

<b>Model Engineering Lab</b> ISE/FD - Information Systems Engineering Foundation 188.923 VU, WS 2020/21	<b>Assignment 1</b>
<b>Deadline:</b> Push solution to master branch until Tuesday, November 3 <sup>th</sup> , 2020, 23:55 Assignment Review: Wednesday-Thursday, November 25 <sup>th</sup> -26 <sup>th</sup> , 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.

At the end of this document you will also find many helpful Tips and links to tutorial videos and further material that will help you solve this lab.

## 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 thing. 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 1: Metamodeling

Your first task is to find a good level of abstraction, such that salesmen and customers can design customised smart solutions together. However, it should contain enough details, so the technicians of your company are able to implement a system.

As orientation you were provided 2 example systems which are already deployed. Your design should be able to model those 2 example systems. Figure 1 and Figure 2 show a graphical representation of those systems.

The goal of this assignment is to develop the abstract syntax of the *Smart Building System Modeling Language (SBSML)*. Therefore, you have to develop a SBSML metamodel with Ecore and furthermore define several OCL constraints to realize additional well-formedness rules for SBSML models.

## Part A: Metamodel Development

### Modeling Domain Description and Example Models

In recent years IoT technologies are on a rise forming more and more smart systems. These can be installed in all kinds of scenarios reaching e.g., from smart homes over smart healthcare to smart factories or farms.

Our company uses smart things of type sensor, actuator, and fog device to build smart systems. In different configurations those things are combined to form networks that can e.g., control the temperature in a room or the humidity in the patch of an indoor farm. This is realized by running controllers on fog devices which process the data provided by sensors and invoke services provided by the actuators in case defined thresholds for the sensor values are reached.

The figures below (Figure 1 and Figure 2) show two example systems modeled with SBSML. Table 1 describes the different modeling concepts in more detail.

