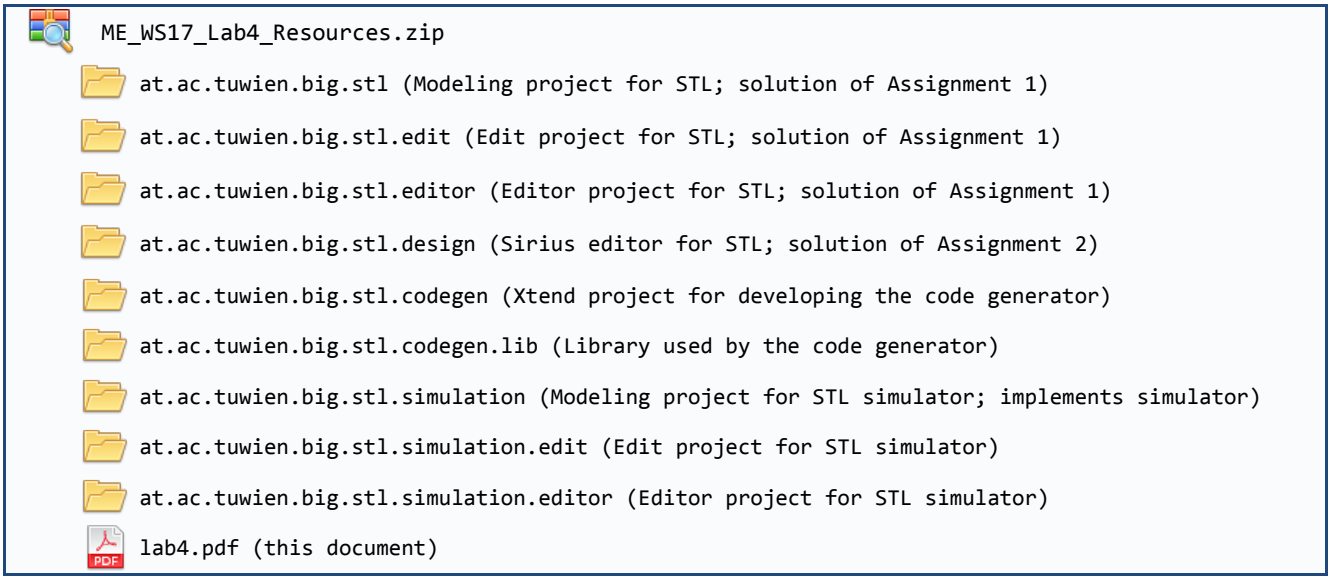
Model Engineering Lab 188.923 IT/ME VU, WS 2017/18	**Assignment 4**
Deadline: Upload (ZIP) in TUWEL until Sunday, January 14th, 2017, 23:55 Assignment Review: Wednesday, January 17th, 2018	25 points

Code Generation

The goal of this assignment is to develop a code generator for the *Simple Transportation Line Modeling Language* (STL) using Xtend. The code generator generates HTML code from an STL model that implements a Web-based control panel for the modeled transportation line. This control panel displays the status of the transportation line, such as the utilization of the different components, the number of produced products, etc. Thereby, the displayed data is computed by a simulator written in Java. Besides generating the HTML code for the control panel, you will also generate Java classes starting the simulation of STL models.
In Part A of the assignment, you will develop the Xtend code generator. In Part B, you will implement some parts of the simulator.

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_Lab4_Resources.zip* and import all 9 project listed above.

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: Code Generator

In the first part of this assignment, you have to develop an Xtend code generator for STL that generates HTML code and Java code for STL models.

The generated code shall implement a Web-based control panel that displays the status of a modeled transportation line. The displayed data is computed by a simulation of the transportation line. The generated code should work as shown in Figure 1: A Server receives as input an STL simulation model (*.stlsimulation) that contains configuration data for simulating an STL model (*.stl). The metamodel for STL simulation models is shown in Figure 2 and described below. The Server hands this STL simulation model over to a WebServer and starts the WebServer. The WebServer subsequently starts the simulation of the STL model, which is carried out by a Simulator. The Simulator notifies the WebServer about changes in the state of the simulated transportation line, such as the creation of items by item generators, the processing of items by machines, etc. When receiving a state update notification, the WebServer updates the HTML-based control panel (HTML UI), which displays the state of the transportation line. The control panel comprises an overview (`index.html`) of all components of the input transportation line and their state (e.g., utilization), and details pages for all components (`details_*.html`) that show further state information (e.g., error rate, average utilization). When the simulation terminates, the WebServer stores the simulation result in a new STL simulation model (STL Simulation Model [result]).

