Business Informatics Group

From In-Person to Distance Learning: Teaching Model-Driven Software Engineering in Remote Settings

Dominik Bork, Andreas Fend, Dominik Scheffknecht, Gerti Kappel, Manuel Wimmer

In:

2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) – Educator Symposium, 2021, pp. 702 – 711.

©2021 by IEEE.

Final version available via DOI:
https://doi.org/10.1109/MODELS-C53483.2021.00114

www.model-engineering.info

From In-Person to Distance Learning: Teaching Model-Driven Software Engineering in Remote Settings

Dominik Bork
TU Wien, Vienna, Austria
Business Informatics Group
dominik.bork@tuwien.ac.at

Andreas Fend
TU Wien, Vienna, Austria
Business Informatics Group
andreas.fend@tuwien.ac.at

Dominik Scheffknecht
TU Wien, Vienna, Austria
Business Informatics Group
dominik.scheffknecht@tuwien.ac.at

Gerti Kappel
TU Wien, Vienna, Austria
Business Informatics Group
gerti.kappel@tuwien.ac.at

Manuel Wimmer
Johannes Kepler University Linz
CDL-MINT, Linz, Austria
manuel.wimmer@jku.at

***Abstract*—The COVID-19 pandemic did not only dramatically impact the personal and social lives, for many academics, it also demanded immediate changes to the way their courses are taught. While a pragmatic approach is to do conventional lectures via video streaming platforms, much more may be done to educate students also in a remote setting properly. This particularly holds true for practice-oriented and technology-engaging courses. This paper describes our experience of transforming an in-person Master level class on model-driven software engineering into a distance learning one. We describe the structure, the content, the teaching and examination format, and the used platforms in detail. We critically reflect on our experiences and report the feedback gained by a post-class student evaluation. We believe this paper provides meaningful lessons learned and best practices for other educators challenged with the task of teaching similar courses in a remote setting. With this paper, we publish an openly available Github repository that features all course content including sample solutions for all practical lab assignments.**

I. Introduction

The COVID-19 pandemic did not only dramatically impact our personal and social lives, it also changed the way Higher Education Institutions (HEI) taught their classes. While distance learning universities have a long lasting experience in remote teaching and students of such universities are used to distance learning, both is not the case for ’regular’ universities and their students, respectively. This establishes a challenge for many academics to transform their in-person classes into ones delivered in remote settings.

This paper reports on our lessons learned of conducting a model-driven software engineering (MDSE) course at TU Wien. The course, entitled *Model Engineering*, was held in the winter term 2020/21. As a response to the global COVID-19 pandemic the rectorate of TU Wien decided that all courses need to operate in a distance learning format, giving the educators three months for transforming their courses. While much literature on the theoretical concepts behind and the design of Massive Open Online Courses (MOOCs), distance learning, and flipped classroom based HEI courses exist, related work on how to conduct conceptual modeling and model engineering in such a format is still underdeveloped [1]. A recent survey by Ciccozzi et al. [2] provided insights on the status quo of teaching modeling and model-driven engineering. Being published in 2018, i.e., before COVID-19, the survey confirms that most of these highly technical and practical courses were initially taught *"based on lectures, often in combination with laboratory activities."* [2, p. 124]

Aside from the long history of conducting the model engineering course at the Business Informatics Group at TU Wien (cf. e.g., [3]), conducting the course in a distance learning format posed several challenges. This paper lists some of these challenges and highlights our lessons learned together with some best practices on how we addressed them. We believe our experience is of value to the conceptual modeling and metamodeling communities in general and the MDSE community in particular. Furthermore, we share the resources used in our course openly such that other educators can adapt and reuse it [4]. The course’s structure is aligned to one of the de-facto standard course books on MDSE [5]. The course's content furthermore maps to essential parts of the model-based software engineering body of knowledge [6], [7]. We describe the building blocks of the course by means of the running scenario we developed. The scenario centers the development of a Smart Building Systems Modeling Language (SBSML).

This paper reports on our experience of transforming an existing in-person model engineering course into a distance learning format. We report on the structure as well as on the organizational and technological setup for preparing the course material, conducting the distance learning course, and examining the students in Section II. The content of the course is then presented in Section III. Eventually, we report the results of the student evaluation (Section IV) and critically reflect on the lessons learned (Section V) before we conclude the paper (Section VI).

II. Course Structure

In the following, we first report how we implemented the *flipped classroom* teaching concept (Section II-A) followed by a discussion on how we conducted the examinations (Section II-B). The team responsible for the model engineering course in the winter term 2020/21 was led by the first author who was supported by guest lecturers in the teaching duties and two student tutors (12 semester week hours) who were primarily engaged in the lab parts (the other authors of this paper).