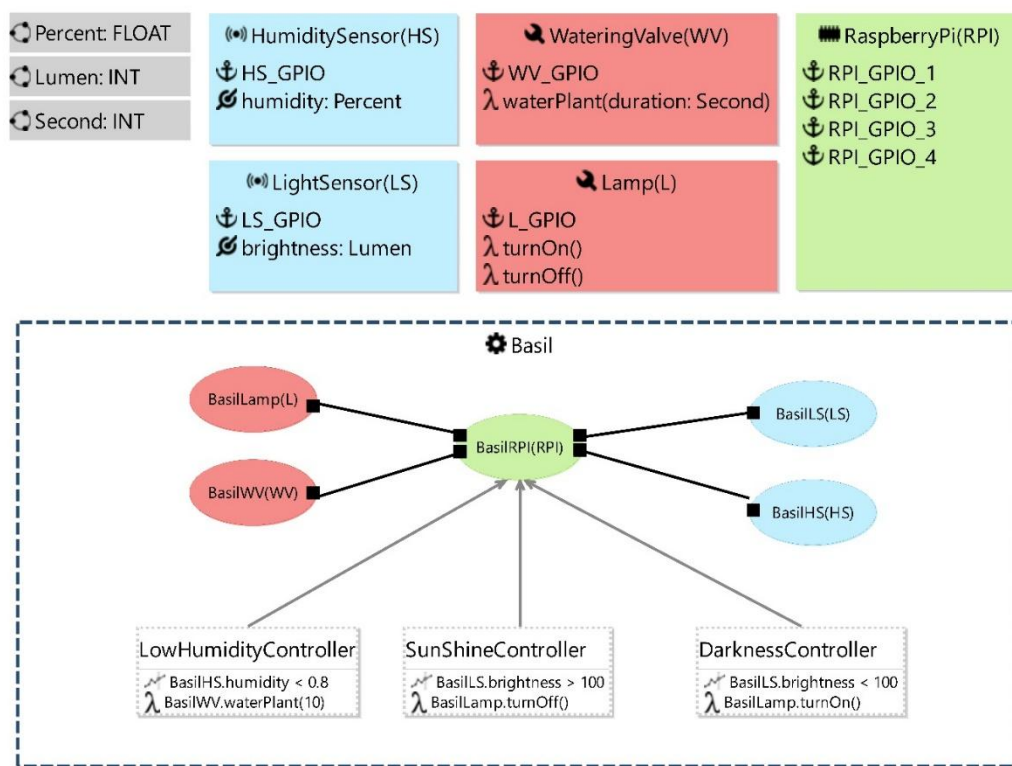


Figure 1: SBSML example of an Indoor Farming model

In Figure 1 you can see the graphical representation of the Indoor Framing example. A more detailed textual description can be found [here](#).

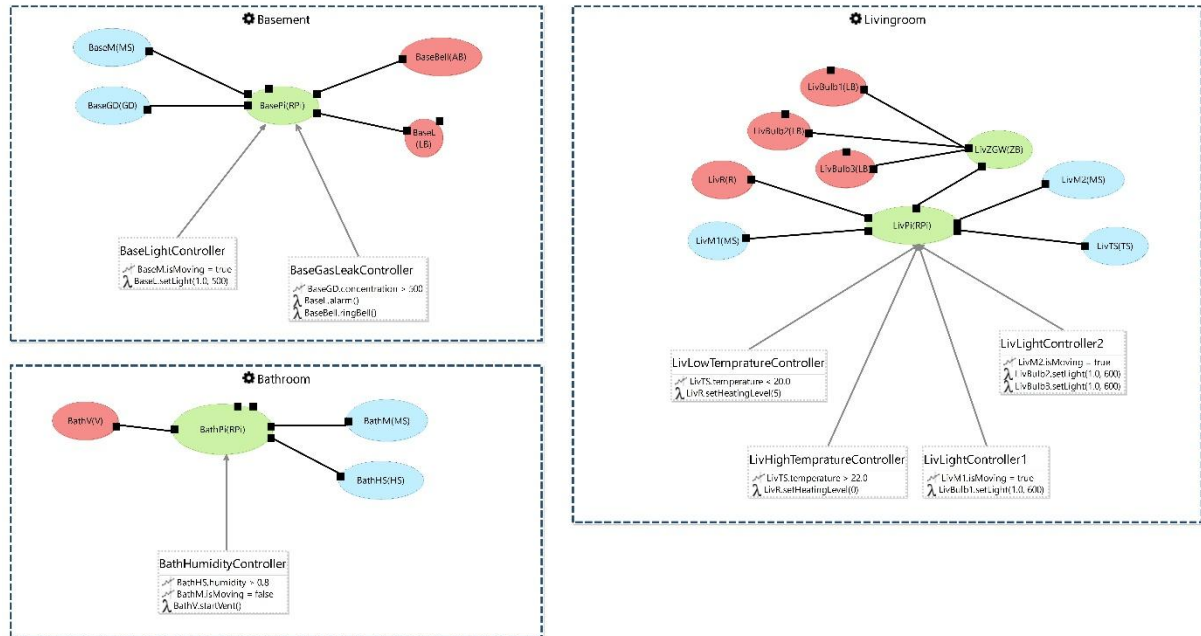


Figure 2: SBSML example of a Smart Home model







In Figure 2 you can see the more complex Smart Home example. This model shows you some concepts not covered by the Indoor Farming example, so you can understand what the SBSML metamodel should cover:

- Multiple connections to a single port like at the node LivZGW in the Livingroom configuration
- Unused ports like in the Bathroom or the Basement configuration
- Multiple service calls like in the LivLightController2 or the BaseGasLeakController
- Multiple thresholds like in the BathHumidityController
- More complex connection network like in the Livingroom configuration

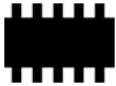

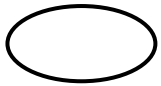


## Modeling Language Description



Table 1 describes the language concepts of the graphical syntax in more detail. Important concepts that you most likely have to cover in your metamodel either as class, relation or attribute, are highlighted. Concepts that are shown in this table in one row are written with the first letter in upper case (e.g. **Sensor**).

