UML++: Enhancing Student Learning of Object-Oriented Modeling through Executable Objects

Pierre Maier
pierre.maier@uni-due.de
University of Duisburg-Essen
Essen, Germany

Tobias Schwarz
tobias.schwarz.1@stud.uni-due.de
University of Duisburg-Essen
Essen, Germany

Abstract

Motivated by the importance of object-oriented modeling in education, we introduce UML++, an object-modeling language that supports instantiation and execution of object models at run time. UML++ is complemented by a modeling tool, UML-MX©, that aims at improving the learning of object-oriented modeling by making modeling more tangible and engaging for students. We outline eight requirements for UML-MX© and present its core features with a focus on the components tailored to levitate learning experiences. A preliminary evaluation indicates that UML-MX© is able to meet most requirements and may significantly aid students in the appreciation and comprehension of object-oriented modeling. Future work will focus on further evaluating the tool's effectiveness in a course setting, refining current features for broader educational use, and expanding its support to other areas of conceptual modeling.

CCS Concepts

• **Applied computing** → **Interactive learning environments**; • **Software and its engineering** → **Object oriented development**; **Abstraction, modeling and modularity**; **Unified Modeling Language (UML)**.

Keywords

UML, Multi-level Modeling, Modeling Education, Modeling Tool

ACM Reference Format:
Pierre Maier and Tobias Schwarz. 2024. UML++: Enhancing Student Learning of Object-Oriented Modeling through Executable Objects. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3652620.3687777

1 Introduction

Object orientation has profoundly shaped the landscape of modern software development, providing powerful language concepts that may extend across the entire software life cycle [26, p. 22]. Central to object-oriented development are object models. They support communication and coordination within software development projects and can serve as a "thinking tool used to aid in the formalization of knowledge" [33, p. 7].

Object-oriented modeling has played an important part in Computer Science (CS) and Information Systems (IS) education for long [12]. The most recent competency models for CS education (jointly published by the ACM, IEEE, and AAAI in 2023 [22]) and for IS education (jointly published by the AIS and ACM in 2021 [24]) underscore the still important role of object-oriented modeling for university curricula; both define object-oriented modeling as a compulsory competence for CS and IS graduates.

Despite the largely undisputed significance of teaching object-oriented modeling, it is still a demanding activity that often enough does not lead to the desired outcome. While the term 'object' is often considered a tangible metaphor, Meyer [26, p. 177] even states that objects "are there just for the picking," the design and analysis of object-oriented systems requires the design of classes from which objects can be instantiated. The design of interrelated classes, however, is no matter of identification, but the result of an abstraction process aimed at solving a particular problem. As Nierstrasz [28] emphasizes: "[…] in the real world, there are only objects. Classes exist only in our minds." Although this abstraction process is essential to high-quality object-oriented systems, our teaching experience suggests that students often struggle with it. This impression is shared and emphasized by various lecturers and researchers in the field [13, 20, 27].

Teaching object-oriented modeling is linked to the overarching goal of developing abstraction skills. Lecturing is insufficient for this purpose. Kramer [21] emphasizes the need for creating conducive educational environments to teach abstraction skills. Such an environment, Kramer (p. 41) continues, should give "students the opportunity to explore many hypothetical questions." In the case of object-oriented modeling, we postulate, this is supported by the provision of an interactive learning environment that allows students to create and manipulate classes and objects at run time to receive immediate feedback on changes in the model.

Teaching object-oriented modeling by focusing on the interrelationship of objects and classes is no novel proposal. In 2002, Andrianoff and Levine [1] proposed that learners should role-play as objects and proclaimed that this can better "illustrate the dynamic behavior" (p. 124) of object-oriented systems. In 2003, Brinda [6] defended an exploration-based approach to learning object-oriented modeling that demanded students to "instantiate objects and construct an object diagram" and "check whether an object diagram is consistent with a class diagram" (pp. 18-19). But as observed by Moisan and Rigualt [27, p. 49], UML modeling tools face a

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687777

core restriction here: "A particularly annoying issue [of UML modeling tools] is that of model consistency. UML proposes a wealth of different views on the system, at different levels of abstraction, corresponding to different phases of development. This is good! But these views are not independent, they have strong and consistent relationships. Most of these relations are ignored by tools and cannot be introduced by the modeler." This shortcoming of UML modeling tools is partly caused by the fact that objects and classes are actually modeled on the same level of abstraction (explained in detail in Sec. 3.1). The official UML documentation itself states that run-time objects "don't appear in models directly" and only "underlie the meaning of models" [29, p. 11].

This restriction of the UML language architecture has long been acknowledged and motivated the introduction of multi-level modeling languages [2, 3]. Multi-level modeling languages alleviate this restriction by allowing the specification of an arbitrary number of modeling levels. Our research group has been developing an executable, multi-level object-oriented modeling language, called the Flexible Multi-Level Modeling and Execution Language (FMML^{x}), that is supported by a corresponding modeling tool, the XModeler^{ML}, for over ten years. The multi-level-modeling environment enables access to models at different classification levels and supports the creation and execution of model instances. Given that the instantiation and manipulation of models at run time may leverage learning effects for students, but common UML tools face an architectural limitation to support this feature adequately, we implemented an object-oriented modeling tool, called UML-MX$^{©}$, that supports modeling with an executable, multi-level UML surrogate, which we refer to as UML++.

To give a comprehensive impression of UML++ and UML-MX$^{©}$, we proceed as follows. At first, we present a list of requirements that should be addressed by an object-oriented modeling tool used for teaching (Sec. 2). Subsequently, UML++ and UML-MX$^{©}$ are described, with a particular focus on the tool's didactic enhancements and underlying language architecture (Sec. 3). The tool is supplemented by a preliminary evaluation in Sec. 4.