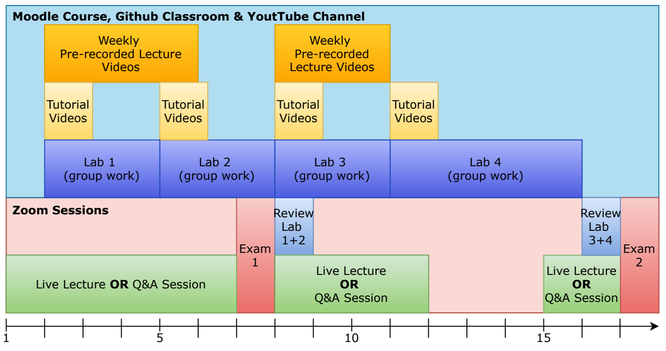
Moodle Course, Github Classroom & YoutTube Channel
Weekly Pre-recorded Lecture Videos
Weekly Pre-recorded Lecture Videos
Tutorial Videos
Tutorial Videos
Tutorial Videos
Tutorial Videos
Lab 1 (group work)
Lab 2 (group work)
Lab 3 (group work)
Lab 4 (group work)
Zoom Sessions
Exam 1
Review Lab 1+2
Review Lab 3+4
Exam 2
Live Lecture OR Q&A Session
Live Lecture OR Q&A Session
Live Lecture OR Q&A Session
1
5
10
15

Fig. 1. Course flow (figure adapted from [1]).

A. Teaching Method

Aiming to maintain a course agenda that fits into the overall curriculum of a – so far – regular university study program, to reuse parts of the well established material, and to not completely cut historical bonds, it was decided to implement – for the first time – a flipped classroom concept [8].

Instead of weekly in-person classes with frontal teaching of the model engineering theory, the content of each class was divided into smaller modules for which individual lecture videos were produced (see Table I). Each of these videos takes between five and 30 minutes - most of the videos last approximately 20 minutes. All videos of a lecture were shared one week before a virtual, live Q&A session was conducted to discuss the contents of that lecture with the students. Thus, the theoretical content was consecutively shared with the students to ensure a balanced workload throughout the semester and to make sure the discussions in the Q&A sessions are focused on the current topics.

Similarly to the theoretical content, also the practical content was distributed consecutively throughout the semester. Lab descriptions and practical tutorial videos were shared with the students only as soon as the assignment of the preceding lab was concluded. Moreover, also for the practical parts, live, virtual Q&A sessions were conducted to provide students with the possibility to directly ask their technology- and lab-related questions.

The weekly virtual meetings were conducted via Zoom[1]. After the first two weeks, we observed that students are not responsive in the virtual setting. In order to engage them actively in the course, we decided to start each session with an interactive quiz realized within the Zoom platform. The quiz featured five to ten single or multiple choice questions related to the current modules.

B. Examination Method

The examination of the model engineering course was twofold to properly reflect the bipartite nature of the course. It aims to develop, to equal extents, theoretical and practical competencies. The examination of the theoretical part was conducted with two open book, multiple choice tests that were executed via the e-learning platform Moodle[2].

The practical competencies were examined by four lab assignments and two solution reviews – one combined for the lab 1 and lab 2 assignments and the second one for the remaining two lab assignments. The course leader together with at least one student tutor conducted the solution reviews which were organized as an individual virtual Zoom meeting for each group.

III. Course Content

In the following, the prerequisites (Section III-A) and the aim (Section III-B) of the model engineering course

[1]Zoom platform [online]: https://zoom.us/
[2]Moodle e-learning platform [online]: https://moodle.com

are introduced. Afterwards, Sections III-C and III-D describe the theoretical and practical content covered in the course, respectively. Eventually, Section III-E closes with a mapping of the content to Bloom's revised taxonomy on learning objectives [9].

A. Prerequisites

The model engineering course at TU Wien is offered at the Master level [3]. It is an obligatory course for business informatics master students and is optional for master students of related study programs such as computer science, data science, and software engineering & internet computing. The course presumes that students have already skills and knowledge in object-oriented modeling, object-oriented programming, data modeling, and data engineering, as well as in software engineering in general. All students have completed the course "Introduction to Object-Oriented Modeling" (cf. e.g., [10]) which aims to teach the basic modeling concepts of a selected subset of UML. In the "Data Modeling" course, the Relational Model as well as the Entity Relationship Model have been introduced. Depending on their specialization, students may have additional modeling knowledge and skills, e.g., in business process modeling.

B. Aim of the course

From a curricular point of view, students at TU Wien have so far seen models in a self-contained environment where the teaching goal was to basically introduce the notation of a modeling language for object-orientated analysis and design. Alternatively, students used models for sketching and documentation purposes in the traditional software engineering courses. The model engineering course aims to close the gap between previous courses focusing topics such as programming, software engineering and modeling. This is an important point as otherwise these topics may be seen as competitors by the students. On the contrary, these topics have to be combined to reach higher abstraction and automation levels. Thus, the model engineering course shall help the students to gain an idea of the practical usage of modeling languages and models in a broader context. The lectures, focusing the knowledge transfer, and the labs, focusing on the development of MDSE capabilities, are worth 3.0 ECTS (European Credit Transfer System) points each. The overall workload is thus 150 hours.

C. Theory

Table I provides an overview of the decomposed previous lecture units into smaller modules, the provided lecture and tutorial videos, and it shows the mapping of the course content to the four practical labs (to be detailed in Section III-D). Moreover, Table I maps the content to the corresponding chapters in the MDSE book [5] and the Model-Based Software Engineering Body of Knowledge (MBSE BoK) [7].

The model engineering course at TU Wien covers the essential parts of the MDSE book [5] (see Table I). Whereas some advanced topics of the book are not covered – like model management and model-driven testing – as the focus of the model engineering course is to not cover as much theory as possible. Instead, the course aims at accompanying the theory with practical hands-on assignments wherever possible. The primary focus of the course is on the *language engineering* and *transformation engineering* parts of MDSE.

