UML-MX©: Boosting Power of Object-Oriented Modeling and Enriching User Experience

Ulrich Frank[1][0000−0002−8057−1836] and Pierre Maier[1][0009−0000−4594−6578]

University of Duisburg-Essen

Abstract. Despite their considerable dissemination, existing UML modeling tools suffer from significant limitations that stand in the way of their profitable use in practice as well as in teaching. This paper presents a new UML modeling tool, called UML-MX©, that overcomes these limitations. It is based on a language architecture that not only enables the integration of class and object diagrams, but also the execution of objects in the diagram editor. Thus, it promotes a more inspiring learning experience. At the same time, it goes beyond the limitations of traditional approaches to model-driven software development by enabling a common representation of models and programs.

Keywords: multi-level language architecture · teaching UML · executable models.

1 Introduction

Despite justified criticism of various weaknesses, UML is widely used in practice. Object-oriented modeling with UML is also an integral part of many curricula, both in computer science and business informatics. In this paper, we present a UML object-model editor that promises substantial advantages over existing UML modeling tools. It is worth noting that the development of a UML tool was not actually on our research agenda. For more than ten years, our research was mainly focused on 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$^{ML©}$ [3] (for more details see www.le4mm.org), [9], which is based on the foundational language engineering environment XModeler© [7]. UML played no role in this context, since the aim of our research was to overcome the limitations of languages like UML. Nevertheless, we still had to deal with UML as part of our teaching program – mainly in a modeling course at Bachelor level.

Against this background, and in view of the large number of existing UML tools, it may seem absurd to develop yet another UML modeling tool. However, there are two main reasons why we decided to do so. On the one hand, we had to realize that despite the obvious advantages associated with multi-level language architectures, software developers and modelers are often reluctant to adopt multi-level modeling. The effort to learn and appreciate multi-level modeling is perceived as rather high, not to say as daunting. In addition, there is

no standard yet, which is a threat to the protection of respective investments. Against this background, it became evident that providing convincing incentives for using multi-level modeling is essential for its adoption. On the other hand, our experience with teaching object-oriented modeling reveals that students often struggle with learning how to design a proper object model, a fact that led to the question how students' modeling skills could be effectively improved. One possible answer to this question was to provide them with a modeling environment that enriches their learning experience by providing a more natural access to objects and classes.

At first, the emphasis of our work was mainly on lowering the entry barriers to multi-level modeling for modelers and software developers. Based on the assumption that UML is still used by many, we came to the conclusion that providing modelers with a UML object-model editor that provides advantages over traditional UML tools might work as an effective incentive. Once modelers got used to these advantages, it might be easier to draw their attention to further multi-level features and eventually to multi-level modeling in general. From a managerial perspective this approach makes sense, too. At first, stick to the standard, then gradually extend it to a more versatile and powerful tool. The idea was, in other words, to introduce multi-level modeling through the backdoor [3].

In parallel we worked on improving a course on object-oriented modeling in a Bachelor's program and a Master's course on advanced modeling and DSML design. In both cases, we were not satisfied with the students' achievements. Even though students represent a different target group than professional users, it became obvious that a UML editor that is based on a multi-level language architecture would also help with improving teaching and learning of object-oriented modeling and would be suited to pave the way for teaching the development and application of DSMLs and, eventually, of multi-level models.

The UML-MX© ("Modeling and Execution") tool presented in this paper serves both purposes. In the following we will describe benefits of using a multi-level tool like the XModeler^{ML} for creating and using UML class diagrams and then define requirements for a dedicated UML object model editor based on the XModeler^{ML} (Sec. 2.2). Against this background, we present the design and implementation of UML-MX© (Sec. 3) as well as a preliminary evaluation (Sec. 4).

2 Prospects and Requirements

Given the limited resources we had for developing UML-MX©, it was essential that the XModeler^{ML} already provides substantial benefits for developing UML-like object models. However, its additional features, such as an arbitrary number of classification levels, deferred instantiation of properties, and the fact that every class is an object is likely to be perceived as confusing, both by experienced professionals and students. Therefore, UML-MX© should allow to benefit from

specific advantages of a multi-level language architecture without placing the burden on users to learn specific multi-level concepts.

2.1 Existing Benefits

A multi-level language architecture offers some clear attractive advantages for creating and using object models. In the industrial application of conceptual modeling, model-driven software development is often regarded as a particularly efficient way to produce code of high quality [10]. In the ideal case, code is widely, if not entirely, generated from models. However, a serious problem stands in the way of this appealing vision. Model and code are represented separately, with the result that sooner or later they will no longer be synchronised. As a consequence, the investments in the models gradually lose their value. That may lead to the question why there is need for generating code, and, as a consequence, for two separate representations anyway. In fact, there is no compelling reason for this. Rather, this circumstance is due to the limitations of common object-oriented programming languages. Classes, which are conceptually located at M1, are actually represented by objects at M0 in the model editor. Objects at M0 do not allow for further instantiation. Therefore, generating code is the only option to enable the instantiation and execution of models. UML even provides two languages for creating class and object diagrams, without offering a proper integration of the two.

