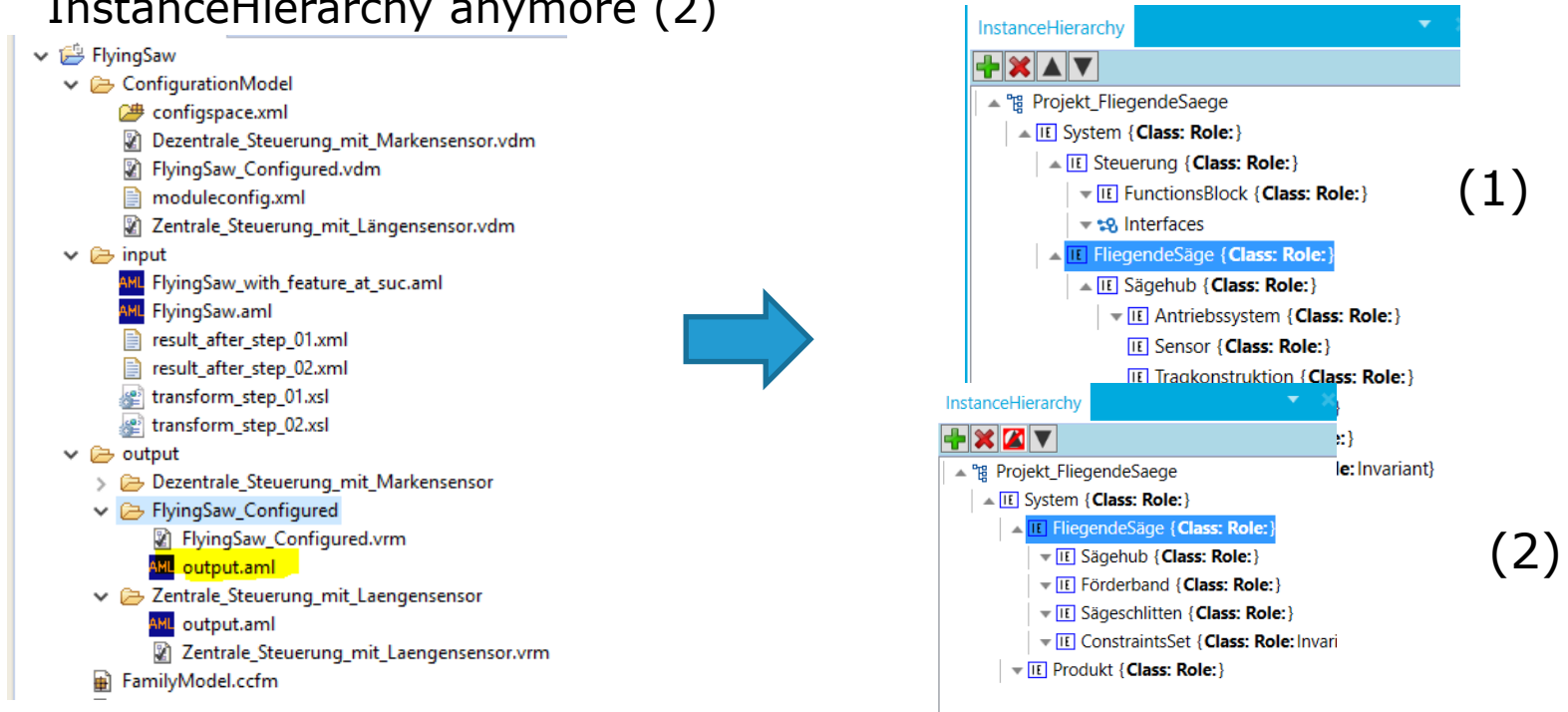
Configure multi-variant AutomationML-Model VI

- Finally create the less-variant AutomationML-File via the „FileProcessing“



Created less-variant AutomationML-File

- The new folder below the „output“-Folder contains the less-variant AML-File
- If you choosed the centralized control, all function blocks below the „FlyingSaw“-InternalElement are gone (1)
- If you choosed the decentrailized control, the main control isn't inside the InstanceHierarchy anymore (2)