Your task is to generate the Server class and HTML UI for STL models (highlighted in gray color in Figure 1). Thereby, the Server class is generated from an STL simulation model (`*.stlsimulation`), and the HTML UI is directly generated from an STL model (`*.stl`).

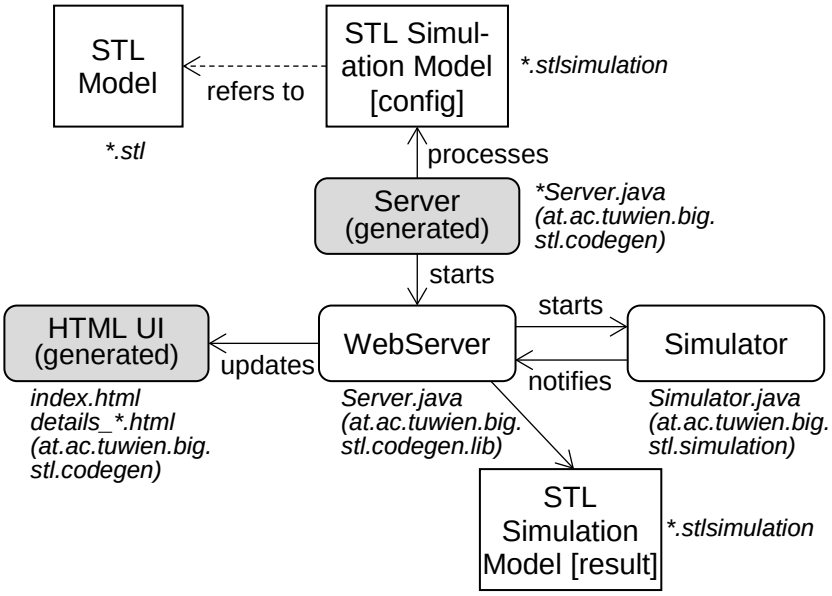


Figure 1: Overview of generated code

STL simulation models

Figure 2 shows the metamodel for STL simulation models. Such a model (metaclass `SimulationModel`) consists of a simulation configuration (metaclass `SimConfiguration`) specifying parameters of the simulation, such as the duration of the simulation (attribute `simulationTime`), as well as information about the state of a simulated STL model, such as the number of produced items

(attribute `itemThroughput`). Furthermore, it contains state information about all components and connectors of a modeled transportation line (classes `ComponentInfo` and `ConnectorInfo`), as well as of items produced and processed by the transportation line (class `ItemInfo`).

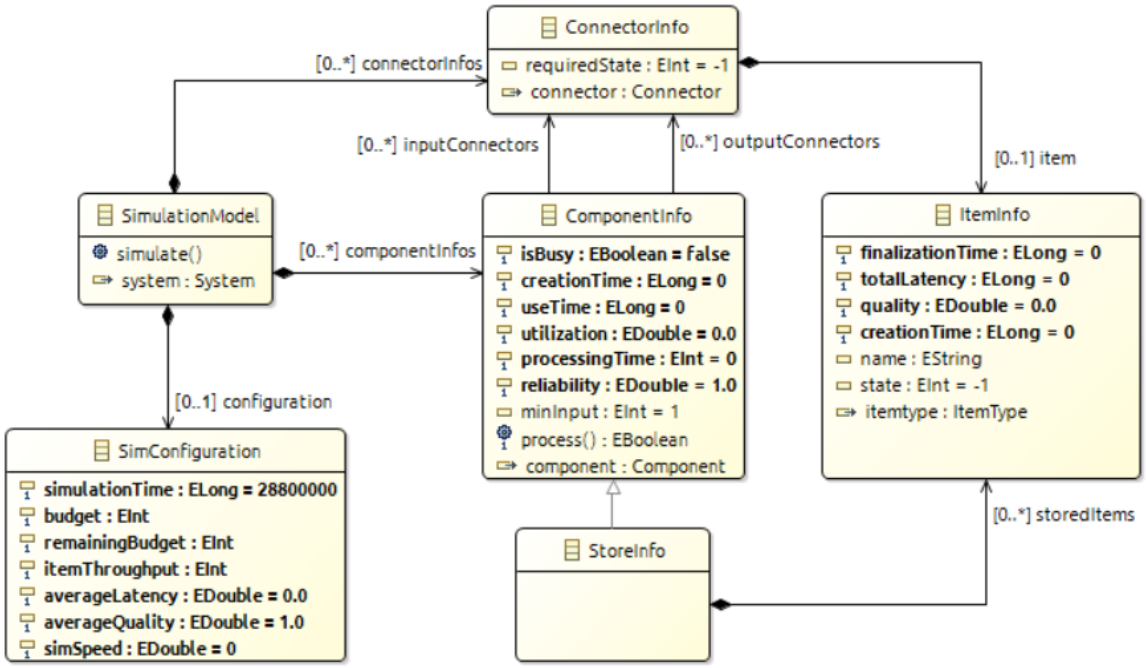


Figure 2: Metamodel for STL simulation models (provided in `at.ac.tuwien.big.stl.simulation/model/simulation.ecore`)