By contrast, a multi-level language architecture does not only allow for an arbitrary number of classification levels, it also stipulates that every class is an object, that is, has a state and is executable. Hence, classes that are conceptually located at M1 can be implemented at M1. In other words: model and code share the same representation. This is at least the case for executable multi-level languages like the FMML^{X} which was developed by our team [3]. As a consequence, modelers are relieved from the burden of synchronizing code and model. At the same time testing a model is supported by checking its instances in the same editor. In addition, there are two further goodies provided by the XModeler^{ML} already. Constraints are immediately effective after their specification. They are specified with XOCL, an extended, executable version of OCL [7]. Delegation is not just a pattern like in UML but a language concept with execution semantics [6].

Students are likely to benefit from the integration of class and object diagrams, too. First, there are various studies, which indicate that beginners often struggle with the abstraction required to appreciate the concept of a class. It is easier for them to think of particular objects at first. Therefore, a tool that allows to look not only at classes and objects simultaneously, but also to see the effect of changing a class has on its instances immediately should help students with developing a proper understanding of the fundamental dichotomy of types and instances.

Also, the common representation of object models and their instantiations helps with illustrating problems originating in misleading abstraction. This is,

the case, e.g., for circles or for the inappropriate use of specialization. While corresponding object models may look fine at first glance, looking at corresponding instances will often immediately reveal the problem.

2.2 Specific Requirements

Making use of the features already provided by the XModeler^{ML} would suffice to realise a significant advance over existing UML modeling tools. This applies in particular to the ability to edit class and object diagrams together as well as the executability of objects and constraints. Therefore, the development of UML-MX© was mainly aimed at avoiding the confusion caused by multi-level concepts. To provide for a systematic development of the tool, we conducted a requirements analysis. The following requirements are grouped into three categories. General requirements are marked with "G", those that are marked with an "S" are specifically focused on teaching and learning issues, while "P" marks requirements that may concern professional developers.

As the scope of this paper is limited and the didactically motivated extensions of UML-MX© will be described in more detail in another publication, we will limit ourselves here to a small selection of corresponding requirements.

Requirement G1: Abstract syntax, semantics and concrete syntax should widely correspond to the UML. *Rationale*: Professionals should be familiar with the UML and students are supposed to learn it.

Requirement G2: Specific multi-level concepts and corresponding GUI elements should be effectively hidden from users. *Rationale*: Multi-level concepts could distract from UML, and might be perceived as confusing.

Requirement P1: The tool should allow for adding new language concepts. *Rationale*: To increase productivity and quality in modeling projects it can be helpful to add specific language concepts to the UML that are immediately effective after their specification – even while working on a model. The UML addresses this need with stereotypes. UML tools hardly allow for defining a precise semantics of stereotypes.

Requirement G3: An advanced UML editor should offer optional concepts suited to overcome specific shortcomings of the UML. These extensions should be monotonic, that is, they should not prevent from using UML in a "regular" way. *Rationale*: Respective extensions would facilitate a more efficient and consistent use of UML.

Requirement S1: A UML editor used for educational purposes should offer modeling exercises to students. *Rationale*: Including modeling exercises helps students to apply their knowledge and self-assess their skills.

Requirement S2: A UML editor should give instructive feedback on modeling errors. *Rationale:* Offering detailed and instructive feedback on errors helps students understand their mistakes and thus improve their modeling skills. Errors should be perceived as valuable teaching moments.

Requirement S3: A seamless transition from UML to DSML development and full-scale multi-level modeling should be supported. *Rationale*: Enabling a seamless transition to DSMLs and multi-level modeling ensures that students

can build upon foundational modeling knowledge to address more specific and complex modeling needs step-by-step.

Most of these requirements are widely satisfied already by inherent features of the XModeler^{ML} already, which will be shown in the next section.

3 Design and Implementation

At the core of the proposed tool is a language architecture that is enabled by a specific meta model.