D. Practice

For the practical part of the course, the students were divided into groups of three. In these groups four lab assignments had to be solved. In the following, the application scenario and the four labs will be introduced. Excerpts of the solutions will be shown, the complete solutions can be found in the accompanying repository [4].

Application Scenario: In order for the students to better comprehend how the lab parts fit together in a simplified MDSE project, we developed an application scenario that provided a coherent, guiding storyline throughout the semester:

> 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 of describing the same thing. In order to overcome this problem, SensAction 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.
>
> The language should be able to describe sensors, actuators, fog devices and configurations, where these devices are installed as nodes and are connected to each other. Such a configuration also describes controllers, which read sensor values and invoke actuators if a given threshold is reached.

TABLE I
Structure and content of the model engineering course.

Module	**MDSE book** [5]	**MBSE BoK** [7]	**Material**		
			Lecture Video	**Tutorial Video**	**Lab**
Introduction to MDE	1 & 2 & 3 & 4	1	MDE Principles		Lab 1: D1.1: SBSML Metamodel D1.2: SBSML example model D1.3: OCL constraints & models
Metamodeling	6 & 7	1 & 4	Introduction		
EMF-Programming	7	4.2	Metamodeling with EMF Programming with EMF OCL Introduction OCL Tools and Examples	Metamodeling Modeling OCL	
Textual Modeling Languages	7	1.1 & 4.1 & 5.4	Introduction Xtext	Xtext	Lab 2: D2.1: Xtext grammar D2.2: Xtext scoping D2.3: Textual model
Graphical Modeling Languages	7	1.1 & 4.1 & 5.4	Introduction Approaches	Sirius	D2.4: Sirius mapping D2.5: Sirius tools D2.6: Graphical model
Model Transformations	8	8.1 & 8.2	Introduction Graph Transformations	Henshin	Lab 3: D3.1: Henshin transformation D3.2: ATL transformation
ATL	8	8.1 & 8.2	Introduction	ATL	
Code Generation	9	8.3	Introduction Model2Model Transformation Programming Languages Xtend	Xtend Setup Xtend	Lab 4: D4.1: Xtend code generator

For all four model engineering lab assignments, we provided the same two sample models, which should ease the understanding of the modeling language and the tools the students had to design and implement. One of the sample models was an indoor farming system, which can be seen in our graphical notation in Figure 3. It consists of a humidity sensor for measuring the humidity of a patch and to water it using a watering valve if the humidity falls under a specific threshold. Additionally, a light sensor and a lamp is used, which is automatically turned on or off depending the brightness of the room.

Lab 1: Metamodelling: The goal of the first assignment was to develop the abstract syntax of the Smart Building System Modeling Language (SBSML). Therefore, the students had to develop a *SBSML metamodel* with Ecore and furthermore define several *OCL constraints* to realize additional well-formedness rules for SBSML models. We provided two sample SBSML models in a graphical notation and a tabular metamodel [11] that specifies the metamodel by mapping metamodel elements to a sample graphical representation and textual semantic description including four constraints.

The students started from scratch and had to define all metamodel concepts with Ecore. Moreover, they had to realize not only the four pre-defined constraints using OCL, they also had to define and implement four additional, meaningful constraints. Students needed to submit the Ecore project containing the metamodel with the eight OCL constraints and an SBSML model of the indoor farming system as a solution.

The sample solution we provided after the submission deadline can be seen in Figure 2. The root element of such a SBSML model is a *SmartSystem*, which contains *Things*, *Units*, and *Configurations*. Such a *Configuration* further defines *Nodes*, which point to a single *Thing* and *Controllers*. A *Controller* defines a behavior using a *Threshold* and a *ServiceCall*.

The reason why we chose this domain example is that the metamodel is large enough to represent the model engineering core concepts such as classes, attributes, relations, containments, abstraction and enumerations, and at the same time it is not too large, thereby limiting repetitive work where students would not gain further knowledge.

Lab 2: SBSML Concrete Syntax: In the second lab, the students had to implement a *concrete graphical syntax* and a *concrete textual syntax* for the SBSML modeling language. Moreover, tools for the creation and editing of SBSML models needed to be created. Consequently, the assignment was divided into two independent parts. We provided a sample solution from the first assignment, i.e., the implemented metamodel