2 Requirements for a Didactic Object-Modeling Tool

The development of any tool demands reflection about the needs it should address. We can distinguish two different perspectives here: a teacher's perspective and a student's perspective. Teachers are primarily invested in conveying the teaching content appropriately so that students achieve the required learning objectives. In Sec. 2.1, we describe our teaching context, lay down some of our assumptions concerning learning success and failure of students, and in Sec. 2.2 we explicate the scope of object-oriented modeling concepts we consider. Students, on the other hand, have an interest in accomplishing and mastering the learning objectives. For intrinsically motivated students, learning might suffice as an end for itself. But also for extrinsically motivated students (or even unmotivated students), the attainment of the required skills and knowledge is relevant at least to the degree that students desire to pass the final course exam [25]. Students have no authority to adjust the learning objectives, but they can still report on barriers they encounter and possibly also suggest modifications to the teaching that they assume would improve their learning. We conducted first exploratory experiments with students using a UML modeling tool and a multi-level-modeling tool. In Sec. 2.3, we report on the received student feedback and induce requirements for UML++ and UML-MX$^{©}$ from them.

2.1 Teachers' Perspective: Our Teaching Context and Didactic Assumptions

Our research group has been teaching object-oriented modeling to undergraduate and graduate students for over twenty years as part of our bachelor and master courses on enterprise modeling. The courses cover a variety of modeling topics, such as data modeling, business-process modeling, or domain-specific modeling languages. In our bachelor course, object-oriented modeling concerns structural modeling with class diagrams only. It is attended by IS, CS, economics, and business students. Coming from diverse backgrounds and experiences, students approach the topic of object-oriented modeling differently. In our experience, business and economics students tend to struggle with understanding technical aspects of object orientation: how classes allow for the creation of run-time objects or how objects are syntactically composed, for example. CS and IS students, who mostly have already visited programming and software-development courses, tend to struggle with the abstractions provided in object-oriented modeling. Many of them have problems with the conceptual difference between associations and attributes; others struggle to see the benefit of modeling in the first place. We don't assume that our teaching context is unique but rather think that some of the challenges we encounter are also present in other course settings.

Challenge 1: Diversity of student backgrounds and experiences. One significant reason for the relevance of object-oriented modeling lies in its ability to facilitate communication among various stakeholders. In our university setting, students from diverse academic backgrounds are required to study object-oriented modeling, even if they do not pursue any CS-related degree. Around 200 students take the exam in our undergraduate course, and students encounter individual barriers on their learning paths. These barriers can be subtle and difficult for teachers to identify. With limited resources that prevent personalized training for each student, addressing these diverse learning barriers becomes a considerable challenge. We anticipate that providing students with a tool tailored to accommodate and support their individual learning paths could mitigate this challenge. This may include accounting for specific learning obstacles that may result from specific backgrounds, such as students who have not yet been exposed to object-oriented languages or programming in any way.

Requirement 1: *A didactic object-modeling tool should include features that accommodate and support individual learning paths. This entails providing learning paths that account for (a) varying paces of learning, (b) diverse academic backgrounds, and (c) different levels of prior knowledge in object-oriented modeling.*

Challenge 2: Struggle to reach students, lack of engagement with the subject matter. Although approximately 200 students participate in our exam each semester, only between 25 and 50 consistently

attend our lectures and tutorials. The number of regular participants seems to be declining for some years. Causes for this reduced participation might be multi-faceted and it is not within the scope of this paper to explore these in depth. Nevertheless, we note that engaging students has become increasingly challenging for us, and this negatively affects our ability to deliver the teaching content effectively. The use of software tools to address this challenge reflects a didactic guideline that can be traced to the 1980s [31]: reduce the number of teaching instructions and increase the use of tailored software tools that support students to explore the content on their own. Papert [31, 32] refers to this as a constructionist approach to teaching. Specifically, he proposes to think of learners as builders that must be provided with "materials to build with" [31, p. 7]. A modeling language may serve as such a material. This self-guided exploration, however, needs to be accompanied by active student engagement. For this very purpose, Laszlo and Castro [23, p. 10] note that interactive learning environments should be "simple" and "fun."

Requirement 2: *A didactic object-modeling tool should serve as an interactive learning environment that (a) facilitates self-directed learning by offering prototypical modeling problems that prompt students to search for solutions and (b) motivate students to engage in the subject matter. Self-directed learning entails the provision of suited feedback to solution proposals.*

2.2 Teachers' Perspective: Scope of Object-Oriented Language Features

Any interactive learning environment is primarily aimed at supporting learning and teaching particular learning objectives. To support teaching and learning object-oriented modeling, a tool should implement a corresponding modeling language. Given that the UML is the de-facto standard for object-oriented modeling, we think a modeling tool should maintain the UML syntax and notation as long as they do not violate particular learning objectives being taught. Within our teaching context, only class and object diagrams need to be supported. Support for further UML diagram types is subject to future work (see Sec. 5).

Requirement 3: *Since UML is the standard language for object-oriented modeling, UML syntax and notation for class and object diagrams should be maintained.*

Furthermore, we hypothesize that the instantiation and execution of UML class diagrams may levitate the learning experience of students. Potential effects of this hypothesis are discussed in Sec. 4.

