Model Engineering Lab ISE/FD - Information Systems Engineering Foundation 188.923 VU, WS 2020/21	**Assignment 2**
Deadline: Push solution to master branch until Monday, November 23th, 2020, 23:55 Assignment Review: Wednesday-Thursday, November 25th-26th , 2020	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:

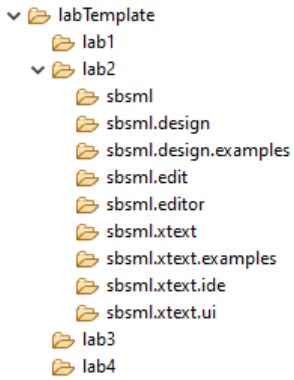
The SensAction Company is a young rising start-up company that wants to revolutionise the smart building market. Since the company consists of members of lots of different sectors, they have major communication difficulties, because everybody has an own way to describe the same things. In order to overcome 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 2: Concrete Syntax

Your second task is to provide the tools to create and present models in your language. The technical department prefers *text-based tools*, while the sales agents want a *visual tool* such that also the customers can understand the model more intuitively.

Lab 2 Setup

Before starting with the assignment clone the lab2 template repository [1]and copy the content to your lab2 folder of your group's repository. We recommend creating a new Eclipse workspace for lab2. There you can import the projects from the lab2 folder.



Part A: Textual Concrete Syntax

In the first part of this assignment, you have to develop the textual concrete syntax (grammar) for SBSML with Xtext. Additionally, you have to implement scoping support for the textual editor that is generated from the developed grammar.

The grammar has to be built upon the sample solution of the SBSML metamodel provided in the assignment resources (*sbsml/model/sbsml.ecore*).

Important: Do **NOT** use the metamodel you have developed in Assignment 1 but the sample solution provided in the assignment resources! Make yourself familiar with the sample solution metamodel before starting with this exercise.

[1] Lab 2 Template https://github.com/ModelEngineeringWS20/lab2Template.git

A.1 Xtext Grammar for SBSML

Develop an Xtext grammar that covers all modeling concepts provided by SBSML.

You can find the SmartHome example from Assignment 1 as an example textual model in a separate pdf file (*smart_home_textual_model.pdf*) as well as in the assignment resources (*sbsml.xtext.examples/smart_home.sbs*).

To implement the Xtext grammar for SBSML, perform the following steps:

1. Setting up your workspace

As mentioned in the beginning, the Xtext grammar has to be based on the sample solution of the SBSML metamodel. We provide this sample solution as well as skeleton projects for the Xtext grammar as importable Eclipse projects.

Import projects
To import the provided project in Eclipse select *File → Import → General/Existing Projects into Workspace → Select* the root folder of the assignment resources and import the following projects:

1. *sbsml*
2. *sbsml.edit*
3. *sbsml.editor*
4. *sbsml.xtext*
5. *sbsml.xtext.ide*
6. *sbsml.xtext.ui*

Register the metamodel
Should the Xtext grammar file *Sbsml.xtext* (project *sbsml.xtext*) show errors just register the SBSML metamodel in your Eclipse instance, such that the Xtext editor with which you develop your grammar knows which metamodel you want to use. You can register your metamodel by right-clicking on *sbsml/model/sbsml.ecore → Epackages registration → Registre Epackages into repository*. Clean your projects afterwards by selecting *Project → Clean... Clean all projects.*

After performing these steps, you should have six projects in your workspace without any errors (there may be warnings that you can ignore).

2. Developing your Xtext grammar

Define the Xtext grammar in the file *Sbsml.xtext* located in the project *sbsml.xtex*t in the package src/at.ac.tuwien.big. Documentation about how to use Xtext can be found in the lecture slides[2], in the Xtext documentation[3] and in the tutorial videos[4].

Hint: You do not need to enforce formatting like indentation or new lines.