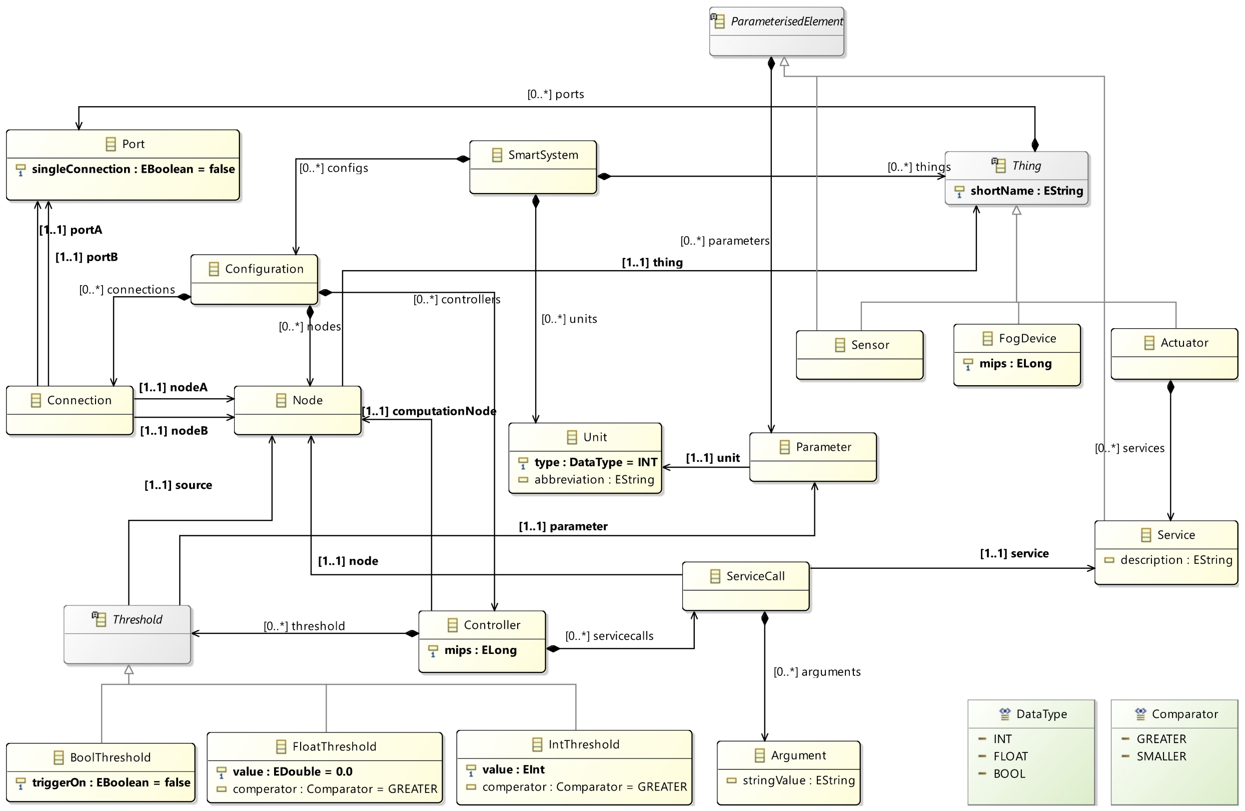

Fig. 2. Sample metamodel for the SBSML language.

shown in Figure 2, such that all groups start with the same setup and students with an insufficient solution for the first lab did not have any disadvantage. In addition, we provided templates for both tasks such that the students could focus on the actual implementation and not have to deal with project setups and configurations.

Graphical Concrete Syntax: For the graphical concrete syntax, the Sirius[3] framework was used. The students had to implement two viewpoints for different aspects of a SBSML model. The viewpoints should define mappings to display the model elements. In addition, for both viewpoints, certain creation tools should be provided, which allow modellers to create model elements.

We used the same graphical models that the students already knew from the first assignment (see Figure 3), therefore the students could concentrate on the task itself. In addition, we defined certain mappings as well as creation tools in the project template for lowering the entry barriers and easing understanding. A template project contained the two example projects and was configured in such a way that the Sirius mappings to be implemented were applied directly to the models. The students could thus directly use EMF

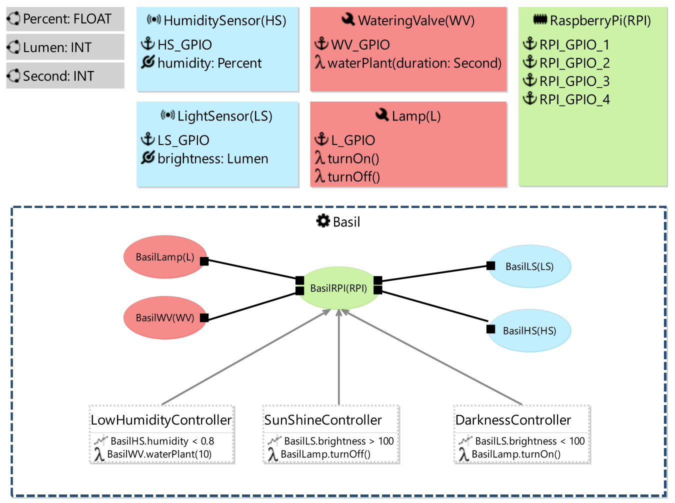

Fig. 3. Indoor farming graphical model.

Compare[4] to compare their solution with the provided graphical examples.

The students needed to submit the implemented Sirius project and a small example model, which was to be created with the implemented graphical editor. Moreover, the realized creation tools had to be submitted for this subtask of lab 2.