Requirement 4: *Instantiation and execution of models at run time should be supported.*

2.3 Students' Perspective: Insights from Exploratory Student Experiments

The following description reports on an experimental use of a multi-level-modeling tool in our bachelor course on enterprise modeling. We typically use MEMO4ADO [4, 5], a tool that was developed in our research group on the basis of ADOxx, as a modeling tool throughout the semester. It supports modeling all sorts of different diagram types, one of which are UML class diagrams. Last semester, we used the XModeler^{ML}, our multi-level-modeling tool (see Sec. 3.1), which has not been adapted to teaching purposes in any way. Upon reaching the segment on object-oriented modeling in the middle of the semester, we introduced the XModeler^{ML} to the students. In the tutorial sessions on object-oriented modeling, we conducted two in-class assignments with the XModeler^{ML}. Some of our student assistants provided general help with installing and using the tool. For the first in-class assignment, we presented the students with a description of domain objects upon which they should construct classes. In the second in-class assignment, students were provided class diagrams within the modeling tool and were given object-level requests from different imagined stakeholders. They were asked to evaluate whether the request could be fulfilled with the class diagram given and, if not, adjust the class diagram accordingly.

The experience gathered from those in-class sessions was mixed. One notable observation was the considerable time students required to familiarize themselves with the new modeling tool. Many students expressed feeling overwhelmed and confused by the features of the tool. The primary lesson learned from this experience is the necessity to improve initial support for students using the tool, along with restricted access to only those tool features that are also required as part of the course.

Requirement 5: *The number of tool features should be restricted to those required for the course curriculum. Note that this may include tailoring the scope of features towards particular user groups (cf. Requirement 1).*

Requirement 6: *A didactic object-modeling tool should be provided with (a) a comprehensive initial training module and (b) tutorials that support students in using the tool.*

The final assignment was to be completed outside of class. Students were asked to construct a class diagram of a video-streaming platform with a tool of their choice (MEMO4ADO or XModeler^{ML}) based on a scenario description. The scenario description integrated both instance-level and type-level information to avoid that "[all] necessary abstractions have been already made by formulating the text" [13, p. 314]. An excerpt from the assignment reads as follows: "Katherine uploaded a video entitled 'Visiting Utah' on Jul 18, 2023. The video 'Visiting Utah' is a so-called premium video. Premium videos are only available to users who are subscribed to the user who uploaded the video." Students could hand in their solution and a feedback on the chosen modeling tool for bonus points. A total of 34 students submitted their solutions and feedback, with 27 opting to use the XModeler^{ML}.

The feedback we received reinforced our observations from the in-class sessions. Many students noted usability issues or requested additional support within the tool (n=14). Most usability issues were generic (in the style of "difficulties with using the tool"), while others were more specific (e.g., unclear how to add domain-specific data types). Such usability issues can be counteracted by an introduction and guide on how to use the modeling tool (cf. Requirement 6). Incomprehensible tool errors, that made the use of the tool burdensome, were also mentioned by many students (n=11). Some students also noted that they welcomed errors in cases where they indicated faults in the model itself (e.g., not possible to draw link between two objects).

Requirement 7: *Errors that occur when using the tool should be informative and guide students towards correcting flaws in the model.*

Other than this, students requested explanations of object-oriented concepts within the tool (n=8) and noted that they would welcome feedback on their solution or parts of their solution directly within the tool (n=2). Feedback for a student's solution is already entailed by Requirement 2.

Requirement 8: *A didactic object-modeling should include explanations and illustrations of object-oriented modeling concepts within the editor interface.*

This results in a final list of eight requirements that serve as the basis for the development of UML++ and UML-MX©.

3 UML-MX©: The UML++ Modeling Tool

The research of our group is centered around the development and use of domain-specific languages in general and multi-level language architectures in particular. This work led to the development of a multi-level modeling and execution environment, the XModeler^{ML} [8], which is based on the foundational language-engineering environment XModeler [11]. In Sec 3.1, we provide a general overview of the XModeler^{ML} and the multi-level-modeling language it implements, the FMML^{x}. We specifically explicate how they serve to overcome shortcomings of the UML. In Sec. 3.2, we explain how we adapted the XModeler^{ML} to address the requirements outlined in Sec. 2.

3.1 Shortcomings of UML Tools and Prospects of the FMML^{x} and XModeler^{ML}

Traditionally, object-oriented modeling and programming is restricted to two levels of abstraction: a type level that represents classes and an instance level that represents objects. A language user cannot add further levels. Fixed, two-level languages like the UML lead to the following problems:

(1) Objects and classes are represented on the same level of abstraction. The UML is based on a fixed four-level language architecture and is maintained by the OMG. UML models constructed by users reside on level M1. Each modeling element of an M1 model must be an instance of a modeling concept of the M2 UML specification. In principle, the UML allows to model classes and objects. However, since all modeling concepts of the UML reside on level M2 only and modeling elements are instances thereof on M1, classes and objects reside on the same modeling level. Objects in the UML are not instances of user-defined classes. As noted in the UML documentation, an object modeled with the UML "does not depict a precise run-time structure. [...] No conclusions can be drawn about the implementation detail of run-time structures" [30, p. 55]. As a result, the UML only allows for modeling separate class and object diagrams that are not integrated with each other.