Syntax	Description
	Everything is contained within the <b>Smart System</b> . The <b>Smart System</b> has a name. In addition, a <b>Smart Systems</b> defines various <b>Things</b> . Those <b>Things</b> are then further used in one or more <b>Configurations</b> of the <b>Smart System</b> . Also <b>Units</b> are part of the <b>Smart System</b> .

	<p>There are 3 different types of <b>Things</b>: <b>Sensors</b>, <b>Actuators</b> and <b>Fog Devices</b>. All those <b>Things</b> have a <b>name</b> and a <b>shortName</b>, which is used for identification. <b>Things</b> can also define multiple <b>Ports</b>.</p> <p>e.g. HS_GPIO is the General Purpose Input/Output <sup>1</sup> <b>Port</b> of the HumiditySensor which has the <b>shortName</b> 'HS'.</p>
	<p>A <b>Port</b> has a <b>name</b> and provides an endpoint which is used to communicate with <b>Things</b>.</p> <p>Two kinds of Ports can be distinguished:</p> <ul style="list-style-type: none"> <li>• <b>Single-Connection</b>: The <b>Port</b> can only be used for one <b>Connection</b></li> <li>• <b>Multi-Connection</b>: The <b>Port</b> can be used for arbitrary <b>Connections</b></li> </ul>
	<p>A <b>Sensor</b> has one or more <b>Parameters</b>. <b>Parameters</b> describe current sensor values.</p> <p>e.g. humidity is a <b>Parameter</b> of the HumiditySensor with the <b>Unit</b> Percent.</p>
	<p>A <b>Parameter</b> consists of a <b>name</b> and a <b>Unit</b>.</p>
	<p>A <b>Unit</b> has a <b>name</b>, an optional <b>abbreviation</b> with a maximum of 5 characters and a <b>data type</b>, which is either INT, FLOAT, or BOOL.</p> <p>e.g. Percent is a <b>Unit</b> of type FLOAT.</p>
	<p><b>Actuators</b> expose one or more <b>Services</b>, which can be invoked by <b>Service Calls</b>.</p> <p>e.g. Lamp is an <b>Actuator</b> with the exposed <b>Services</b> turnOffLight and turnOnLight.</p>
	<p>A <b>Service</b> has a name. Additionally, <b>Services</b> can have multiple <b>Parameters</b> like procedures in common programming languages. A <b>Service</b> can also have an optional <b>description</b>.</p> <p>e.g. waterPlant is a <b>Service</b> with the <b>Parameter</b> duration of <b>Unit</b> Sec.</p>

<sup>1</sup> [https://en.wikipedia.org/wiki/General-purpose\\_input/output](https://en.wikipedia.org/wiki/General-purpose_input/output)