Hint: If you use special characters like ':', '(' or ')' put them in separate literals. Like this you ensure that it does not matter if e.g., parentheses follow in the same line or in a new.

[2] Xtext lecture slides https://tuwel.tuwien.ac.at/mod/resource/view.php?id=974110
[3] Xtext documentation https://www.eclipse.org/Xtext/documentation/
[4] Lab 2 tutorials https://tuwel.tuwien.ac.at/mod/page/view.php?id=950743

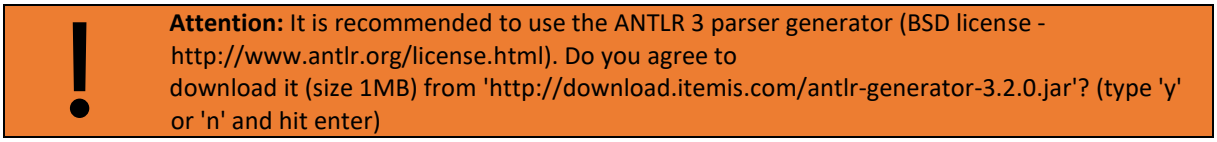
3. Testing your editors

Generate Xtext artifacts

After you have finished developing your Xtext grammar in *Sbsml.xtext*, you can automatically generate the textual editor. The Xtext grammar project *sbsml.xtext* contains a so called model workflow file *GenerateSbsml.mwe2*, which orchestrates the generation of the textual editor. By default, the workflow will generate all the necessary classes and stub classes in the grammar project and in the related ide-project and ui-project.

To run the generation, right-click on the workflow file *GenerateSbsml.mwe2* and select *Run As → MWE2 Workflow*. This will start the editor code generation. Check the output in the Console to see if the generation was successful. Please note that after each change in your grammar you have to re-run the workflow to update the generated editor code.

If you start the editor code generator for the first time the following message may appear in the console:



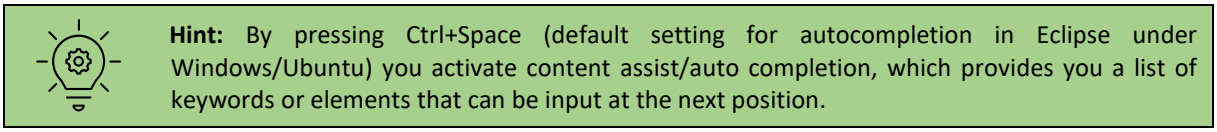
Type 'y' and download the jar file. It will be automatically integrated into your project.

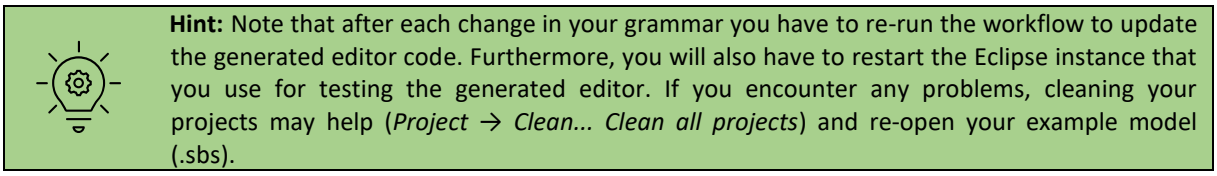
Start a new Eclipse instance with your plugins

For starting the generated editor, we have already provided a launch configuration located in the project *sbsml.xtext*, which is called "*ME Lab 2 Runtime.launch*". Run it by right-clicking on it and selecting *Run As → ME Lab 2 Runtime*. You can have a look at the configuration by right-clicking on that file and selecting *Run As → Run Configurations….*

Start modeling

In the newly started Eclipse instance, you can import the *sbsml.xtext.examples* and create a new file (*File → New → File*) with the extension *.sbs* in this project. If you are asked to add the Xtext nature to the project ('Do you want to convert 'sbsml.xtext.examples' to an Xtext project?') hit 'Yes'. Right-click on the created *.sbs* file and select *Open With → Sbsml Editor*. Now you can start modeling an SBSML model to test the generated textual SBSML editor.