[3]https://www.eclipse.org/sirius/

[4]https://www.eclipse.org/emf/compare/

Listing 1. Excerpt of the SBSML Xtext solution for lab 2.

```
SmartSystem:
'system' name=ID '{'
    ('units' '{' (units+=Unit (',' units+=Unit)*)? '}')?
    (things+=Thing|configs+=Config)*
'}'
;

Unit returns Unit:
    name=ID ('(' abbreviation=STRING ')')? ':' type=DataType
;

Thing returns Thing:
    Sensor|Actuator|FogDevice
;

Port returns Port:
    (singleConnection?='single')? 'port' name=ID
;

Sensor returns Sensor:
    'sensor' name=ID '(' shortName=ID ')' '{'
        (ports+=Port)*
        (parameters+=SensorParam)*
    '}'
;

SensorParam returns Parameter:
    'param' name=ID ':' unit=[Unit]
;
```

***Textual Concrete Syntax**:* Xtext[5] was used for implementing a concrete textual syntax for SBSML. In a first step, the students had to define the Xtext grammar, which they had to derive from a textual model of the indoor farming system and the smart home system that we provided. In a second step, the scoping for five pre-defined scenarios had to be implemented, ensuring that the Eclipse content assist only displays valid suggestions.

We implemented some Xtext rules and terminal definitions in the template project for lowering the entry barriers and easing understanding. In addition, the two example models were included in the textual syntax as well as customized examples for scoping. These examples enabled the students to immediately verify their solution. Listing 1 shows an excerpt of the Xtext sample solution for lab 2.

The students needed to submit their implemented Xtext project and the same example model from the graphical concrete syntax sub-task (see Figure 3)), which had to be modeled with the implemented Xtext editor (see Listing 2).

Lab 3: SBSML Model Transformation: Lab 3 was again divided into two independent sub-tasks, one of which dealt with *out-place transformations* and the other with *in-place transformations*. As for lab 2, the sample solution metamodel for lab 1 was provided to the students to ensure all had the same prerequisites.

[5]https://www.eclipse.org/Xtext/

Listing 2. Textual concrete syntax.

```
system IndoorFarming {
    ...
    config Basil {
        node bashs: HumiditySensor
        node basls: LightSensor
        node baswv: WateringValve
        node basl: Lamp
        node basrpi: Raspberry

        connections: {
            basrpi.RPI_GPIO_1 >-< bashs.HS_GPIO,
            basrpi.RPI_GPIO_2 >-< basls.LS_GPIO,
            basrpi.RPI_GPIO_3 >-< baswv.WV_GPOI,
            basrpi.RPI_GPIO_4 >-< basl.L_GPIO
        }

        controller LowHumidityController computed on basrpi {
            mips: 500
            on:{
                FloatThreshold(bashs.humidity < 0.8)
            }
            call:{
                baswv.waterPlant(10);
            }
        }
        ...
    }
    ...
}
```

***Out-Place Model Transformations**:* Out-place transformations show to the students that models created with one language can be transformed easily into models of another one to e.g., address different stakeholders and purposes. ATL[6] was used for implementing the out-place transformations. The two provided sample models had to be transformed into an *AutomationML/CAEX*[7] conforming model according to certain rules we specified. AutomationML 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.

A configured ATL template project was supplied in which only the ATL rules and helpers had to be implemented–for an excerpt, see Listing 3. It was mandatory to use certain ATL concepts, such as *rule inheritance* and *helper functions*. In addition, the two example models were supplied as source models, their expected transformation result, as well as the launch file for executing the transformation. With the model compare functionality of Eclipse, the students could compare their transformation results with the expected ones, and thus, directly verify their implementation.

As a deliverable for this sub-task of lab 3, the students needed to submit an ATL project with the implemented ATL rules. Listing 3 shows an excerpt

[6]https://www.eclipse.org/atl/
[7]https://www.automationml.org/o.red.c/home.html

of the lab 3 sub-task solution by means of the ATL rules for transforming *SBSML FogDevice* instances into *CAEX InternalElement* instances.

Listing 3. ATL rules for FogDevice Nodes.

```
abstract rule Node2InternalElement {
  from
    n : SBSML!Node
  to
    ie : CAEX!InternalElement(
      name <- n.name + '(' + n.thing.name + ')'
    )
}

rule FogDeviceNode2InternalElement extends Node2InternalElement {
  from
    n : SBSML!Node (n.thing.oclIsTypeOf(SBSML!FogDevice))
  to
    ie : CAEX!InternalElement (
      attribute <- thisModule.createAttribute('mips', n.thing.mips)
    )
}

-- Called rule
rule createAttribute(name: String, value: OclAny) {
  to
    attribute: CAEX!Attribute (
      name <- name,
      value <- value.toString()
    )
  do {
    attribute;
  }
}
```

***In-Place Model Transformations**:* In this sub-task of lab 3, we introduced in-place transformations to the students, using them for assuring the quality of the SBSML models. The transformations were applied to automatically fix corrupted models. Henshin[8] was used for realizing the in-place transformations.

We described three different scenarios of incorrect SBSML models and how these models should be fixed by applying Henshin rules. In Figure 4 you can see the sample solution for the scenario *split workload* using the sequential unit *DuplicateFogNodeAndMoveController*, which invokes the rules *Duplicate1FogNode* and *MoveController*. The rule *Duplicate1FogNode* creates a new *Node* element for each existing *Node* with a name passed as an rule argument and connects it to the already existing one. The rule *MoveController* sets the *computationNode* attribute from a *Controller* element, again specified by a rule argument, to a specific Node, as well specified by a rule argument.

