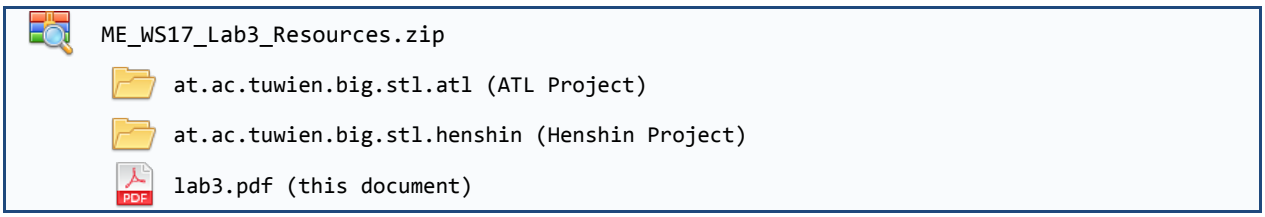
Model Engineering Lab 188.923 IT/ME VU, WS 2017/18	**Assignment 3**
Deadline: Upload (ZIP) in TUWEL until Sunday, December 17th, 2017, 23:55 Assignment Review: Wednesday, January 17th, 2018	25 points

Model-to-Model Transformation

The goal of this assignment is to develop model-to-model transformations for the *Simple Transportation Line Modeling Language* (STL) using ATL and Henshin. In Part A of the assignment, you will develop an ATL transformation that translates STL models into *AutomationML/CAEX* models. In Part B of this assignment, you will develop a Henshin transformation that applies a refactoring on STL models.

Assignment Resources



Setting up your workspace

Before starting this assignment, make sure that you have all necessary components installed in your Eclipse. A detailed installation guide can be found in the TUWEL course[1].

Import the projects provided in the assignment resources into your workspace. Therefore select *File → Import → General/Existing Projects into Workspace → Select archive file → Browse*. Choose the downloaded archive *ME_WS17_Lab3_Resources.zip* and import the following projects:

- `at.ac.tuwien.big.stl.atl`
- `at.ac.tuwien.big.stl.henshin`

It is recommended that you read the complete assignment specification at least once. If there are any parts of the assignment specification or the provided resources that are ambiguous to you, don't hesitate to ask in the forum for clarification.

[1] Eclipse Setup Guide: https://tuwel.tuwien.ac.at/mod/page/view.php?id=388945

Part A: ATL Transformation

In the first part of this assignment, you have to develop an ATL model transformation to translate STL models (source models) into AutomationML models (target models).
AutomationML[2] is a family of standardized modeling languages designed to support the data exchange among heterogeneous engineering tools in the production system automation domain. For exchanging information about the hierarchical structure of production systems, AutomationML provides the CAEX modeling language.
Your task is to implement an ATL model transformation that transforms STL models into CAEX models. The goal of this ATL model transformation is to allow the exchange of information captured in STL models with engineering tools that support CAEX.
Your transformation has to implement the following strategies for mapping STL model elements (source models) to CAEX model elements (target models):

STL Concept	CAEX Concept
System	CAEX File with one Instance Hierarchy.
Area	Internal Element contained by the Instance Hierarchy created for the System.
Component	Internal Element with Attribute “cost”. The Internal Element has to be contained by the Internal Element created for of the Area that contains the Component.
Storage	Same mapping as for Component but with an additional “capacity” Attribute.
Slot	External Interface contained by the Internal Element created for the Component that contains the Slot.
Connector	Internal Link connecting the External Interfaces created for the connected Slots. The Internal Link has to be contained by the Internal Element created for the Area that contains the Connector.

Figure 1 shows an excerpt of the CAEX metamodel, which includes all metaclass relevant for the implementation of the ATL model transformation.