Validate your solution

In the newly started Eclipse instance make sure that your Xtext editor for SBSML can process the following examples without problems, i.e. shows no errors or warnings when you open the models:

- *indoor_farming.sbs*
- *smart_home.sbs*

Also make sure that values for attributes and references are actually assigned to the model elements. For checking that, you need to convert your *.sbs* files into *.sbsml* files. This can be done by right-clicking a *.sbs* file and selecting *Open With → Other... → Sample Reflective Ecore Model Editor*. The selected file is now opened with a tree-based editor. Now select *File → Save As... (from the menu bar)* and save the model into a file with

the *.sbsml* extension. Investigate the elements these models in the tree-based editor of SBSML (right-click on a *.sbsml* model file and select *Open With → SBSML Model Editor*) and make sure that their attribute values and references are correctly set via the properties view (if the properties view is not shown right-click on any model element and select *Show Properties Views*).

Example model
Additionally, model the following system with your textual editor:

The "occupation" system contains a "lock" sensor which checks if a door is locked and a "display" actuator which displays information when calling the service "set" with the argument "doorState" of type BOOL. In the "WC" configuration the sensor and the actuator are connected to a RaspberryPI with 2 single connection ports. The "LockedController" calls the set service of the display with the argument true if the door is locked, and the "UnlockedController" calls it with the argument false, if the door is not locked.

A.2 Scoping Support for SBSML

In the grammar that you have developed, you should have defined some cross-references, e.g., a controller needs a computation node which has a thing of type fog device. When testing your editor, you will notice that whenever there is a cross-reference, the content assist (Default Eclipse setting Windows/Ubuntu: *CTRL+Space*) will provide all elements of the respective type that the editor can find. Which elements the editor provides is defined in the scoping of the respective reference. By default, the editor searches through the whole class-path of the project and you will notice that if you have many models in your project, a lot of elements will show up as possible reference.

To restrict this behavior, Xtext provides a stub where developers can define their own scope. The stub for the SBSML language can be found in the grammar project *sbsml.xtext* in the package *src/at.ac.tuwien.big.scoping (SbsmlScopeProvider.java)*. Your task is to implement the following:

1) Scoping for the Ports of Connections

The list of suggested ports for a connection should only include ports defined in the thing the node used in the connection, refers to.

E.g., in the configuration Basement of the *smart_home.sbs* model the suggested ports for the node BaseL should only be LB_ZIGBEE or LB_GPIO.

2) Scoping for the second Node of a Connection

After the first node together with its ports are defined within a connection the suggested counterpart node has to be a different node of the same configuration.

E.g., in the configuration Bathroom of the *smart_home.sbs* model the suggested counterpart node for a connection which defined BathV as first node should be the nodes BathPi, BathM and BathHS.

3) Scoping for the computation Node of a Controller

The node which is used for executing a controller needs to be part of the same configuration and thing it refers to needs to be of type fog device.

E.g., the BaseLightController of the *smart_home.sbs* model can only use the BasePi node as computation node.

4) Scoping for the source of a Threshold

The source node of a threshold needs to be part of the same configuration. Its Thing it refers to needs to be a sensor. Further it is required that the sensor has a parameter where the unit's data type matches the type of the threshold.

E.g., the BoolThreshold in the BaseLightController in the *smart_home.sbs* model should only have the BaseM node as source as its sensor contains the parameter isMoving of unit Moving which is defined as BOOL.

5) Scoping for the parameter of a Threshold

Only parameters of the source node should be suggested and the datatypes should also be taken into account.

E.g., the IntThreshold of the BaseGasLeakController in the *smart_home.sbs* model should only suggest the concentration parameter as the sensor-thing of the already defined source node BaseGD only contains the correct parameter concentration (that unit is of type INT).

