Model Engineering Lab ISE/FD - Information Systems Engineering Foundation 188.923 VU, WS 2020/21	**Assignment 3**
Deadline: Push solution to master branch until Monday, December 14th, 2020, 23:55 Assignment Review: Wednesday-Thursday, January 20th-21st , 2021	25 Points

It is recommended that you read the complete specification at least once before starting with your implementation. If there are any parts of the specification that are ambiguous to you, don't hesitate to ask in the TUWEL forum for clarification.

Lab Setting SensAction Company:

The SensAction Company is a young rising start-up company that wants to revolutionise the smart building market. Since the company consists of members of lots of different sectors, they have major communication difficulties, because everybody has an own way to describe the same things. In order to overcome this problem, they installed a new department and hired you and your colleagues. The purpose of your department is to design a new domain-specific language, the **Smart Building System Modeling Language (SBSML)**, describing smart building systems your company offers and providing tools that enable the other departments to work with the language.

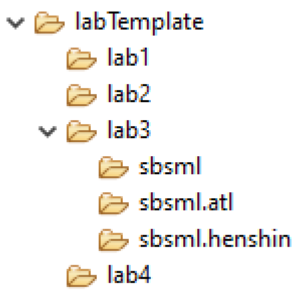
Lab 3: Model-to-Model Transformations

After you have established SBSML in your company you get aware of the CAEX[1] language which is an industry standard in the domain of your company. Since now all of your collegues are used to work with SBSML and this language really improved the business processes in your company the management decides to keep the SBSML language. However there should also be a mechanism to model smart systems with the CAEX industry standard.

As a model engineer, you chose to use model-to-model transformations to transform SBSML models into CAEX models. While thinking about these transformations, you remember about the possibilities of using in-model transformations to correct, revise, and optimise existing models.

Lab 3 Setup

Before starting with the assignment clone the lab3 template repository[2] and copy the content to the lab3 folder of your group's repository. We recommend creating a new Eclipse workspace for lab3 and import the projects from the lab3 folder.



[1] https://en.wikipedia.org/wiki/CAEX
[2] Lab 3 Template https://github.com/ModelEngineeringWS20/lab3Template.git

Part A: ATL Transformation

In the first part of this assignment, you have to develop an ATL model transformation to translate *SBSML* models (source models) into *AutomationML/CAEX* models (target models).

AutomationML[3] 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 SBSML models into CAEX models. The goal of this ATL model transformation is to allow the exchange of information captured in SBSML models with engineering tools that support CAEX. Your transformation has to implement the following strategies for mapping SBSML model elements (source models) into CAEX model elements (target models):

SBSML Concept	**CAEX Concept**
SmartSystem	*CAEXFile* and *InstanceHierarchy*. The (CAEX) *file name* and the *name* of the instance hierarchy should have the name of the *smart system*. The *instance hierarchy* should be contained by the *CAEXFile*.
Configuration	*InternalElement*. Contained in the *instance hierarchy* created for its *smart system*. The *name* of the *internal element* should be the *name* of the *configuration*.
Node	*InternalElement*. Contained in the *internal element* created for its *configuration*. The *name* of the *internal element* should be the *name* of the *node element* followed by the *name* of the referenced *thing* in parentheses (e.g. *BasilHS(HumiditySensor)*). Each *internal element* created for a *node* should contain further *internal elements* or *attributes* depending on the type of the *thing* referenced by the *node*. If the referenced *thing* of the *node* is a *fog device,* create an *attribute* named “mips” with the amount of *mips* of the referenced *fog device* as *value*.
Sensor Parameter	*InternalElement*. Contained in the *internal element* created for each *node* referencing a *sensor*. The *name* of the *internal element* should be the *name* of the sensed *parameter*. Additionally, create an *attribute* named “unit” with the *name* of the *parameter’s unit* as *value*.
Service	*InternalElement*. Contained in the *internal element* created for each *node* referencing an *actuator*. The *name* of the *internal element* should be the *name* of *service* together with all its *parameters* (in the schema “name: unit”) separated by a colon in parentheses (e.g. *setLight(brightness: Percent, timeout: Second)*). Additionally, if the *service* has a *description*, create an *attribute* within the *internal element* of the *service* named “description” and the *description* of the *service* as *value*.