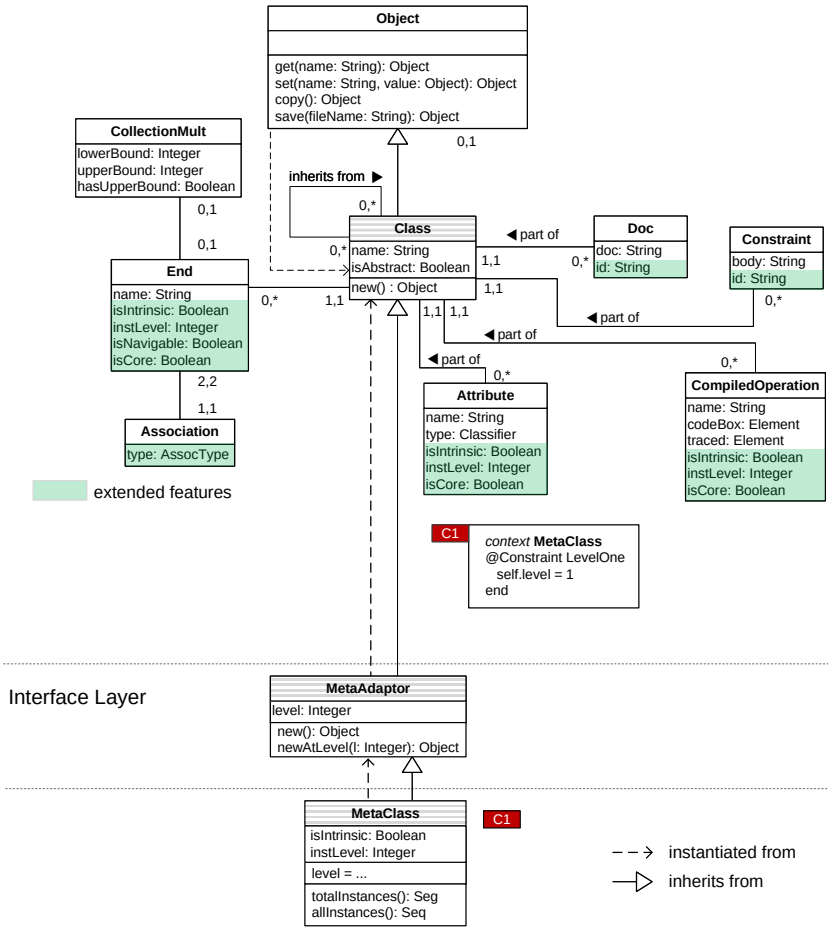

Fig. 1. Foundational Meta Model of UML-MX©

.

3.1 Meta Model

The XModeler^{ML} is based on a reflexive meta model called XCore ([7], pp. 40), which is extended by specific multi-level features resulting in the multi-level language FMML^{X} (see Fig. 1).

It allows for creating multiple classification levels. It also makes sure that every class is an object, since `Class` inherits from `Object`. Objects can execute operations that are defined with the class (instance of `Class`), the objects are instantiated from. Regular FMML^{X} classes can be instantiated at any level from the class `MetaClass`.

The meta model allows to modify existing language features and add new ones with full execution semantics (req. P1).

3.2 Adaptation of GUI and Concrete Syntax

UML-MX© differs from the regular XModeler^{ML} in three ways. First, multi-level language features irrelevant for modeling UML classes and objects have been faded out for the user (Req. G2). Different from the XModeler^{ML}, every class created with UML-MX© is always at M1. As a consequence, the instantiation level of all class properties such as attributes, operations and associations is zero.

Second, the concrete syntax of UML was widely adopted (Req. G1). As shown in Fig. 2, the graphical notation corresponds to UML, with the exception of added features such as delegation and constraints.

Third, a *guided-modeling mode* has been implemented that serves as the basis for self-guided training exercises (Req. S1).

UML-MX© and XModeler^{ML} represent two versions of the same system. To allow for a smooth transition from UML to multi-level languages (Req. S3), additional GUI elements can be gradually unlocked, eventually resulting in a fully-featured multi-level modeling tool.

In order to overcome certain limitations of UML, UML-MX© implements an extended version of UML, called UML++. Note that these extensions are monotonic, that is, they do not exclude "regular" use of UML:

- Objects and classes may be modeled within the same diagram.
- An object created with UML++ is always instantiated from its actual, previously defined class.
- The specification of operations is not restricted to their signature but may also contain a body, which can be accessed by double-clicking on the signature.
- Representations of UML++ objects include a compartment for the return values of their operations.
- Each attribute in UML++ is specified either by a given data type or class (such as `Date`) or by a user-defined class.
- In UML++, the representation of a class contains an additional compartment that includes identifiers of its constraints.

- Violations to constraints are shown in UML++ objects. This includes violations of custom constraints (XOCL expressions) or implicit constraints such as association multiplicity.
- In contrast to the UML, UML++ provides native support for delegation relationships as a means to overcome pitfalls of specialization (see Req. G3).

We have specified ten learning units that each contain an exemplary illustration and a set of modeling exercises. Modeling exercises are opened in the guided-modeling mode, which assists students in developing a model step-by-step.