Also for this sub-task we provided a fully configured Henshin template project. The rules for the three scenarios were already defined and had to be implemented by the students. For each of the three scenarios, a corrupted SBSML model, its expected transformation result, as well as the launch file for

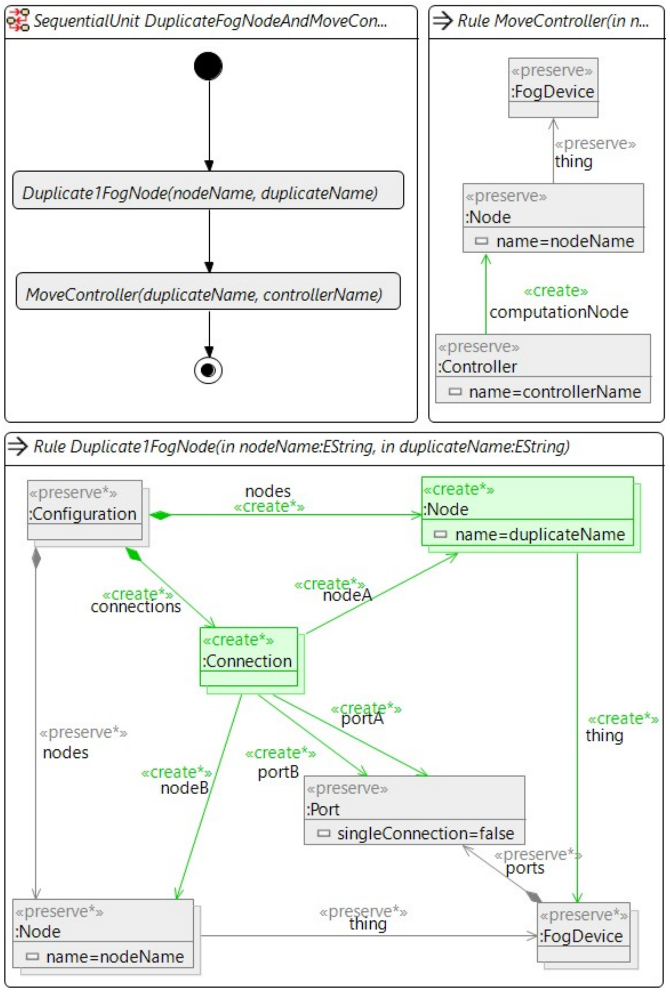

Fig. 4. Sample Henshin rules for the *split workload* scenario.

executing the transformation was supplied. In this way, the students were again able to verify their implementation with EMF compare.

For this sub-task, the students had to submit the Henshin project with the implemented Henshin rules.

Lab 4: SBSML Code Generation: Lab 4 focused on the implementation of a code generation tool that transforms SBSML models into a Java application that simulates the behaviour of the modeled smart building system. The application also monitors the system behaviour by providing sophisticated console logs. When executing such a generated Java program, a scenario file is passed as an argument, which defines an ordered set of values for each sensor node of the system. These sensor values are read and processed according to the thresholds of the controllers and, if applicable, actions of actuators are executed.

For the code generator, Xtend[9] was used with several Eclipse plugins for processing EMF models. Xtend is well suited for this task because the language provides

[8]https://www.eclipse.org/henshin/

[9]https://www.eclipse.org/xtend/

template expressions and is also very similar to Java, which is expected to be mastered by the students.

The example metamodel of lab 1 and the two example models were again provided. In addition, a fully configured code generation project was provided, where the students only had to implement the Xtend files to generate the required Java code. As previously, we included the expected generated Java code, thereby again enabling that students could focus solely on generating the code instead of brainstorming a possible Java application for the scenario. This also enabled us to control the complexity of the code generator and ensure identical challenges for all groups. The Xtend file for the generation of SBSML units was provided fully implemented for lowering the entry barriers and easing understanding. We also provided launch files to execute the generated code and generate log files. The students could use these log files and compare them with the expected log files to continuously verify their implemented code generator.

The students needed to submit their code generation project with the implemented Xtend files (see Listing 4 for an excerpt of the code generator of the Unit class).

Listing 4. Xtend code generator for Units.

```
package at.ac.tuwien.big.sbsml.codegen.xtend

import sbsml.Unit
import sbsml.DataType
import static at.ac.tuwien.big.sbsml.codegen.xtend.NameUtil.toClass

class UnitGenerator {

  public static final String UNIT_PACKAGE = "unit"

  private def dataType(DataType type) {
    return type === DataType.BOOL ? "boolean" :
           type === DataType.INT ? "int" :
           "double" ;
  }

  def generate(String packageName, Unit unit)
  '''
  package «packageName».unit;

  public class «toClass(unit.name)» {
    private «dataType(unit.type)» value;

    public «toClass(unit.name)»(«dataType(unit.type)» value) {
      this.value = value;
    }

    public «dataType(unit.type)» getValue() {
      return value;
    }

    @Override
    public String toString() {
      return this.value + "«IF unit.abbreviation !== null
      && unit.abbreviation.length > 0»«unit.abbreviation»«ENDIF»";
    }
  }
  '''
}
```

E. Summary of Learning Objectives

Following the preceding descriptions, one can grasp the comprehensive aim this course follows. Considering the framework for teaching conceptual modeling and metamodeling proposed in [12], which is based on Bloom's revised taxonomy [9] on learning objectives, the presented course content covers the following dimensions.