	<p><b>Fog Devices</b> are used for the computation. They have an attribute, that describes their computation power in <b>mips</b> (micro instructions per second) which has to be greater than 0.</p> <p>e.g. RaspberryPi is a <b>Fog Device</b> with the computation power of 10000 <b>MIPS</b>.</p>
	<p>A <b>Configuration</b> has a <b>name</b>, consists of at least one <b>Node</b>, and of at least one <b>Controller</b>. If there are multiple <b>Nodes</b> in a <b>Configuration</b>, they can be connected with each other as defined by a <b>Connection</b>.</p> <p>e.g. The <b>Configuration</b> named Basil with the <b>Nodes</b> BasWV, BasLamp, BasRPI, BasHS and BasLS and the <b>Controllers</b> BasLowHumidity, BasSunShine and BasTooDark.</p>
	<p>A <b>Node</b> has a <b>name</b> and refers to a <b>Thing</b>. <b>Nodes</b> together with their <b>Connections</b> describe an IoT network.</p> <p><u>Constraint1:</u> If the <b>Thing</b> of the <b>Node</b> is a <b>Fog Device</b>, it can be used to run <b>Controllers</b>. But the sum of all <b>Controller's mips</b> must be smaller or equal to the <b>MIPS</b> of the <b>Fog Device</b> the <b>Node</b> references.</p> <p>e.g. BasRPI is a <b>Node</b> which refers to the <b>Fog Device</b> RaspberryPi.</p>
	<p>A <b>Connection</b> describes how two <b>Nodes</b> are connected with each other. Besides the <b>Nodes</b>, they also refer to the <b>Ports</b>, which are used for the <b>Connection</b>.</p> <p>Only <b>Nodes</b> of the same <b>Configuration</b> can be connected with each other and <b>Nodes</b> can only be connected to other <b>Nodes</b>.</p> <p><u>Constraint2:</u> The <b>Ports</b> which are used have to be part of the <b>Things</b> which the <b>Nodes</b> of the <b>Connection</b> refer to.</p> <p>The <b>Port</b> of a <b>Node</b> can only be used once if it is of type <b>Single-Connection</b>.</p> <p>e.g. the BasLamp <b>Node</b> is connected to the BasRPI <b>Node</b>.</p>
	<p>Each <b>Controller</b> has a <b>name</b> and needs a specific amount of computation power to be executed. This is again given in <b>mips</b> (always greater than 0). The <b>Controller</b> references also the <b>Node</b> on which it is executed. This <b>Node's Thing</b> must be a <b>Fog Device</b>. Besides, a <b>Controller</b> also has at least one <b>Threshold</b> and at least one <b>Service Call</b>. If all <b>Thresholds</b> are reached, the <b>Service Calls</b> are invoked.</p>

	<p>e.g. the BasLowHumidityController needs a computation power of 500 <b>MIPS</b>. If the HumiditySensor BasHS senses a humidity greater than 80% (0.8) then the waterPlant <b>Service</b> of the WateringValve BasVW is called.</p>
	<p>A <b>Service Call</b> refers to one <b>Service</b> and one <b>Node</b>. The <b>Thing</b> of the <b>Node</b> it refers to needs to expose the <b>Service</b> the <b>Service Call</b> refers to. Additionally, a <b>Service Call</b> can have multiple <b>Arguments</b>.</p> <p><u>Constraint3:</u> The amount of <b>Arguments</b> must be equal to the amount of <b>Parameters</b> of the <b>Service</b>.</p> <p>e.g. there is one <b>Service Call</b> in the BasLowHumidity Controller, which invokes the waterPlant <b>Service</b> of the BasVW <b>Node</b> with the <b>Argument</b> `10`.</p>
	<p>An <b>Argument</b> consists of the concrete <b>value</b> represented as string.</p>
	<p>A <b>Threshold</b> refers to a <b>Node</b> and a <b>Parameter</b>. The <b>Thing</b> of the <b>Node</b> needs to be a <b>Sensor</b> and the <b>Parameter</b> of the <b>Threshold</b> needs to part of that <b>Sensor</b>.</p> <p>For each <b>data type</b> there is a concrete <b>Threshold</b>: <b>IntThreshold</b>, <b>FloatThreshold</b> and <b>BoolThreshold</b>.</p> <p>e.g. there is one <b>Threshold</b> in the BasLowHumidity Controller which is reached when the humidity <b>Parameter</b> of the BasHS <b>Node</b> is greater than `0.8`.</p>
	<p>The <b>IntThreshold</b> has additionally a value and a <b>comparator</b>. The <b>comparator</b> is either GREATER or SMALLER. The <b>Threshold</b> is reached, when the value of the <b>Nodes Parameter</b> is GREATER/SMALLER than the given value.</p> <p><u>Constraint 4a:</u> Additionally, the Type of the <b>Unit</b> of the <b>Parameter</b> the <b>Threshold</b> refers to needs to be INT.</p> <p>e.g. there is a IntThreshold in the BasSunShine <b>Controller</b>.</p>
	<p>The <b>FloatThreshold</b> has additionally a value and a <b>comparator</b>. The <b>comparator</b> is either GREATER or SMALLER. The <b>Threshold</b> is reached, when the value of the <b>Nodes Parameter</b> is GREATER/SMALLER than the given value.</p>