(2) Modeling tools implement classes as objects. Modeling tools implemented with a traditional, two-level object-oriented programming language (e.g., Java) face an additional limitation. Every diagram element, such as a user-defined class, is itself an instance of some class within the programming code. This prohibits any further instantiation into user-defined objects within the diagram. This mismatch between the conceptual-modeling level and the implementation level is illustrated in Fig. 1. Consequently, users cannot interact with an integrated view of M0 objects and M1 classes in traditional UML modeling tools. Tools must implement additional

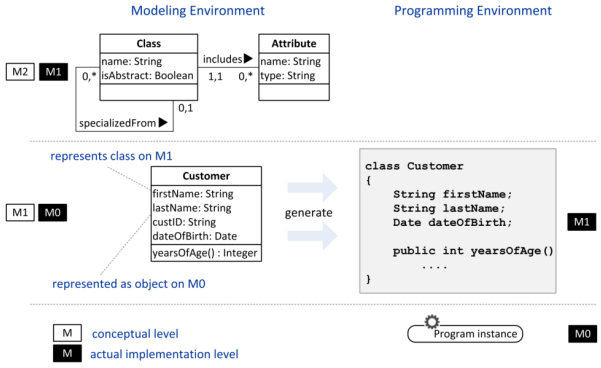

Figure 1: Illustration of mismatch between abstraction levels in conceptual models and programming code [16, p. 469]

consistency checks which are faced with all kinds of difficulties (e.g., type checking). In model-driven software development, this also creates the need to transform models into code – two separate representations typically written with two different languages are required.

Multi-level modeling aims to overcome these problems. In our research group, we have developed the Flexible Multi-Level Modeling and Execution Language (FMML^{x}) that is supported by a respective modeling tool, the XModeler^{ML}. The FMML^{x} is an object-oriented multi-level-modeling language. Its core modeling concepts are similar to the UML's: at the core are classes that may have attributes and may be interrelated by associations. But in contrast to the UML, the FMML^{x}, which is an extension to XCore [9, 10], is a meta-circular language. `Object` in the FMML^{x} meta model is an instance of `Class` and `Class` is a specialization of `Object`. This meta circularity of the FMML^{x} allows for the specification of an arbitrary number of modeling levels. Any class, as an object itself, can be an instance of a further meta class.

Objects in the FMML^{x} represent actual run-time objects. Consequently, the FMML^{x} is an executable modeling language. Next to associations and attributes, classes may also contain constraints and operations. Constraints and operations are written in the executable object constraint language (XOCL), which is a variant of the OCL used for specifying constraints in UML diagrams. FMML^{x} diagrams can be constructed and managed with the XModeler^{ML}, an open-source software that can be downloaded at https://www.wi-inf.uni-due.de/LE4MM/. More in-depth accounts of the FMML^{x}, XModeler^{ML}, and related technologies are provided in [7, 9, 10, 14, 15].

As a result of these language features, the FMML^{x} allows for a common representation of model and code. FMML^{x} diagrams created with the XModeler^{ML} are executable. No additional representation for execution is needed. The model is the code. This avoids any need for further code generation and allows the instantiation and execution of models directly within the tool.

3.2 Adaptation to UML++ Editor

According to our teaching experience and first experiments conducted with students (see Sec. 2), adaptation of the XModeler^{ML}

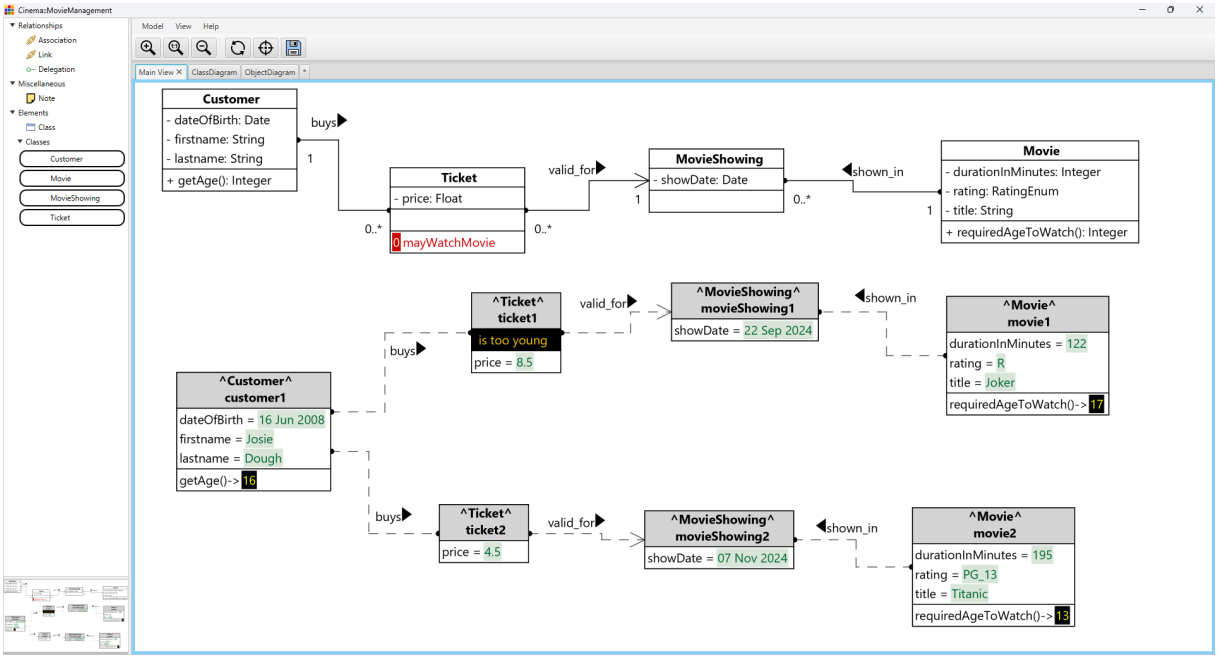

Figure 2: Screenshot from UML-MX© with UML++ diagram

is required for three reasons. (i) First, as a multi-level-modeling tool, it includes many features unfit for the purpose of teaching object-oriented modeling. (ii) Second, while the FMML^x and UML share a set of core modeling concepts, the FMML^x uses a different graphical notation than the UML. (iii) Third, as the XModeler^{ML} was originally not intended for teaching purposes, it does not offer any didactic support. A screenshot of the UML-MX© modeling editor with a UML++ diagram is shown in Fig. 2.