Knowledge Dimension: The presented course covers all four knowledge dimensions. Students need to have the *factual knowledge* about, e.g., modeling languages and model transformations. They also need to have *conceptual knowledge* on how metamodel concepts are organized in meta-hierarchies. Moreover, students had to have *procedural knowledge* as they had to actually develop the SBSML metamodel. Eventually, the course also touched to some extent *metacognitive knowledge* as for some aspects several alternatives have been discussed and realized, e.g., textual and graphical concrete syntax, and different model transformation frameworks.

Cognitive Process Dimension: When considering the cognitive process dimension, students in the described course needed to *Remember* and *Understand* the core concepts of metamodeling and *Apply* them in a new domain. As a prerequisite, students also needed the competencies to *Analyze* the SBSML domain and *Evaluate* intermediate solutions with respect to the requirements of the course and the domain specifics. An emphasis of the course is obviously on the *Create* aspect as the students are heavily engaged with applying their model engineering skills.

IV. Evaluation

As the COVID pandemic forced us to rapidly transform the model engineering course into a distance learning one, we were keen to learn how students perceived the new course format. Moreover, we were interested to see, whether the student's results differed when comparing them with those of the previous years. We used a post-class anonymous evaluation, where out of 61 course participants, 40 responded. Our evaluation focuses on the following research questions:

RQ-1 Is a flipped classroom an adequate teaching paradigm for MDSE?
RQ-2 What is the perceived quality of a distance learning MDSE course?
RQ-3 How do course results differ compared to a physical course?

In response to RQ-1, we can state that students responded very positively about the flipped classroom concept in general and the use of it for the MDSE course (see Figure 5). However, we also recognized

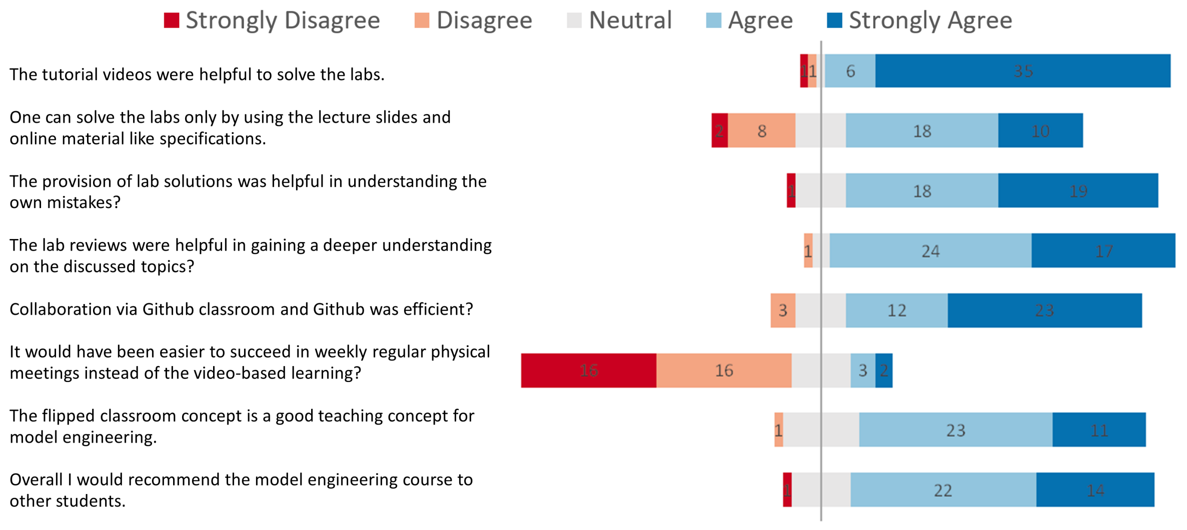

Fig. 5. Results of the student survey.

that approximately only half of the enrolled students participated in the Q&A sessions. On average, students responded that they participated in 3.9 ± 2.3 of eight possible Q&A sessions – 18 out of 40 only attended three or less sessions. In contrast, students watched in average 7.2 ± 1.2 of eight available tutorial videos – 27 of 40 responded they watched all eight videos. Moreover, the hits on the lecture videos on YouTube was very good (in average 143.5 ± 45.5). Overall, students very positively responded to the question whether they think the flipped classroom is a good teaching concept for model engineering (see Figure 5).

What concerns RQ-2, we were interested in the perceived quality of the course by students and the final, overall assessment of the students. For the latter part, we asked in the survey whether or not students would recommend the course. The response was very good with 33 out of 40 agreeing or strongly agreeing to recommend the course.

For RQ-3, we compared the results of the distance learning course with the results of the previous years, where the course with a similar content was conducted with weekly physical classes. When comparing the awarded points in several years, no significant differences could be found. However, it needs to be acknowledged that the change in evaluating the theoretical part hampers a concluding comparison on the knowledge transfer. Considering the MDSE capabilities, the evaluation method and the points for the labs were comparable.

V. Lessons Learned

Before concluding the paper, we share our lessons learned such that others can benefit from what worked and mitigate problems we faced.

Course structure: The flipped classroom concept did work, but with limited adoption on student side. It remains a challenge how to further engage the students actively also in a remote setting. The students throughout the course emphasized the importance and efficiency of providing answers through the student forum. Having access to their repositories, we often used code snippets of their code in our response.