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

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

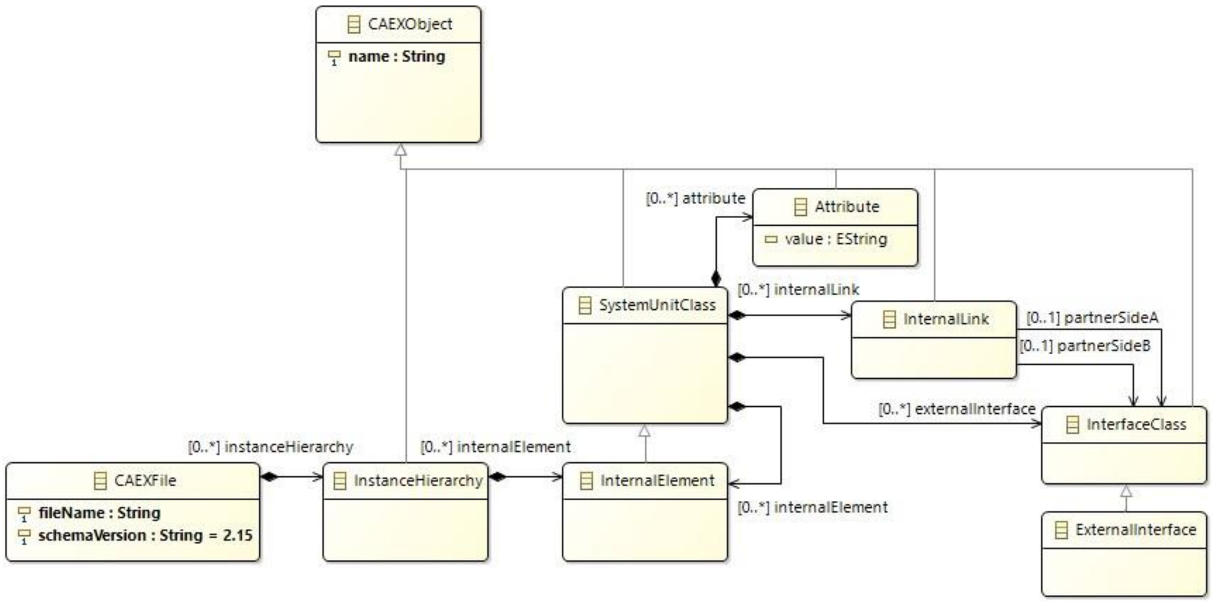


Figure 1: CAEX metamodel (excerpt required for Part A)

Additional Requirements

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

- Create at least **one helper function**
- Create at least **one abstract rule**

1. Developing the ATL model transformation

Define the ATL model transformation in the file `sbsml2caex.atl` located in the root folder of the project `sbsml.atl`. This is the only file you need to change for implementing the ATL model transformation.

Resources of `sbsml.atl`:

- **Folder input**: Provides example models which will be processed by your ATL model transformation.
- **Folder caex:** Provides the CAEX metamodel as `.ecore` file. A graphical representation is available via the `.aird` file.
- **Folder output:** Models transformed by your ATL model transformation are stored in this folder as `*–transformed.xmi`. This folder also provides CAEX models that are expected to be produced as output of your ATL model transformation (`*_expected.xmi`).
- **Folder runconfigs:** Provides launch configurations (`sbsml2caex_*.launch`) for executing your ATL model transformation on the provided example SBSML models.
- The ATL file `sbsml2caex.atl` that you have to use for implementing the requested model transformation can be found in the root folder of the project.