Additional Requirements

Besides implementing the mapping from STL to CAEX given above, your ATL model transformation has to meet the following requirements:

- Create at least one helper function
- Avoid imperative code (*do* section) as much as possible

1. Developing the ATL model transformation

Define the ATL model transformation in the file `STL2AML.atl` located in the project `at.ac.tuwien.big.stl.atl` in the folder `transformation`. This is the only file you need to change for implementing the ATL model transformation.

[2] https://www.automationml.org

Resources of `at.ac.tuwien.big.stl.atl`:

- **Folder `metamodels`**: Provides the metamodels of STL and CAEX (`stl.ecore` and `CAEX.ecore`). Graphical representations of the metamodels are provided in the diagram files `stl.aird` and `CAEX.aird`
- **Folder `models`:** Provides example STL models that should be processed by your ATL model transformation as input, and corresponding CAEX models that are expected to be produced as output of your ATL model transformation.
- **Folder `runner`:** Provides launch configurations for executing your ATL model transformation on the provided example STL models.
- **Folder `transformation`:** Provides the ATL file `STL2AML.atl` that you have to use for implementing the requested model transformation.

2. Testing the ATL model transformation

For testing your ATL model transformation, we provide the example STL models `Example1-IAFProductionLine.xmi` and `Example2-ShelfSawingProductionLine.xmi` (folder `models`). When executing your ATL model transformation for these example STL models, it should produce CAEX models corresponding to the provided CAEX models `Example1-IAFProductionLine_expected.xmi` and `Example2-ShelfSawingProductionLine_expected.xmi`.

To execute your ATL model transformation for the example STL models, use the launch configurations `STL2AML-Example1.launch` and `STL2AML-Example2.launch` (folder `runner`). These launch configurations will execute the ATL model transformation for the provided example STL models and produce the CAEX models `Example1-IAFProductionLine_transformed.xmi` and `Example2-ShelfSawingProductionLine_transformed.xmi`. Therefore, right-click on `*.launch` and select *Run As → STL2AML-ExampleX*.

Compare the produced CAEX models (`*_transformed.xmi`) with the expected CAEX models (`*_expected.xmi`). You can compare the models either manually by opening them in the tree-based editor or automatically using EMF Compare.

Before you open the models for manual inspection, you have to register the STL and CAEX metamodels once. Therefore, right-click on the corresponding `.ecore` files and select *EPackages registration → Register EPackages* into repository. Thereafter, you can open any STL or CAEX model by right-clicking on it and selecting *Open With → Other… → Sample Reflective Ecore Model Editor*.

To compare the models with EMF Compare, select both in the Model Explorer or Project Explorer, right-click on one of them and select *Compare With → Each Other*. The comparison result showing any differences between the models will then be opened automatically.

Note that the ordering of elements in the produced CAEX models is not strict. Thus, your ATL model transformation may produce a CAEX model where the ordering of elements is different than the ordering in the expected CAEX model. The correct containment hierarchy is, however, important.

Note that your ATL model transformation should be defined in such a way, that it works properly for all possible STL models, i.e., produce for any STL model a CAEX model following the mapping given above. To ensure this, additional testing from your side is required.