1. Developing the Xtend code generators

For developing your Xtend code generator, we provide the Xtend project `at.ac.tuwien.big.stl.codegen`. The code generation for the Server has to be implemented in the class `ServerGenerator.xtend` located in `src/at.ac.tuwien.big.stl.codegen`. The code generation of the HTML code has to be implemented in the class `HTMLGenerator.xtend` located in the same package. These two files are the only files that you need to change for implementing the Xtend code generator.

Develop the code generator by examining the example models provided in the folder `models` and the code that is expected to be generated for these models, which 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.

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

- **Folder `src`**: Provides the Xtend classes `ServerGenerator.xtend` and `HTMLGenerator.xtend` that you have to complete for implementing the requested code generator. Furthermore, it contains the MWE2 workflow `CodeGenerator.mwe2` and associated workflow components that you can use for executing your code generator.
- **Folder `xtend-gen`:** Contains the Java classes resulting from the compilation of the Xtend classes `ServerGenerator.xtend` and `HTMLGenerator.xtend`.
- **Folder `src-gen`:** When running the workflow `CodeGenerator.mwe2`, the code generated by

your code generator classes `ServerGenerator.xtend` and `HTMLGenerator.xtend` will be stored in this folder.

- **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 `execution`:** The result of a simulation of an STL model will be stored in this folder as a `*.stlsimulation` file.
- **Folder `html`:** Contains static resources (CSS, images, and JavaScript code) required by the HTML code that you have to generate.
- **Folder `models`:** Contains example STL models and STL simulation 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`.

2. Generating code with the Xtend code generators

For running your code generator, i.e., the classes `ServerGenerator.xtend` and `HTMLGenerator.xtend`, you have to run the MWE2 workflow `CodeGenerator.mwe2`. This workflow will execute both of your generator classes for the STL models and STL simulation 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`.
Note that after each change in one of your code generator classes `ServerGenerator.xtend` or `HTMLGenerator.xtend`, you have to re-run the workflow to update the generated code.

3. Testing the generated code

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 as provided in `src-gen-expected`. The only exception is in the generated Server classes, where the path provided as input parameter to the called operation `ModelLoader.load(String)` will differ depending on location of the project `at.ac.tuwien.big.stl.codegen` in your file system.
For instance, for the example STL model `shelf-sawing-production-line.stl`, the HTML file `index.html` should be produced by your `HTMLGenerator.xtend` code generator class in the folder `src-gen` in the package `shelfsawingproductionline.html`, as well as nine `details*_.html` files – one for each component defined in `shelf-sawing-production-line.stl`. They should contain the same HTML code as the corresponding HTML files provided in the folder `src-gen-expected`. For the example STL simulation model `shelf-sawing-production-line.stlsimulation`, the Java class `ShelfSawingProductionLineServer.java` should be produced by your `ServerGenerator.xtend` code generator class in the folder `src-gen` in the package `shelfsawingproductionline.server`. This Java class should contain the same Java code as the corresponding Java class provided in the folder `src-gen-expected` (besides the above mentioned difference in the path to the `*.stlsimulation` model).

Note that your code generators have to be generic enough to process any kind of STL model and STL simulation model. Thus, be careful about which parts of the generated code are static, i.e., independent of the input model, and which parts are dynamic, i.e., depend on the input model.

Also open the generated `index.html` files in your browser, and compare the displayed user interface with the expected one. Visit the details pages of the different components and again compare them with the expected ones.

Part B: Simulator

When you open the `index.html` files generated for the provided example models, you will notice that the displayed state information does not change. This is because for starting the simulation, which will update the displayed state information, we first have to start the Server, which will in turn start the simulation. However, in this part of the assignment, you have to implement some missing parts of the simulator as explained in the following:

1. Implement model loader and model store

For starting the simulation of an STL model, the Server generated by your code generator class `ServerGenerator.xtend` needs to load the respective STL simulation model, which provides the simulation configuration for this STL model (e.g., simulation speed, simulation duration, etc.). Therefore, the operation `load(String)` of the class `ModelLoader.java` located in the project `at.ac.tuwien.big.stl.codegen.lib` (folder `src`, package `at.ac.tuwien.big.stl.codegen.lib.util`) is called. This operation has to be implemented by you.