	<p><u>Constraint 4b:</u> Additionally, the <b>Type</b> of the <b>Unit</b> of the <b>Parameter</b> the <b>Threshold</b> refers to needs to be FLOAT.</p> <p>e.g. there is a FloatThreshold in the BasLowHumidity <b>Controller</b>.</p>
	<p>The <b>BoolThreshold</b> has additionally a trigger value of type boolean. The <b>Threshold</b> is reached when the trigger value is equal to the <b>Nodes Parameter</b> value.</p> <p><u>Constraint 4c:</u> Additionally, the Type of the <b>Unit</b> of the <b>Parameter</b> the <b>Threshold</b> refers to needs to be BOOL.</p>

*Table 1: SBSML concepts*

## Task Description

Develop the metamodel for SBSML with EMF's metamodeling language Ecore and define OCL constraints for ensuring the well-formedness of SBSML models.

- Make sure that you specified the metamodel as precisely as possible, i.e., the metamodel must contain all described language concepts.
- Use the "Sample Ecore Model Editor", the "Ecore Editor", or the "Sirius Diagram Editing Editor" to create the metamodel.
- It is not required to define operations for the metaclasses.
- Constraints regarding the well-formedness of SBSML models, which cannot be ensured by the metamodel only, should be defined as OCL invariants. Define at least 8 OCL Constraints. Constraints 1-4 are explicitly pointed out in table 1. Add at least 4 more OCL Constraints which are implicitly given in the descriptions. You should be able to explain, why you implemented these constraints, in the assignment review. Providing additional meaningful constraints will not result in point deductions 😊.
- Use the "OCLinEcore Editor" to add the OCL constraints to your metamodel.



**Hint:** While editing in the OCLinEcore editor, it is only possible to save if the file contains no errors.



**Hint:** Autocompletion in the OCLinEcore editor does not always work properly, so do not get confused if functions or attributes that you think are valid, are not proposed.



**Hint:** In OCL use `'->'` to call methods on collections. E.g.: `self.things->collect(t | t.name)` vs. `self.thing.ocllsTypeOf(Sensor)`.



**Important:** Do NOT translate the example models depicted in Figure 1 and Figure 2 to Ecore, but develop a modeling language (i.e., a metamodel) to define the language concepts as described above that allows the definition of the example models.

## Part B: Metamodel Testing

### Task Description

In Part A you developed the metamodel for SBSML using Ecore and defined OCL constraints. They have to be tested in this part of the lab.

- Use the functionalities of EMF to generate a tree-based model editor.
- Model the example system depicted in Figures 1 with the help of this editor. Assume attribute values that are not explicitly given in this document (e.g., mips, names for the ports, etc.). A more detailed description of the indoor farming system can be found [here](#).
- For each of your defined OCL constraints create a copy of your valid indoor farming model and edit it in a way where you violate that constraint. Alternatively, you can construct one small example model which fulfills the constraint and one small example model which does not fulfill the constraint. The invalid example models should only violate the single tested constraint; all other constraints should be fulfilled.
- Put the test models into a new folder named 'examples'. Name your models accordingly to the tested constraint, e.g., 'mipsPositiveNumber\_invalid.xml'.

## Submission & Assignment Review

At the assignment review, you will have to present your metamodel for SBSML, the additionally defined OCL constraints, the generated modeling editor for SBSML, as well as the example models for testing the metamodel and the OCL constraints. The metamodel has to contain all described concepts and it must be possible to model the example smart systems depicted in Figure 1 and Figure 2. Furthermore, the correctness of the defined OCL constraints must be shown using the prepared example models. You also have to show that you understand the theoretical concepts underlying this assignment.