2. Testing the ATL model transformation

For testing your ATL model transformation, we provide the example SBSML models `indoor_farming.sbsml` and `smart_home.sbsml`. When executing your ATL model transformation for these example SBSML models, it should produce CAEX models corresponding to the provided CAEX models `indoor_farming_expected.xmi` and `smart_home_expected.xmi` in the folder `output`.

To execute your ATL model transformation for the example SBSML models, use the provided launch configurations:

- `Sbsml2caex_indoor_farming.launch`
- `Sbsml2caex_smart_home.launch`

These launch configurations will execute the ATL model transformation for the provided example SBSML models and produce the CAEX models `indoor_farming_transformed.xmi` and `smart_home_transformed.xmi`. To launch a transformation, right-click on `sbsml2caex_*.launch` and select *Run As → sbsml2caex_**.

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 by using the EMF Compare tool. To use EMF Compare, select two models in the workspace, right-click one of them and select *Compare With → Each Other*.

If you want to inspect the provided input models, you have to register the SBSML metamodel once. To do so, right-click on the `.ecore` file found in the `metamodel` folder and select *EPackages registration → Register EPackages into repository*. Now you can open any SBSML model by right-clicking on it and selecting *Open With → Sample Reflective Ecore Model Editor*.

Note that the correct containment hierarchy is important whereas the ordering of elements in the produced CAEX models is not. 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.

Note also, that your ATL model transformation should be defined in such a way, that it works properly for all possible SBSML models, i.e., produce for any SBSML model a CAEX model following the mapping given above. The CAEX models should also be valid, i.e. show no errors if you validate them. To validate a model, open the model in the Sample Reflective Ecore Model Editor and select *Sample Reflective Editor > Validate* from the context menu.

Part B: Henshin Transformation

In the second part of this assignment, you have to refactor SBSML models using Henshin transformations according to the following scenarios:

Scenario 1: Make single connection ports to multi connection ports (Rule: `SingleToMultiConnPort`, Example model: `scenario_1_singleToMultiConnPort.xmi`) If the *port* of one *node* is used multiple times for a *connection* and the *port* is defined as a single connection *port* set the *singleConnection* attribute to false.

Important: You only have to check the case where the *port* is used as *portA* within two *connections*. All other combinations (*portA-portB*, *portB-portB*) would work similarly and thus don't have to be implemented in this exercise.

Note: The example model for scenario 1 is OCL invalid. The transformation should fix the OCL violation and thus should be OCL valid.

Scenario 2: Remove unused Sensor nodes (Rule: `RemoveUnusedSensorNodes`, Example model: `scenario_2_unsedSensorNode.xmi`)

If a *sensor* is never used in a *threshold* the *sensor* can be removed from the *configuration*. In this example *connections* using the deleted *node* are not considered. In our example models *nodes* that should get deleted are never connected to any other *node*. The rule only has to work properly in such cases.

Scenario 3: Split workload (Unit: DuplicateFogNodeAndMoveController, Rules: `Duplicate1FogNode`, `moveController`, Input parameters: nodeName, duplicateName, controllerName, Example model: `scenario_3_sequence_dublicate1FogNodeAndMoveController.xmi`)

For the case that a *fog node* has not enough *computational power* to execute all its assigned *controllers* properly a mechanism to duplicate the *fog node* and move a *controller* should be provided. For this use a *sequential unit* to call the following 2 rules:

a) **Rule Duplicate1FogNode**: Duplicates the *node* with the name given by the input parameter nodeName. The duplicated *node* should be named as the input parameter duplicateName and reference the same *thing* as the original *node*. Additionally, the duplicated *node* should get connected to the original one using an existing *mutli connection port*. For this *connection*, the new *node* should be referenced as *nodeA* and the original *node* as *nodeB*.

b) **Rule MoveController**: Re-assigns the *computationNode* of a *controller* named as the input parameter controllerName to the *fog node* named as the input parameter nodeName.