Hint: All code parts that you should implement is marked with a TODO comment in the *SbsmlScopeProvier.java* file.

Validate your solution

After you have implemented the scoping as defined above start again a new Eclipse instance (use the provided launch configuration "*ME Lab2 Runtime.launch*") and test your scoping with the content assist (*Ctrl+Space*).

We already provided three example models in the project *sbsml.xtext.examples* that you can use for checking the scoping. Remove the [?] placeholders and check for each model if the correct suggestions are provided:

- *scoping/scoping_connections.sbs* (for the scoping requirements 1 and 2)
- *scoping/scoping_controller_computationnode.sbs* (for the scoping requirement 3)
- *scoping/scoping_threshold.sbs* (for the scoping requirements 4 and 5)

Part B: Graphical Concrete Syntax

In the second part of this assignment, you have to develop a graphical concrete syntax and a graphical editor for SBSML with Sirius. Like the textual concrete syntax, also the graphical concrete syntax has to be built upon the sample solution of the SBSML metamodel provided in the assignment resources. A graphical concrete syntax maps concepts from the metamodel to a graphical representation. For example, the configuration class is represented as rectangle with a blue dashed border. Your graphical syntax should be able to give a complete overview of a smart system containing all things, units, and configurations. Additionally, while installing the system the on-site technicians need a view of the network topology where they can see how to connect the nodes. They only need a focused view, where nodes, ports and connections of the configurations are shown. For this, you have to define two different viewpoints, one *System Viewpoint* (Figure 1) and one *Network Viewpoint* (Figure 2).

Sirius Diagram Editor for SBSML

Develop a graphical concrete syntax and diagram editor that allows to display and edit SBSML models. For this, develop *diagram mappings* and *element creation tools* as described in the following.

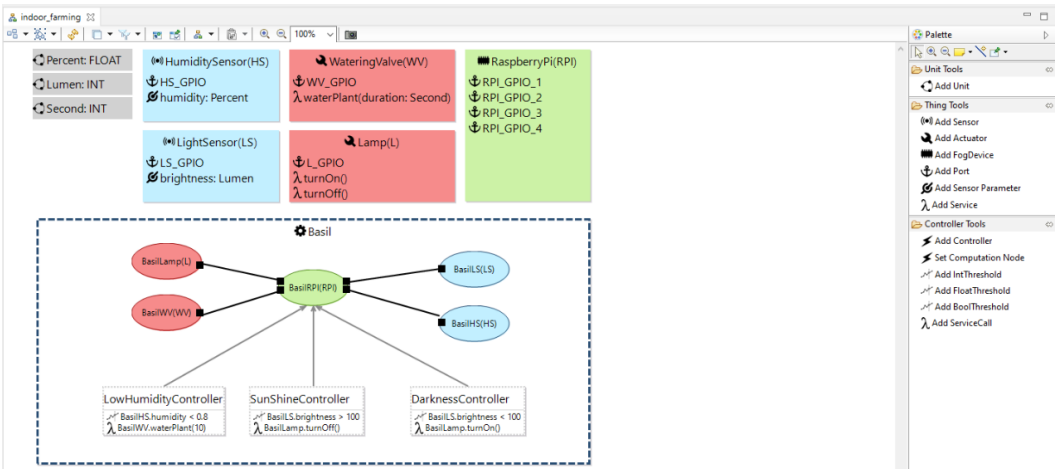


Figure 1: Graphical Concrete Syntax of the indoor farming model (System Viewpoint)