Exercise III

–

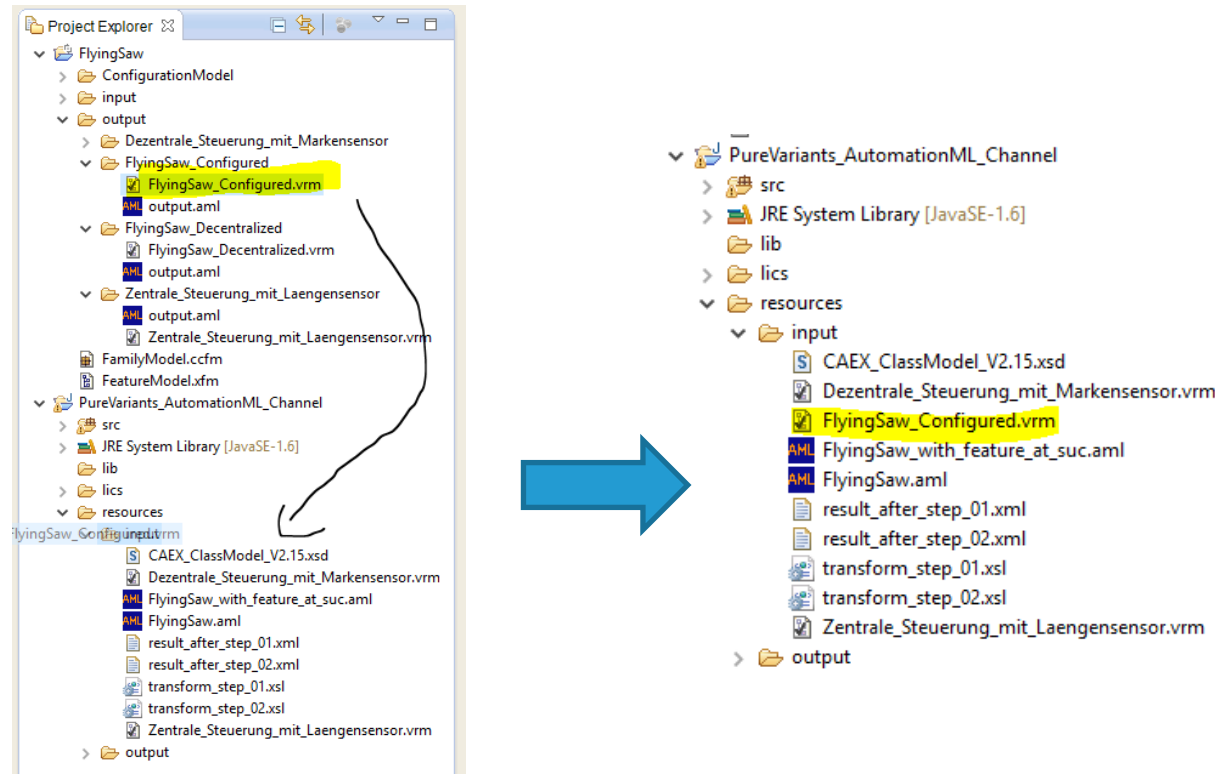
Configure a multi-variant
AutomationML-Model and
create a less-variant one
programmatically

–

Let's try

Programmatically way I

- Expand the „PureVariants_AutomationML_Channel“ to resources/input
- Drag&Drop the created FlyingSaw_Configured.vrm to the input folder of the „PureVariants_AutomationML_Channel“



Programmatically way II

- Go to the Trigger.java-File and adapt the fileName if you don't choose „FlyingSaw_Configured“
- Start the JavaApplication via right-click inside the editor and „Run As“→Java Application