Step 1: Reduction to essential tool features. At first, we set out to identify which tool features were irrelevant to construct executable class diagrams. Irrelevant tool features have been faded out. In UML-MX©, we removed the possibility for users to create multi-level models. Only UML++ diagrams (see Step 2) can be added. User dialogues that open when a class is added, an attribute is modified, or similar actions have been stripped of multi-level language features. Upon opening the modeling tool, the *Control Center* appears, from which users can navigate to the overview of available learning units (see Step 3) and create new UML++ models.

Step 2: Reconception of FMML^x as UML++. UML++ diagrams represent executable models that may include classes on level 1 and run-time objects on level 0. Classes and objects in UML++ diagrams replicate the graphical notation of classes and objects as specified in the UML. UML++ is a confined version of the FMML^x that prohibits the specification of higher-level classes and implements UML notation. Although UML++ resembles the UML, they are not identical. Some fundamental differences include the following:

- Traditionally, the UML offers two separate diagram types for classes and objects. In UML++, objects and classes may be modeled within the same diagram. Note that it is still possible to model classes and objects in different diagrams if desired.
- UML++ objects must be instances of modeled UML++ classes. In contrast to the UML, the class name of an object is not entered manually but is determined automatically. As a result, it may not be any string value but must correspond to its actual class name.
- Operations and constraints in UML++ are executable. While in the UML operations are confined to their signature, the executability of UML++ offers the specification of an operation body. We have adjusted our operation editor so that a user can switch between a *normal mode* and an *expert mode*. In the normal mode, users can only adjust the signature; the operation body can only be modified in the expert mode.
- The graphical representation of objects in UML++ includes a compartment with all return values of executed operations.
- Because UML++ objects are run-time instances of UML++ classes, we require the use of actual data types. UML++, as a subset of the FMML^x offers the following data types per default: `Boolean`, `Integer`, `Float`, `String`, `Date`, `Currency`, and `MonetaryValue`. It is also possible to add, what we refer to in our lectures as, domain-specific data types, i.e., to use one user-defined class as a type for an attribute of another class. Furthermore, users can specify enumerations that contain a custom set of values. In case an enumeration is selected as a data type for an attribute, users can select the slot value via a drop-down menu.
- The context of a constraint must not be added by a user but is added automatically. In UML++, the representation of a class contains an additional compartment that includes its constraints.
- Violations to constraints are shown in UML++ objects.
- UML++ offers native support for delegation associations [18]. Aggregation and composition associations have not yet been implemented at the time of writing.

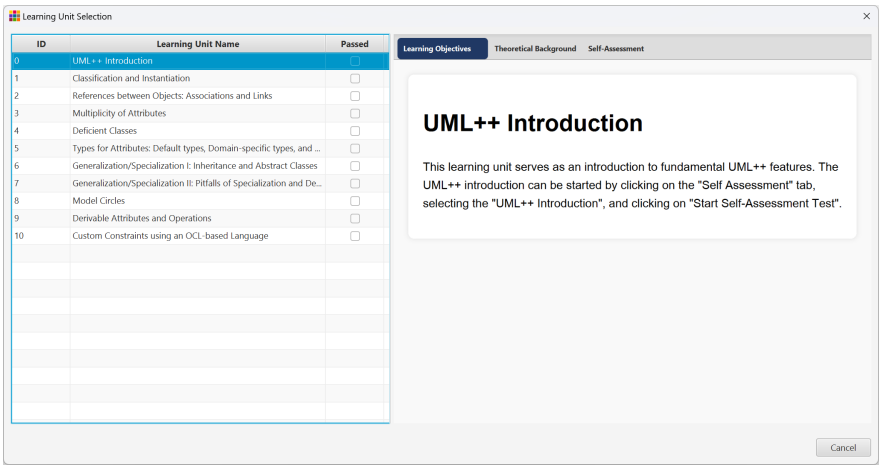

Figure 3: Overview of UML++ learning units in UML-MX©

Step 3: Addition of didactic features. We have implemented a *Guided-Modeling Mode* in UML-MX© designed to assist students in solving modeling problems and gaining a deeper understanding of fundamental modeling concepts. In this mode, the regular modeling editor is extended with a *Task Viewer* that provides users with a task description. Users can only proceed to the next task after successfully creating a model that meets the given description.

The Guided-Modeling Mode serves as the basis for developing learning units (LUs). We have specified ten sequential learning units that align with the learning objectives covered in our courses. Fig. 3 shows the "Learning Unit Selection" window that includes the current set of ten learning units along with a special LU-0 designed to introduce UML++ concepts to beginners (selected in Fig. 3). At the time of writing, the learning units LU-1 (Classification and Instantiation), LU-4 (Deficient Classes), LU-6 (Generalization/Specialization I), LU-8 (Model Circles), and LU-9 (Derivable Attributes and Operations) have been specified in detail and are currently being implemented.