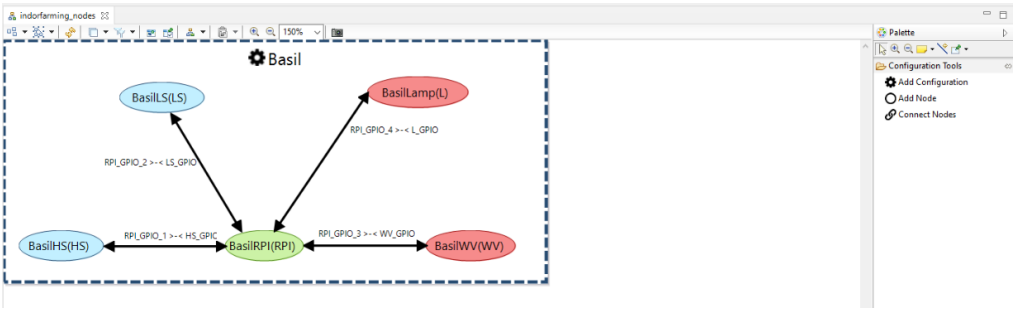


Figure 2: Graphical Concrete Syntax of the indoor farming model (Network Viewpoint)

Mappings

In the provided description file, the mapping for the concept "Sensor" in the “System Viewpoint” is already given. The “Network Viewpoint” is empty. For displaying whole SBSML models as diagrams, develop diagram container and node mappings as well as diagram edge mappings for the following SBSML concepts:

System Viewpoint:

- Unit
- Actuator
  - Port
  - Service
- Fog Device
  - Port
- Configuration
  - Node

        - Ports(bordered nodes)
    - Connection (edge mapping)
    - Controller
        - Threshold
        - Service Call
        - Controller.computationNode(edge mapping)

Network Viewpoint:

- Configuration
    - Nodes
    - Connection(edge mapping)

The diagram editor has to be able to display the mentioned elements following the *graphical concrete syntax* that is illustrated in Figures 1 and 2. This example models are available in the project sbsml.design.examples*/models* in the model file *indoor_farming.sbsml*. A second example model *smart_home.sbsml* is provided, which displays each concept at least once. Additionally, the project holds screenshots of the two example model diagrams for comparison.

Hint: Conditional styles do not always work even if they are correct in theory. Restarting Eclipse and regenerating the graphical representations can sometimes help. If you have problems with the conditionals, choose a plausible default value and leave the conditionals in your mappings. If we see you understood the concept, no points will be deduced.

Creation Tools:

In the provided description file, the tool section “Controller Tools” as well as the “Add Service Tool” are already given.

For editing SBSML models, develop element creation tools for the following SBSML concepts (see also the tool palettes shown in Figures 1 and 2).

System Viewpoint:

- Unit Tools:
    - Add Unit: Creates an instance of unit.
- Thing Tools:
    - Add Sensor: Creates an instance of sensor.
    - Add Actuator: Creates an instance of actuator.
    - Add FogDevice: Creates an instance of fog device.
    - Add Port: Creates an instance of port and adds it to the thing it is added to in the editor.
    - Add SensorParameter: For creating an instance of service parameter, make use of a dialog which provides a *Text Widget* to type in a name for the parameter and a select widget to select a unit. The *Select Widget* should display all units defined in the smart system. The created sensor parameter should be added to the sensor.

Network Viewpoint:

- Configuration Tools:
    - Add Configuration: Creates an instance of configuration.
    - Add Node: Make use of a dialog which provides a Text Widget to give a name for the node and a *Select Widget* where the thing for the Node.thing reference can be selected. All things of the smart system should be selectable.
    - Connect Nodes: With the connect nodes tool you should be able to select two nodes which should be connected. It should not be possible to connect a node with itself or with a node of another configuration. After selecting the 2 nodes a dialog with one *Select Widget* per node

should be displayed, where the ports used for the connection can be set. Show all ports of the thing referenced by the nodeA relation in the first *Select Widget* and all ports of nodeB in the second.

Where possible set default values for attributes and references. For example, names of actuators or MIPS values for fog devices. You may assume values for those attributes.

Hint: The icons to be used are provided in the img/icon folder in the sbsml.design project.