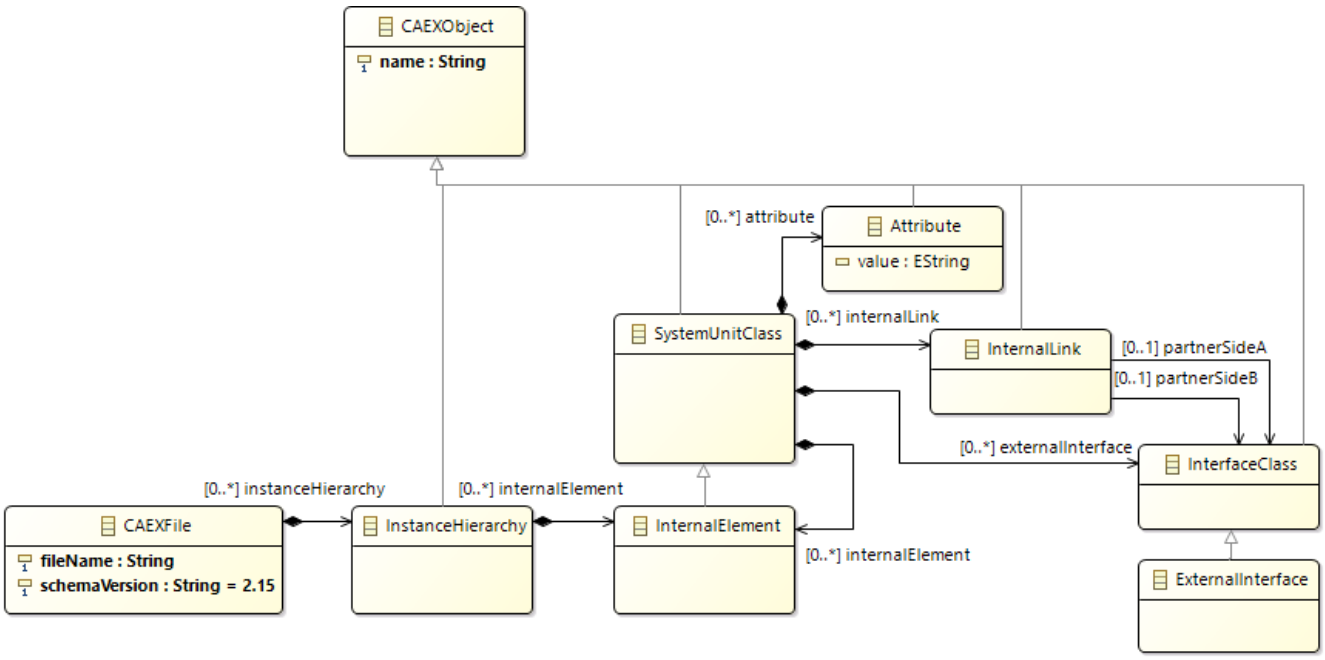


Figure 1: CAEX metamodel (excerpt required for Part A)

Part B: Henshin Transformation

In the second part of this assignment, you have to develop a Henshin transformation that implements a refactoring rule for STL models called "*addBufferToMachine*". This refactoring should insert buffers between machines and components that provide input to these machines.
For this, each input slot of each machine in an STL model has to be investigated. If a machine input slot is not already connected to an output slot of a buffer, then a new buffer with one input slot and one output slot should be created. The buffer's output slot has to be connected to the machine's input slot. The buffer's input slot has to be connected to the output slot that was previously connected to the investigated machine input slot. For this you need to modify existing connectors and create new connectors. Note that for the slots of a newly created buffer, the "requiredType" needs to be set correctly, namely to the required type of the respective machine input slot. Furthermore, a newly created buffer as well as a newly created connector have to be contained by the same area that also contains the connector that was previously connecting a (non-buffer) component with a machine.

Use the following naming scheme for newly created elements:

- Buffer: "GenBuffer"
- Input slot: "GenBuffer_IN"
- Output slot: "GenBuffer_OUT"

A single execution of this Henshin rule should create all missing buffers. Note that for a machine, all, some, or none of the input slots may be already connected to a buffer. Your Henshin transformation has to consider all possible cases.
Make sure that the STL models created by your Henshin transformations are valid, i.e., conform to the STL metamodel and fulfill all OCL constraints. To validate an STL model, open the model in the Sample Reflective Ecore Model Editor and selecting from the menu *Sample Reflective Editor → Validate*.

1. Developing the Henshin transformation

Define the Henshin transformation in the file `stl.henshin` located in the project `at.ac.tuwien.big.stl.henshin` in the folder `henshin`. You can use the file `stl.henshin_diagram` (located in the same folder) to define the transformation using the graphical Henshin editor. These are the only files you need to change for implementing the Henshin transformation.