The screenshot shows an IDE with the Project Explorer on the left and the Trigger.java file open in the editor. The Project Explorer shows a project structure with a package de.skaegebein containing a Trigger.java file. The Trigger.java file contains the following code:

```

package de.skaegebein;

import java.util.ArrayList;

/**
 * @author kaegebein
 */
public class Trigger {

    private static String SEP = System.getProperty("file.separator");
    // "Dezentrale Steuerung mit Markensensor.vrm"
    public static String fileName = "FlyingSaw_Configured";

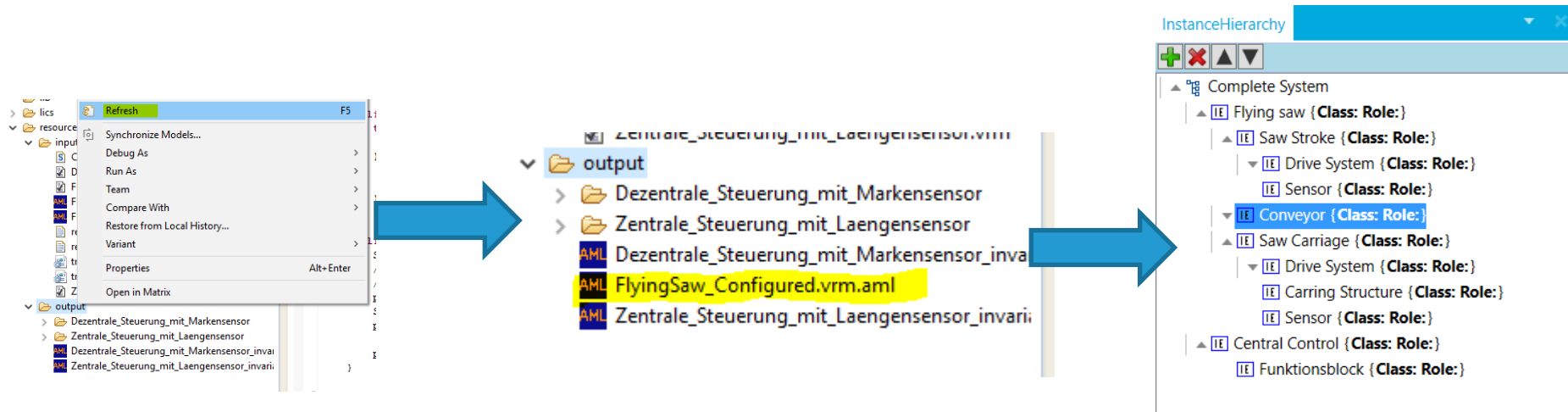
    public static String VRM_FILE_NAME = fileName + ".vrm";
    public static String AML_TEMPLATE_FILE_NAME = "FlyingSaw_with_feature_at_suc.aml";
    public static String AML_OUTPUT_FILE = VRM_FILE_NAME + ".aml";

    public static void main(String[] args) {
        try {
            new Trigger().run();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
  
```

A blue arrow points from the `fileName` variable in the code to the right-click context menu. The context menu is open, showing various options. The "Run As" option is highlighted, and a submenu is visible showing "1 Java Application" and "Run Configurations...".

Programmatically way III

- Refresh the output-Folder → the generated file is shown up
- The generated AML-File contains the hierarchie structure based on the feature model
- In comparison to the exercise before, the basic input was a variant flat SUC-Model



Summary

- The provided toolset realises the configuration of an variant AutomationML-Model and the creation of an invariant InstanceHierarchy
- Another possibility is the conversion of an AutomationML-Model into EMF and vice versa
- This realises the usage of different Eclipse-Plugins without loosing the core idea of AutomationML as an exchange format
- Moreover with the CAEX-eCore-Model there is a programmatic access in java out-of-the-box
- The two exercise should show you the usage of the OCL- and pure::variants-Plugins as a example what can be possible inside the eclipse world

Thank you for your attention!

—

Sven Kägebein
kaegebein@inpro.de
030-399-97-282

Yibo Wang
wang@informatik.uni-hamburg.de
040-428-83-2703

inpro

Innovationsgesellschaft für fortgeschrittene
Produktionssysteme in der Fahrzeugindustrie mbH

Hallerstr. 1

D-10587 Berlin

www.inpro.de

Technical infos

- Eclipse-Installation
 - <http://sourceforge.net/p/emftrace/wiki/Installation/>
 - Pure::Variants-Communityedition via UpdatSite (<http://www.pure-systems.com/pvce-update>)
 - Eclipse OCL only via direct download an local installation link (Install new software)
<http://archive.eclipse.org/modeling/mdt/ocl/downloads/drops/5.0.0/R201406091420/mdt-ocl-Update-5.0.0.zip>
 - ReqIf-Plugin (<http://download.eclipse.org/rmf/updates/releases>)