Hint: When debugging your select widgets you can use the function *eContents()* to get all children of the container in the candidate expression (e.g. *self.eContainer().eContents()*). You can display them in the select widget by using *aql:candidate.toString()* in the candidate display expression.

To implement the graphical concrete syntax and graphical (diagram) editor for SBSML, perform the following steps:

1. Setting up your workspace
Import the following projects into the Runtime Eclipse Instance:

1. sbsml.design
   You will use this project, to develop the graphical concrete syntax and graphical editor of SBSML
2. sbsml.design.examples
   You will use this project to test your graphical editor.

Lastly, switch into the *Sirius* perspective by selecting *Window* → *Perspective* → *Open Perspective* → *Other…* → *Sirius*. This perspective provides the *Model Explorer* and *Interpreter* views, which are useful for developing Sirius viewpoint specification models.

2. Develop your graphical editor
Define the graphical concrete syntax of SBSML with Sirius by extending the viewpoint specification model *sbsml.odesign*, which is located in the project *sbsml.design* in the folder description. Documentation about how to use Sirius can be found in the lecture slides, the tutorial videos[5] and in the Sirius documentation[6],[7].

3. Validate your graphical editor
To test your graphical editor, open e.g., the diagram "indoor_farming_system" diagram defined in the file *representations.aird* located in the project sbsml.design.examples (you need to be in the *Sirius* perspective to be able to unfold the content of *representations.aird* in the *Model Explorer*). This diagram is mapped to the SBSML model defined in the file *indoor_faming.sbsml*. Thus, the diagram will be automatically updated with representations of model elements for which you have correctly defined node/edge mappings. Furthermore, the tool palette will show the element creation tools that you have correctly defined. Updates on your Sirius viewpoint specification model *sbsml.odesign* will be automatically reflected in the opened diagram. The same mapping applies to all six diagrams in the *representations.aird*:

- System Viewpoint
  - indoor_farming_system
  - smart_home_system
  - occupy_system
- Network Viewpoint
  - indoor_farming_network

[5] Lab 2 tutorials: https://tuwel.tuwien.ac.at/mod/page/view.php?id=950743
[6] Sirius documentation: http://www.eclipse.org/sirius/doc/
[7] Sirius tutorials: https://eclipse.org/sirius/getstarted.html

- smart_home_network
- occupy_network

For testing your element creation tools you should model the "occupy" system from Part A with your graphical editor tools. For this use the "*occupy_system*" and "*occupy_network*" diagrams and model the system with the element creation tools. You can view the tree representation of the new model in the *occupy.sbsml* file in the *sbsml.design.examples/model* folder.

Submission & Assignment Review

At the assignment review you will have to present your solution, particularly your Xtext grammar, the implemented scoping support, as well as your Sirius viewpoint specification models. You also must show that you understand the theoretical concepts underlying the assignment.

Push the following components into your Github Classroom repository:

- *sbsml*
- *sbsml.design*
  - edited *sbsml.odesign*
- *sbsml.design.examples*
  - 6 diagrams (all diagrams should be arranged in a way that all elements are visible)
- *sbsml.edit*
- *sbsml.editor*
- *sbsml.xtext*
  - edited *Sbsml.xtext*
  - edited *SbsmlScopeProvider.java*
- *sbsml.xtext.examples*
  - created and implemented *occupy.sbs*
- *sbsml.xtext.ide*
- *sbsml.xtext.ui*

Make sure that your solution is pushed to the master branch of your repository. Additionally, tag your final commit with the label '**lab2_solution**' (Command: *git tag lab2_solution*).

Distribution of Points:

- Part A: 9.5 points
  - Xtext grammar: 6 points
  - Xtext scoping: 2.5 points
  - Textual occupy model: 1 point
- Part B: 10.5 points
  - Sirius mappings: 5 points
  - Sirius tools: 4.5 points
  - Graphical occupy model: 1 point

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.