Model Engineering Lab ISE/FD - Information Systems Engineering Foundation 188.923 VU, WS 2020/21	**Assignment 4**
Deadline: Push solution to master branch until Monday, January 18th, 2021, 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:

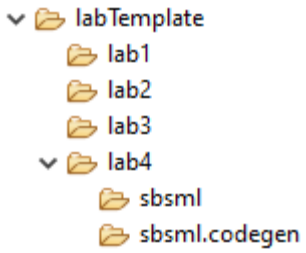
The SensAction Company is a young rising start-up company that wants to revolutionise the smart building market. Since the company consists of members of lots of different sectors, they have major communication difficulties, because everybody has an own way to describe the same things. In order to overcome 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 4: Code Generation

For the last task you have to provide a code-generation tool that transforms a SBSML model into a small Java application that simulates and thereby demonstrates the behaviour of the designed smart building system.

Lab 4 Setup

Before starting with the assignment clone the lab4 template repository[1] and copy the content to your lab4 folder of your group's repository. We recommend creating a new Eclipse workspace for lab4. There you can import the projects from the lab4 folder.



Our simulator code uses ANSI escape codes to change the color of some console outputs. Unfortunately the Eclipse console cannot display these codes correctly. So, you have to install the ANSI Console plugin[2] for Eclipse. To install the plugin in Eclipse select *Help -> Install New Software… and click "Add…"* to add the following URL: http://www.mihai-nita.net/eclipse

Important: Set your workspace default encoding to UTF-8 before importing the template projects. For this you can go to *Window → Preferences → General → Workspace → Text file encoding → Other: UTF-8*

[1] Lab 4 Template: https://github.com/ModelEngineeringWS20/lab4Template.git
[2] ANSI Console Plugin: http://mihai-nita.net/2013/06/03/eclipse-plugin-ansi-in-console/

Generating code to simulate an SBSML solution

In this assignment, you have to develop an Xtend code generator that generates Java code from SBSML models. The generated code is used to simulate the behavior of a smart system. The central components of the simulator are the *Thing* objects created for each *Node* and the *Controllers*. *Sensor* objects read sensor values from a data source, which are further processed by the *Controllers*. In the generated *Controller* code *Thresholds* are validated and *Services* of the *Actuator* objects are invoked.

1. Developing the Xtend code generators

For developing your Xtend code generator, we provide the basic structure in the Xtend project *sbsml.codegen*. The code generation for the various model elements has to be implemented in the corresponding `.xtend` files. These are the only files that you need to change for implementing the Xtend code generator. Write your code in the provided dispatch generate(...) methods, but feel free to create and use helper functions to be able to generalize your code.

Hint: The file UnitGenerator.xtend is already given, you do not need to change it!

Develop the code generator by examining the example models provided in the folder models (open them with the Sample Reflective Ecore Model Editor). The code that is expected to be generated for these models is located in the folder `src-gen-expected`. Identify which parts of the code depend on the model, i.e., have to be computed from the model, and which parts are independent of the model, i.e., static, and define your code generation templates accordingly.

Please make sure to implement the generation of the required Java files in `SbsmlGenerator.xtend` by using the IFileSystemAccess2[3].

To summarize: You have to implement all code sections in the `.xtend` files marked with `#TODO`. To doublecheck that you are not missing anything, you can use the task view in Eclipse by opening *Window* → *Show View* → *Tasks* to show all TODOs.

Hint: Some information on the behavior of the generated code for the different SBSML concepts can be found in the **readme.md** file.

Resources of sbsml.codegen:

- Folder `src`
  - `package at.ac.tuwien.big.sbsml.codegen`: Contains the MWE2 workflow `CodeGenerator.mwe2` that is used to run the code generation.
  - `package at.ac.tuwien.big.sbsml.codegen.lib`: Provides classes that support the code generation.
  - `package at.ac.tuwien.big.sbsml.codegen.xtend`: Provides the Xtend classes you have to complete for implementing the code generator.
- Folder `xtend-gen`: Contains the Java classes resulting from the compilation of the Xtend classes.
- Folder `src-gen`: When running the workflow `CodeGenerator.mwe2`, the code generated by your code generator classes will be stored in this folder.

[3] See http://download.eclipse.org/modeling/tmf/xtext/javadoc/2.9/org/eclipse/xtext/generator/IFileSystemAccess2.html

- Folder `src-gen-expected`: Contains the code that should be produced by your code generator for the example models provided in the folder models
- Folder `models`: Contains the example SBSML models that you can use for testing your code generator. The code that should be generated for these models is provided in the folder `src-gen-expected`.
- Folder `runconfigs`: Provides launch configurations (`_*.launch`) for running simulations.
- Folder `scenarios`: Provides scenario json files for running simulations.

2. Generating code with the Xtend code generators

To run your code generator, you have to run the MWE2 workflow `CodeGenerator.mwe2`. This workflow will execute the generator classes for the SBSML models located in the folder `models`. To run the workflow, right-click on the workflow file `CodeGenerator.mwe2` and select *Run As → MWE2 Workflow*. This will start the code generation. Check the output in the console to see whether the generation was successful. The code generated for the input models will be stored in the folder `src-gen`.

Hint: You have to re-run the workflow after each change you make in one of your code generator classes to update the generated code.

3. Run generated simulator code

Before running the Simulation you have to implement the two methods of the `at.ac.tuwien.big.sbsml.codegen.SmartSystemRunnerProvider` by returning a new instance of the appropriate runner.

To run your generated code, use the launch configurations provided in the `runconfigs` directory of the code generation project:

- `indoor_farming_simulator.launch`
- `indoor_farming_stepwise_simulator.launch`
- `smart_home_simulator.launch`
- `smart_home_stepwise_simulator.launch`

The launch files run the `SmartSystemSimulator.java` file. These expect a runner argument and a scenario file argument. The scenario files are located in the `scenario` folder. If you want to define your own scenarios you can either edit the json files or create new ones. The stepwise launch configurations additionally pass the "-s" flag, which starts the stepwise simulation. A stepwise simulation supports the following commands:

- <x>: Simulates the next <x> steps, where <x> is a number
- run: Simulates until the end of the scenario
- exit: Stops the simulation
- <Enter >: if you just hit enter without any command the next simulation step is executed

Hint: To see the expected behavior of a simulation you can copy the content of the `src-gen-expected` folder into the `src-gen` folder.

We provided the expected console log from the `indoor_farming_simulator.launch` in the `IndoorFarmingScenarioLog.pdf`.

Compare the code generated by your Xtend code generators for the provided example models (i.e., the code produced in the folder `src-gen`) with the expected code provided in the folder `src-gen-expected`. Besides differences in the code layouting (i.e., different line breaks, line indentations, white spaces, etc.), you have to produce the same code output as provided in `src-gen-expected`.

Submission & Assignment Review

At the assignment review you will have to present your solution. You also must show that you understand the theoretical concepts underlying the assignment.

Push the following components into your Github Classroom repository:

- *sbsml*
- *sbsml.codegen*
    - edited *xtend files*

Make sure that your solution is pushed to the master branch of your repository. Additionally, tag your final commit with the label '**lab4_solution**' (Command: *git tag lab4_solution*).

Distribution of Points:

- File generation: 3 points
- Generation of *Runner* code: 2 points
- Generation of *Configuration* code: 4 points
- Generation of *Thing* code: 6 points
- Generation of *Controller* code: 5 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.