When the simulation terminates, the WebServer should store the simulation result, which is captured in a new STL simulation model, in the folder `execution` of the project `at.ac.tuwien.big.stl.codegen`. For this, the WebServer calls the operation `store(SimulationModel, String)` of the class `ModelStore.java` located again in the project `at.ac.tuwien.big.stl.codegen.lib` (folder `src`, package `at.ac.tuwien.big.stl.codegen.lib.util`). You can see this operation call in line 420 of the class `Server.java` (project `at.ac.tuwien.big.stl.codegen.lib`, folder `src`, package `at.ac.tuwien.big.stl.codegen.lib.server`). Again, the operation `ModelStore.store(SimulationModel, String)` has to be implemented by you.

Implement the operations `ModelLoader.load(String)` and `ModelStore.store(SimulationModel, String)`. `ModelLoader.load(String)` should load the STL simulation model (`*.stlsimulation`) located at the path provides as input parameter. `ModelStore.store(SimulationModel, String)` should store the STL simulation model provided as first parameter at the location specified by the path provided as second parameter.

HINT: Information on how to load and store EMF models can be found in the slides of the M3 lecture unit. In particular, have a look at slide 53 “Loading Models” and slide 54 “Saving Models”.

Test your implementations using the test cases provided by the test classes `ModelLoaderTest.java` and `ModelStoreTest.java` provided also in the project `at.ac.tuwien.big.stl.codegen.lib` in the folder `test`.
For executing these test cases, you can use the provided launch configurations `ModelLoaderTest.launch` and `ModelStoreTest.launch` located in the same project. For this, right-click on them and select *Run As* → **Test*. Your implementations are correct, if these test cases pass without errors and assertion violations.

2. Implement simulation of waste stores

The simulation of an STL model is carried out by the STL simulator implemented by the project `at.ac.tuwien.big.stl.simulation`. This simulator is complete, except for the simulation of waste stores, which has to be implemented by you. For this, implement the operation `process(WasteStore)` of the class `StoreInfoImpl.java` located in the folder `src-gen` in the package `at.ac.tuwien.big.stl.simulation.impl`. This operation performs simulation steps for waste store components of STL models as described in the following: If an input connector of the waste store (`ComponentInfo.inputConnector`) passed as input parameter provides an item (`ConnectorInfo.item`) and the capacity of the waste store (`WasteStore.capacity`) is not exceeded yet, the provided item is added to the waste store (`StoreInfo.storedItems`). Furthermore, the usage time of the waste store (`ComponentInfo.useTime`) is updated, i.e., it is increased by the waste store's processing time (`ComponentInfo.processingTime`). Note that these actions are performed for all input connectors of the waste store (in case a waste store has multiple input connectors) in a single simulation step.

Test your implementations using the test cases provided by the test class `WasteStoreSimulationTest.java` provided also in the project `at.ac.tuwien.big.stl.simulation` in the folder `test`.
For executing these test cases, you can use the provided launch configuration `WasteStoreSimulationTest.launch`. For this, right-click on the launch configuration and select *Run As → WasteStoreSimulationTest*. Your implementation is correct, if the test cases pass without errors and assertion violations.

3. Test the simulation

Once your implementations pass the provided test cases, you can test the simulations of the example STL models. To test the simulation of an example STL model, start the Server class generated for this example (e.g., `ShelfSawingProductionLineServer.java`) by right-clicking on it and selecting *Run As → Java Application*. Then, open http://localhost:8888 in your browser. The `index.html` page generated for the STL model should then be displayed. Start the simulation by hitting the dark-blue “Start” button provided in the top-right of the rendered `index.htm`. The state information displayed in the page should now be updated reflecting the simulation data. Check the console in Eclipse for any errors. Note that the simulation data is also updated live in the generated details pages. After the simulation finished, the simulation result should be stored by the implemented `ModelStore.java` class in a new `*.stlsimulation` file in the project `at.ac.tuwien.big.stl.codegen` in the folder `execution`. Check whether this file has been created for the simulation run.
Note that if you want to start another simulation, you first have to terminate the running one. For this, press the “Terminate” button in the Console view of your Eclipse (red rectangle on the top-right of this view). If the Console view is not visible, you can open it by selecting *Window → Show View → Other… → General / Console*.

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 Xtend code generators, as well as your implementations for the model loader, model store, and waste store simulation. 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.