Upon selecting a learning unit, users can switch among tabs displaying (i) the learning objectives for the learning unit (*Learning Objectives*), (ii) background information on the related modeling concepts (*Theoretical Background*), and (iii) a set of self-assessment exercises (*Self Assessment*). An exemplary learning objective from LU-9 is: *"You should be able to correctly identify attributes whose slot values are derivable."* The background information provided for LU-1 includes the following explanation: *"Instantiation refers to the process of creating an object from a given class. In class-based object-oriented languages (such as Java), the specification of a class must always precede the creation of objects. [...]"*. From within the Theoretical Background tab, users can open an example model. In LU-8 this opens a class diagram with instantiated objects where referential integrity is violated. Students may self-assess their understanding of the learning objectives through the different self-assessment tests. Each self-assessment test uses the Guided-Modeling Mode and requires students to fulfill all described conditions for the exercise to be marked as 'passed' within the tool. Completing all tests within a learning unit, marks the unit as 'passed.'

Further information on UML-MX© is provided in [19]. The tool can be downloaded at https://www.wi-inf.uni-due.de/LE4MM/uml-mx/. The web site also provides the latest information on tool updates alongside screencasts demonstrating its use.

4 Discussion and Preliminary Evaluation

A rigid evaluation of UML++ and UML-MX©, also in comparison to other UML modeling tools, has not yet been performed. To the best of our knowledge, no other UML modeling tool supports the instantiation of classes into executable objects. We plan to use our first version of UML-MX© in our bachelor course in the winter term 2024/2025.

First reflections of the current version of UML++ and UML-MX© are given here by two means. In Sec. 4.1 we present further feedback from the first student experiments introduced in Sec 2.3. In Sec 4.2 we reflect upon how the current version of UML-MX© can be evaluated against the requirements presented in Sec. 2.

4.1 Insights from first XModelerML Student Experiments

In Sec. 2.3, we report on first student experiments conducted with the XModelerML without any adaptation for teaching purposes. On the one hand, we noticed several burdens to using the XModelerML and collected student feedback on how a didactic modeling tool might better support their learning experience. This feedback guided the development of our first version of UML++ and UML-MX©. On the other hand, we also observed effects of using the XModelerML that we assume to apply to UML-MX© as well. These include especially the following:

Creation of objects seems to be intuitive and welcomed by students. The first in-class assignment (see Sec. 2.3) required students to construct classes based on a natural-language description of domain objects. During this session, we introduced the students to the XModelerML and explained to them how classes, attributes, and associations can be added. Interestingly, the students started to instantiate objects from classes by themselves without any explicit

instruction or prompt, and even without being introduced to how classes can be instantiated within the tool, to check whether the instantiated objects fit the provided descriptions. This first impression indicated to us that students who are novices in object-oriented modeling intuitively lean towards the creation of objects.

This initial observation was further supported by feedback received from the home assignment (see Sec. 2.3). Of the set of students who used the XModeler^{ML} (n=24), 22 created partial or complete object diagrams to self-assess whether the class diagram met the requirements. The assignment explicitly stated that only a class diagram accounted for the reception of bonus points. Among the smaller group of students who used MEMO4ADO (n=6), our modeling tool that we use for all other tutorials in our course, three complained that the tool did not offer any instantiation of class diagrams. Four students reported that they resorted to modeling objects with pen and paper.

Use of innovative modeling tool seems to increase student engagement. The use of the XModeler^{ML} appeared to have increased student engagement. During in-class sessions, students reported that using an innovative tool which is part of ongoing research sparked their interest. They found the exercises involving the analysis of domain objects, rather than focusing exclusively on the type level as is typical for data, function, and business-process modeling, to be an interesting experience. However, this increased engagement might be attributed to several factors.

For one, if the XModeler^{ML} were used consistently throughout the semester, students might have been less likely to perceive it as a novel and interesting alternative. This shift in perception could also occur over multiple semesters. Once the XModeler^{ML} becomes a standard tool for object-oriented modeling, the initial excitement and engagement it generated may diminish.

Tool seems to support self-guided learning of object-oriented modeling. Overall, our first experimental use of the XModeler^{ML} for teaching purposes supports the hypothesis that simultaneous modeling of classes and executable objects levitates students' learning experience. By allowing students to instantiate and interact with objects derived from their class diagram, the tool facilitated a more tangible understanding of abstract modeling concepts. Students were able to directly observe the impact of their modeling decisions on instantiated objects. This immediate feedback of the tool, albeit it was not yet tailored in any particular way, supports students in their learning endeavors.

What remains an open research question is the extent to which modeling run-time objects by itself increases student engagement and may support self-guided learning. Future experiments should clarify this aspect to better understand how such a hands-on modeling influences student interest and learning outcomes.

4.2 Evaluation against Requirements

At last, we want to reflect upon how the current implementation and conception of UML++ and UML-MX$^{©}$ addresses the requirements outlined in Sec. 2.

The dissection into learning units accounts for Req. 1a. Learning units may be repeated and studied at any desired pace. Req. 1c is also partially addressed by the dissection into learning units in that students are allowed to start with any learning unit they desire. However, the tool currently lacks specific support for common learning barriers and does not offer tailored support for different backgrounds (Req. 1b and 1c).

Req. 2a is addressed by offering modeling problems as part of the different learning units. The set of learning units and especially the number of offered problems is, however, currently restricted and extension of learning units and problems is faced with additional implementation effort. Future versions should (i) provide increased support for instructors to add their own learning units and modeling problems and (ii) account for possibilities to autogenerate modeling problems. Overall, however, it is difficult to evaluate how far exactly UML-MX$^{©}$ facilitates self-directed learning. First exploratory student experiments (see Sec. 4.1) indicate that self-directed learning is also supported through the provision of a run-time, executable modeling environment with immediate feedback on modeling decisions. Those experiments also seem to increase student engagement (Req. 2b) by providing an innovative modeling tool that offers access to objects, which might be more tangible than the manipulation of classes alone. However, we acknowledge that much more research on increasing student engagement and motivation may be accounted for. This may include the implementation of gamification elements, e.g., to provide students with different kinds of rewards.