Course content: We saw that the tutorial videos were watched frequently and the evaluation confirmed their value (see Figure 5). We prepared these videos using the lab assignment of the previous semester. This ensured that the tutorial is very close to the actual assignment of this year while not revealing any solution to the SBSML case. When examining the theoretical aspects of the course in a multiple choice test via Moodle, we identified several issues, e.g., related to the size and presentation of large figures (e.g., metamodels), the time required for students to respond to these questions, etc. Consequently, we need to reflect how testing model engineering theory in remote settings can be further improved.

Technological infrastructure: For all practical work of the course, we set up a Github classroom. Each group was assigned a repository in which they could remotely collaborate throughout the course. The administration was very easy and students perceived it to be very useful. On several occasions we helped students in bug hunting or provided an update to the lab solutions. Moreover, the system also allows to monitor the group's progress and to see, which group members are active. This also proved useful when preparing the solution review meetings.

What concerns the tool development environment (i.e., Eclipse) and the lab assignments, we aimed to prepare as much as possible and to enable students to directly and autonomously verify their solutions. This also proved very efficient as the students gained confidence in their solution and questions regarding the fulfillment degree and correctness of a lab solution hardly came up. Still, providing sample solutions for finished labs and enabling all students to start from the same project is inevitable. We even prepared a ready to use Eclipse environment with all necessary plugins for Windows and Unix. However, this was only used in one single case, where a problem with the Eclipse Update Site occurred.

Albeit the mostly positive feedback regarding Github classroom and Eclipse, we also learned that some teams had issues with setting up Eclipse and Github. Most of these problems related to different operating systems, badly configured local Git ignore files and a lack of experience in collaborative and remote programming. In the future, we hope that with the uptake of web modeling and web model engineering environments we can also mitigate some of these issues as real-time collaboration is enabled.

VI. Conclusion

In the paper at hand we described the transformation of an existing in-person model engineering course into a distance learning format. We presented the structure, the content, and the technological and organizational setup we used. The evaluation results showed that our course was perceived very positively on the student side. When looking at the awarded points, we can confirm that students who participated in this very first distance learning course achieved similar good results compared to students of previous, in-person courses. We believe the detailed description of the course and the comprehensive discussion of our lessons learned enables the MDSE community to reflect on their courses and maybe to adopt some of our best practices.

All course content, except for the already publicly available MDSE book slides[10], can be found at the accompanying Github repository [4]. The repository features all four lab assignments, initial workspace configurations for students to start working on the assignments, and complete sample solutions for each lab. We hope the community finds this material useful and invite them to reuse (parts of) the material in their model-driven software engineering courses.

We furthermore plan to continuously extend the top-level Github organization[11] with future MDSE teaching cases, thereby aiming to establish a knowledge base of MDSE cases and quality-assured solutions similar to the current efforts of developing and maintaining a corpora of models [13]. Of course, we hope that others also contribute their MDSE course material and case studies to also better reflect the heterogeneity and richness of the field.

[10]MDSE book slides: https://mdse-book.com/bonus-content/

[11]https://github.com/MDSE-TeachingCases/

References

[1] D. Bogdanova and M. Snoeck, "Using MOOC technology and formative assessment in a conceptual modelling course: an experience report," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2018, pp. 67–73.

[2] F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, G. Taentzer, A. Vallecillo, and M. Wimmer, "How do we teach modelling and model-driven engineering? a survey," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. ACM, 2018, p. 122–129.

[3] P. Brosch, G. Kappel, M. Seidl, and M. Wimmer, "Teaching model engineering in the large," in *Proceedings of the 5th Educators' Symposium in conjunction with the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009)*, 2009.

[4] D. Bork, A. Fend, D. Scheffknecht, G. Kappel, and M. Wimmer, "SBSML course material," Jun. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5026051

[5] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice, 2nd Edition*, ser. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2017.

[6] F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, G. Taentzer, A. Vallecillo, and M. Wimmer, "Towards a body of knowledge for model-based software engineering," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. ACM, 2018, p. 82–89.

[7] L. Burgueño, F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, G. Taentzer, A. Vallecillo, and M. Wimmer, "Contents for a model-based software engineering body of knowledge," *Softw. Syst. Model.*, vol. 18, no. 6, pp. 3193–3205, 2019.

[8] B. Tucker, "The flipped classroom," *Education next*, vol. 12, no. 1, pp. 82–83, 2012.

[9] D. R. Krathwohl, "A revision of Bloom's taxonomy: An overview," *Theory into practice*, vol. 41, no. 4, pp. 212–218, 2002.

[10] M. Brandsteidl, M. Seidl, M. Wimmer, C. Huemer, and G. Kappel, "Teaching Models @ BIG: How to Give 1000 Students an Understanding of the UML," in *Proceedings of the Educators' Symposium in conjunction with the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008)*, 2008, pp. 64–68.

[11] D. Bork, D. Karagiannis, and B. Pittl, "A survey of modeling language specification techniques," *Inf. Syst.*, vol. 87, 2020.

[12] D. Bork, "A Framework for Teaching Conceptual Modeling and Metamodeling Based on Bloom's Revised Taxonomy of Educational Objectives," in *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*, T. Bui, Ed., 2019, pp. 1–10.

[13] J. A. Hernández López and J. Sánchez Cuadrado, "MAR: a structure-based search engine for models," in *Proceedings of the 23rd International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. ACM, 2020, pp. 57–67.