Note: `scenario_3a` and `scenario_3b` should help to test the subrules.

Note: The example models for scenario 3 are OCL invalid. Applying both transformations with the sequential unit should fix the OCL violation and thus turn them into valid models.

Note: The example model for 3a is equal to the example model for scenario 3. The example model for 3b is equal to the expected outcome of 3a. The expected outcome of 3b is equal to the expected result of scenario 3.

Additional information

Make sure that the SBSML models created by your Henshin transformations are valid, i.e., conform to the SBSML metamodel and fulfill all OCL constraints. To validate an SBSML model, open the model in the Sample Reflective Ecore Model Editor and select *Sample Reflective Editor > Validate* from the context menu.

1. Developing the Henshin transformation

Define the Henshin transformations according to the 3 scenarios described on the previous page.

Implement them in the file `sbsml.henshin` located in the project `sbsml.henshin` in the folder `henshin`. You can also use the file `sbsml.henshin_diagram` (located in the same folder) to define the transformation rules 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 `sbsml.henshin`

- **Folder henshin:** Provides the Henshin files `sbsml.henshin` and `sbsml.henshin_diagram` that you have to use for implementing the requested Henshin transformation.
- **Folder models:** Provides 3 example SBSML models that should be processed by your Henshin transformations as input (`*_invalid.xmi`). It also provides refactored versions of these models that are expected to be produced as output of your Henshin transformations (`*_expected.xmi`).
- **Folder runconfigs:** Provides a launch configuration (`*_example.launch`) for each example model to execute your Henshin transformations on the provided example SBSML models.2. Testing the Henshin transformation

For testing your Henshin transformations, we provide the example SBSML models for each rule and the sequential unit to be defined in the folder `models`.

When executing your Henshin model transformation for these example SBSML models, it should produce SBSML models corresponding to the provided SBSML models `*_expected.xmi`

Hint: Don't forget to register the sbsml.ecore metamodel inside the models folder in the sbsml project.

Run transformations

To execute the Henshin model transformations on the provided example SBSML models, use the launch configurations `*_example.launch` that are located in the folder `runconfigs`.

These launch configurations will execute the Henshin transformation for the provided example SBSML models and produce the transformed SBSML models with the suffix *`_transformed.xmi` in the folder models.

To run a transformation on a certain model, right-click on the respective `*_example.launch` file and select *Run As → ...Example*.

You will need to refresh the `models` folder to see the produced models by selecting the folder in the explorer and press F5 or by right-clicking the `models` folder and selecting *Refresh* in the context menu.

Alternatively, you can also execute transformations by right-clicking a rule in the `sbsml.henshin_diagram` file and selecting “Apply Transformation”.

Compare transformed models

Compare your 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 by using the EMF Compare tool. To use EMF Compare, select two models in the workspace, right-click one of them and select *Compare With → Each Other*.

Submission & Assignment Review

At the assignment review you will have to present your solution, particularly your ATL model-2-model transformations, as well as your Henshin rules in the diagram. You also must show that you understand the theoretical concepts underlying the assignment.

Push all used projects containing the following components into your Github Classroom repository:

- *sbsml*
- *sbsml.atl*
  - edited sbsml2caex.atl
  - transformed models in the output folder
- *sbsml.henshin*
  - edited henshin/henshin_diagram files
  - transformed models in the models folder

Make sure that your solution is pushed to the master branch of your repository. Additionally, tag your final commit with the label '**lab3_solution**' (Command: *git tag lab3_solution*).

Distribution of Points:

- Part A: 10 points
  - ATL transformations: 10 points
- Part B: 10 points
  - Scenario 1: 3 points
  - Scenario 2: 3 points
  - Scenario 3: 4 points

All group members must be present at the assignment review. The registration for the assignment review can be done in TUWEL. The assignment review is divided into 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 five 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, which results in a negative grade for the entire course.