### Push the following components into your Github Classroom repository:

Ecore modeling project "sbsml" including:

- the Ecore model (sbsml.ecore)
- the Ecore diagram (sbsml.aird)
- the 'examples' folder with:
  - IndoorFarming.xml
  - Models for testing the OCL constraints

Make sure that your solution is pushed to the master branch of your repository. Additionally, tag your final commit with the label 'lab1\_solution'.



### Distribution of Points:

- Ecore metamodel: 10 points
- Indoor farming model: 2 points
- OCL constraints + Validation Models: 8 points (1 point per OCL constraint)
- Oral presentation during the assignment review: 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 5 points can be reached by correctly responding to the questions. If a group member is not able to respond during 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 of this student.

### Notes

Tutorials for the tools and techniques used in this assignment can be found in the [TUWEL course](#).

#### Setting up Eclipse

For the lab part, we provide a description of **how to set up an Eclipse version** that contains all necessary plug-ins for all assignments. This Eclipse setup guide is available in TUWEL: <https://tuwel.tuwien.ac.at/mod/page/view.php?id=907975>.

#### Metamodeling with Ecore

Once you have installed Eclipse and all necessary plug-ins, you can **create Ecore models**. Therefore, select *File > New > Other > Eclipse Modeling Framework > Ecore Modeling Project*, click *Next >*, enter the *Project name* “sbsml” on the next page and confirm it with *Next >*, enter the *Main package name* “sbsml” and complete the wizard by clicking *Finish*. The created project “sbsml” contains already some auto-generated files, which can be used to define an Ecore model (sbsml.ecore, sbsml.aird).

The **Ecore diagram** (.aird file) provides a graphical view on the metamodel defined in the Ecore model (..ecore file). For opening the Ecore diagram file make sure that you are in the *Model* perspective (it should be selected in the upper-right corner of Eclipse; if, for instance, the *Resource* or *Java* perspective is opened, select *Window > Perspective > Open Perspective > Other... > Modeling*). In the *Modeling* perspective, you can open the .aird file by double clicking on it and expand it until the diagram “sbsml” is visible. By double clicking, you can open this diagram and start defining your metamodel by dragging metaclasses onto the canvas (*Palette Classifier > Class*).

You can also modify the **Ecore model** (..ecore file) directly with a **tree editor** (right click on the ..ecore file and select *Open with > Sample Ecore Model Editor* or *Ecore Editor*). However, the graphical view in the Ecore diagram will not be automatically updated. To add elements created in the tree editor to the diagram, expand the ..ecore file in the *Model Explorer* and drag and drop the elements onto the diagram canvas.

You can decide, whether you create the metamodel with the graphical editor in the concrete syntax or with the tree-based editor in the tree-based abstract syntax. If you decide to use the tree editor, generate a graphical view after creating the metamodel and layout it such that it is readable.

### **Testing the metamodel**

To **test the metamodel** you can right click on the created .ecore file and select *EPackages registration > Register EPackages into repository*. Now, open the .ecore file in the tree-based editor, right click on the metaclass you want to test and select *Create Dynamic Instance*. Specify a location and name for the new model (e.g., *examples/indoorFraming.xmi*). To determine whether the whole model is correct or not right-click on the root element of the model and select *Validate*.

For more information about EMF visit <http://www.eclipse.org/emf>.

### **OCL in Ecore**

You can add OCL constraints to your metamodel by opening the .ecore file with the OCLinEcore editor by right clicking on your .ecore file and selecting *Open With > OCLinEcore Editor*.

The *OCL Console* is very helpful for defining and testing *OCL constraints*.

To make use of the *OCL Console* open a test model (e.g., *test.xmi*) with the *Sample Reflective Ecore Model Editor*, right click on any model element and select *OCL > Show Xtext OCL Console*.

Further information about OCLinEcore can be found in the help contents of the Eclipse IDE (*Help > Help Contents > OCL Documentation*).



**Hint:** To ensure that edited or new OCL constraints are applied, unregister the EPackages before registering them again.