Req. 3 and 4 are met. More advanced features of UML class diagrams such as association classes or UML profiles are not yet supported since they are not part of our curriculum. All tool features of the XModeler^{ML} have been reduced to the ones required for UML++ modeling. We did not yet account for particularities of different student groups in the provision of tool features (see Req. 1b). Tool features are successively unlocked within the UML++ introduction (Req. 6a). Tutorials are only provided as part of the introduction and learning units (Req. 6b).

Modeling errors and the identification of flaws in models (Req. 7) represent a larger complex than indicated up until this point. On the object level, flaws are easily detectable if they violate specifications made at the class level, e.g., if an object lacks the required number of links or when a slot value violates its attribute data type (*formal-semantic errors*). Similarly, students are informed when they try to violate syntactical rules of the UML++ modeling language, e.g., to leave a class nameless or an association without two ends (*syntactic errors*). These syntactic and formal-semantic errors are identified and reported to the user in UML-MX$^{©}$. The provision of didactic feedback should, however, also account for the adequacy of a modeling solution to provide feedback on modeling decisions that affect reusability or model integrity. This kind of feedback is difficult to implement in a modeling tool because any tool will lack complete information about the modeled domain. Instructive feedback of flawed modeling decisions of this kind is only given within the Guided-Modeling Mode. Here, we also offer explanations and illustrations of UML++ modeling concepts (Req. 8), albeit these are also limited to the current set of learning units.

5 Conclusion and Future Research

In this paper, we presented UML-MX$^{©}$, an object-modeling tool that allows for the instantiation and execution of models at run time without the need for switching between multiple representations.

The tool supports UML++, a multi-level UML surrogate developed on the basis of the multi-level-modeling language FMML^{x}. UML-MX© includes ten learning units that cover foundational concepts of object-oriented modeling and support self-guided learning. First preliminary experiments affirm our hypothesis that modeling executable objects may benefit student learning.

Moving forward, our focus will be on conducting more thorough evaluations of UML++ and UML-MX© within different course settings to refine tool features and provide more support tailored for the use by different educators. This may, for instance, include the implementation of different roles (e.g., instructor, teaching assistant, student) within the tool.

Object-oriented modeling with the UML is only one of many areas of conceptual modeling we cover in our courses. Others include business-process modeling, data modeling, enterprise modeling, meta modeling, and the specification of domain-specific languages. We envision the development of a didactic executable modeling tool that accounts for more than modeling classes and objects with the UML. This may include adding further UML diagram types but also other commonly used modeling languages. Since the UML and the FMML^{x} are both object oriented, no new meta model had to be developed. But the FMML^{x} and XModeler^{ML} can also be adjusted to support other modeling languages without too much effort. Frank and Clark [17], for example, show how meta models for the construction of data-flow diagrams (DFDs) and entity-relationship models (ERMs) can be implemented within the XModeler^{ML}. In the end, we want to offer an educational modeling tool that enables a smooth transition from more fundamental conceptual-modeling areas (like class and object diagrams with the UML) towards more advanced ones (like specifying domain-specific languages) by unlocking more and more tool features as a course advances.

References

[1] Steven K. Andrianoff and David B. Levine. 2002. Role Playing in an Object-Oriented World. In *SIGCSE '02: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, Judith Gersting and Henry M. Walker (Eds.). 121–125.

[2] Colin Atkinson and Thomas Kühne. 2001. The Essence of Multilevel Metamodeling. In *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001. Proceedings*, Martin Gogolla and Cris Kobryn (Eds.). 19–33.

[3] Colin Atkinson and Thomas Kühne. 2002. Rearchitecting the UML Infrastructure. *ACM Transactions on Modeling and Computer Simulation* 12, 4 (2002), 290–321.

[4] Alexander Bock and Ulrich Frank. 2016. Multi-Perspective Enterprise Modeling: Conceptual Foundation and Implementation with ADOxx. In *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*, Dimitris Karagiannis, Heinrich C. Mayr, and John Mylopoulos (Eds.). Springer, Berlin, 241–267.

[5] Alexander Bock, Ulrich Frank, and Monika Kaczmarek-Heß. 2022. MEMO4ADO: A Comprehensive Environment for Multi-Perspective Enterprise Modeling. In *Modellierung 2022 Satellite Events*. 245–255.

[6] Torsten Brinda. 2003. Student Experiments in Object-Oriented Modeling. In *Informatics Curricula and Teaching Methods: IFIP TC3 / WG3.2 Conference on Informatics Curricula, Teaching Methods and Best Practice (ICTEM 2002) July 10–12, 2002, Florianópolis, SC, Brazil*, Lillian Cassel and Ricardo A. Reis (Eds.). Springer, Berlin and Heidelberg, 13–20.

[7] Tony Clark. 2024. Executable Multi-Level Modelling: Establishing Foundations, Methods and Tools. In *Informing Possible Future Worlds: Essays in Honour of Ulrich Frank*, Stefan Strecker and Jürgen Jung (Eds.). Logos, Berlin, 157–172.

[8] Tony Clark and Ulrich Frank. 2020. Multi-Level Modelling with the FMMLx and the XModelerML. In *Modellierung 2020 Proceedings*, Dominik Bork, Dimitris Karagiannis, and Heinrich C. Mayr (Eds.). 191–192.