HINT: Save your Henshin diagram often. If the graphical editor behaves strangely, do *not* press Undo/Ctrl-Z, but close the diagram window and re-open it and try to create the elements in a different way.

Resources of `at.ac.tuwien.big.stl.henshin`

- **Folder `henshin`:** Provides the Henshin files `stl.henshin` and `stl.henshin_diagram` that you have to use for implementing the requested Henshin transformation.
- **Folder `metamodels`:** Provides the metamodel of STL (`stl.ecore`).
- **Folder `models`:** Provides example STL models that should be processed by your Henshin transformation as input, and refactored versions of these models that are expected to be produced as output of your Henshin transformation.
- **Folder `runner`:** Provides the launch configuration `Henshin.launch` for executing your Henshin transformation on the provided example STL models.
- **Folder `src`:** Provides the executor class `HenshinRunner.java` that executes the Henshin transformation.

2. Testing the Henshin transformation

For testing your Henshin transformation, we provide the example STL models `Example1-IAFProductionLine.xmi` and `Example2-ShelfSawingProductionLine.xmi` (folder `models`). When executing your Henshin model transformation for these example STL models, it should produce STL models corresponding to the provided STL models `Example1-IAFProductionLine_expected.xmi` and `Example2-ShelfSawingProductionLine_expected.xmi`.

To execute your Henshin model transformation for the example STL models, use the launch configuration `Henshin.launch` (folder `runner`). This launch configurations will execute the Henshin transformation for the provided example STL models and produce the models `Example1-IAFProductionLine_transformed.xmi` and `Example2-ShelfSawingProductionLine_transformed.xmi`. Therefore, right-click on `Henshin.launch` and select *Run As → Henshin*. You will need to refresh the `models` folder to see the produced models. Therefore, right-click on the `models` folder and select *Refresh* (or press F5).

Compare the produced models (`*_transformed.xmi`) with the expected models (`*_expected.xmi`). You can compare the models either manually by opening them in the tree-based editor or automatically using EMF Compare. Follow the instructions given in Part A for comparing the models.

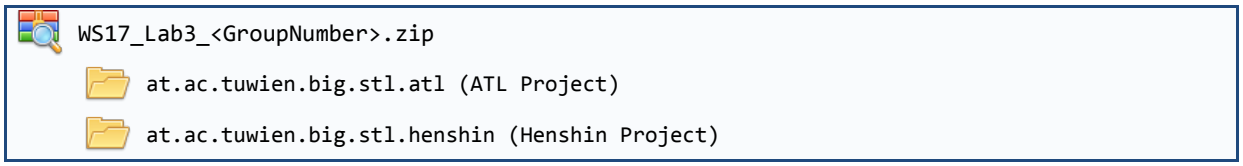
Additional Information

- Literature to solve this assignment is provided in the TUWEL course.
- You can find additional information, examples, and use cases for ATL at https://www.eclipse.org/atl/documentation/. All relevant ATL concepts for this assignment can be found in the ATL user guide.
- You can find additional information, examples and use cases for Henshin at https://www.eclipse.org/henshin/.

Submission & Assignment Review

Upload the following components in TUWEL:

You have to upload one archive file, which contains the following project:



For exporting these projects, select *File* → *Export* → *General/Archive File* and select the projects. Make sure that all folders and files contained by the projects are selected.

Assignment Review:

At the assignment review, you will have to present your ATL transformation and your Henshin transformation. You also have to show that you understand the theoretical concepts underlying the assignment.

All group members have to be present at the assignment review. The registration for the assignment review can be done in TUWEL. The assignment review consists of two parts:

- Submission and **group evaluation:** 20 out of 25 points can be reached.
- **Individual evaluation:** Every group member is interviewed and evaluated separately. The remaining 5 points can be reached. If a group member does not succeed in the individual evaluation, the points reached in the group evaluation are also revoked for this student, resulting in a negative grade for the entire course.