[9] Tony Clark, Paul Sammut, and James Willans. 2008. *Applied Metamodelling; A Foundation for Language Driven Development* (2 ed.). Ceteva, Sheffield. https://eprints.mdx.ac.uk/id/eprint/6060

[10] Tony Clark, Paul Sammut, and James Willans. 2008. *Superlanguages: Developing Languages and Applications with XMF*. Ceteva, Sheffield. https://core.ac.uk/download/42487298.pdf

[11] Tony Clark and James Williams. 2013. Software Language Engineering with XMF and XModeler. In *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, Marjan Mernik (Ed.). IGI Global, Hershey, PA, 311–340.

[12] David E. Douglas and Bill C. Hardgrave. 2000. Object-Oriented Curricula in Academic Programs. *Commun. ACM* 43, 11es (2000).

[13] Gregor Engels, Jan Hendrick Hausmann, Marc Lohmann, and Stefan Sauer. 2006. Teaching UML Is Teaching Software Engineering Is Teaching Abstraction. In *Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshop OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, Nfc, MDD, WUsCaM, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers*, Jean-Michel Bruel (Ed.). Springer, Berlin and Heidelberg, 306–319.

[14] Ulrich Frank. 2014. Multilevel Modeling: Toward a New Paradigm of Conceptual Modeling and Information Systems Design. *Business and Information Systems Engineering* 6, 6 (2014), 319–337.

[15] Ulrich Frank. 2018. *The Flexible Multi-Level Modelling and Execution Language (FMMLx), Version 2.0: Analysis of Requirements and Technical Terminology*. ICB Research Report. Universität Duisburg-Essen, Essen. https://doi.org/10.17185/duepublico/47506

[16] Ulrich Frank. 2022. Multi-Level Modeling: Cornerstones of a Rationale. *Software and Systems Modeling* 21 (2022), 451–480.

[17] Ulrich Frank and Tony Clark. 2022. Peculiarities of Language Engineering in Multi-Level Environments or: Design by Eliminiation: A Contribution to the Further Development of Multi-Level Modeling Methods. In *MODELS '22: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, Thomas Kühne and Vasco Sousa (Eds.). 424–433.

[18] Ulrich Frank, Tony Clark, Jens Gulden, and Daniel Töpel. 2024. An Extended Concept of Delegation and its Implementation with a Modeling and Programming Language Architecture. *Enterprise Modelling and Information Systems Architectures* 19, 2 (2024).

[19] Ulrich Frank and Pierre Maier. 2024. UML-MX: Boosting Power of Object-Oriented Modeling and Enriching User Experience. In *26th International Conference on Business Informatics (CBI 2024)*.

[20] Luz E. Gutiérrez, Carlos A. Guerrero, and Héctor A. López-Opsina. 2022. Ranking of Problems and Solutions in the Teaching and Learning of Object-Oriented Programming. *Education and Information Technologies* 27 (2022), 7205–7239.

[21] Jeff Kramer. 2007. Is Abstraction The Key to Computing? *Commun. ACM* 50, 4 (2007), 37–42.

[22] Amruth N. Kumar, Raj, Rajendra, K., Herif G. Aly, Monica D. Anderson, Brett A. Becker, Richard L. Blumenthal, Eric Eaton, Susan L. Epstein, Michael Goldweber, Pankaj Jalote, Douglas Lea, Michael Oudshoorn, Marcelo Pias, Susan Reiser, Christian Servin, Rahul Simha, Titus Winters, and Qiao Xiang. 2023. Computer Science Curricula. https://doi.org/10.1145/3664191

[23] Alexander Laszlo and Kathia Castro. 1995. Technology and Values: Interactive Learning Environments for Future Generations. *Educational Technology* 35, 2 (1995), 7–13.

[24] Paul Leidig, Salmela, Hannu, Greg Anderson, Jeffry Babb, Carina de Villiers, Lesley Gardner, Jay F. Nunamaker, Brenda Scholtz, Venky Schnkararaman, Raja Sooriamurthi, and Mark Thouin. 2021. IS2020: A Competency Model for Undergraduate Programs in Information Systems. https://www.acm.org/binaries/content/assets/education/curricula-recommendations/is2020.pdf

[25] W. J. McKeachie. 1963. Research on Teaching at the College and University Level. In *Handbook of Research on Teaching*, N. L. Gage (Ed.). Vol. 1118-1172. Rand McNally & Company, Chicago.

[26] Bertrand Meyer. 1997. *Object-Oriented Software Construction* (2 ed.). Prentice-Hall, Upple Saddle River, NJ.

[27] Sabine Moisan and Jean-Paul Rigault. 2010. Teaching Object-Oriented Modeling and UML to Various Audiences. In *Models in Software Engineering: Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009. Reports and Revised Selected Papers*, Sudipto Ghosh (Ed.). Springer, Berlin and Heidelberg, 40–54.

[28] Oscar Nierstrasz. 2010. Ten Things I Hate About Object-Oriented Programming. *Journal of Object Technology* 9, 5 (2010).

[29] OMG. 2005. Unified Modeling Language: Superstructure. https://www.omg.org/spec/UML/2.0/Superstructure/PDF

[30] OMG. 2006. Unified Modeling Language: Infrastructure. https://www.omg.org/spec/UML/2.0/Infrastructure/PDF

[31] Seymour Papert. 1980. *Mindstorms: Childrens, Computers, and Powerful Ideas*. Basic Books, New York.

[32] Seymour Papert. 1993. *The Children's Machine: Rethinking School in the Age of the Computer*. Basic Books, New York.

[33] Sally Shlaer and Stephen J. Mellor. 1988. *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press, Englewood Cliffs, NJ.