3.3 Demonstration

Fig. 2 serves to demonstrate the use of the UML-MX© environment. The tool itself, as well as screencasts that demonstrate the guided tool introduction, can be accessed via www.LE4MM.org/uml-mx/. In contrast to prevalent UML class-diagram editors, the palette is dynamically extended by every class that is created within a diagram. Users can click a class on the palette and then click on the modeling canvas to create an instance of this class. By selecting `Class` from the palette, users can create a new class. The same principle applies to associations, links, delegations, and notes.

The specification of attributes can refer to standard data types (like `String`), predefined classes (like `Date`), user-defined classes, or user-defined enumerations. Slot values are accepted only if they conform to the specification of the corresponding attribute.

In UML-MX©, the context of a constraint is added automatically, users must only specify the XOCL expression. Users can furthermore specify custom fail messages that are shown in an object if the constraint is being violated (see object `Modeling` in Fig. 2). For a detailed illustration of how to specify and evaluate constraints see the screencast at www.LE4MM.org/uml-mx/.

Operations contain a body written in XOCL. Operations may or may not be shown in objects. In Fig. 2, for example, the operation `getCourses(): Set(Course)` is not displayed in student objects. Two modes to add/modify operations are available for students: a standard mode and an expert mode. In the standard mode, users can only adjust the signature of the operation. The body can only be modified in the expert mode.

Furthermore, users can manage a model's complexity with views and diagrams. Views are divisions of a diagram based on layout alone. A diagram is a visual representation of a model that can be manipulated in the editor (Fig. 2 shows one UML++ diagram). Users can create multiple diagrams for a model, allowing for a separation of class and object diagrams if desired.

4 Evaluation

Most of the requirements listed in Subsec. 2.2 have been fully implemented, which follows from the description of the implementation in Subsec. 3. Others, especially those that aim at additional support for students such as requirements S1

or S2 are subject of ongoing work. The step-by-step transition is currently supported only to a limited extend: by switching didactic mode from true to false in the file `users.properties`, one can move from UML-MX© to the XModeler^{ML} or, in other words, from UML++ to a multi-level language that enables the specification and execution of DSMLs.

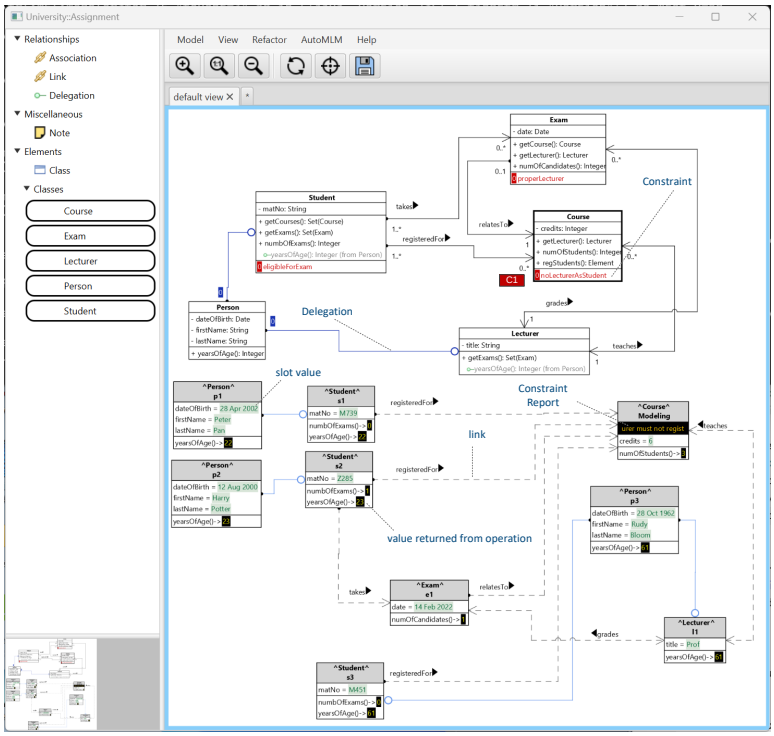

Fig. 2. Screenshot of Diagram Editor within UML-MX©

.

4.1 Preliminary Results from Using the Tool

Object-oriented modeling with UML class diagrams has been taught by our group for over twenty years. In recent years, we used MEMO4ADO, an enterprise-modeling tool developed on the basis of the meta-modeling platform ADOxx, as a modeling tool throughout the course [13]. It includes an editor for common UML class diagrams. Last semester, we used the XModeler^{ML} for teaching object-oriented modeling. It still featured multi-level modeling concepts and lacked UML notation. Students could use either MEMO4ADO or XModeler^{ML}. Subsequently, they were asked to complete a survey on their experience with using the tool as part of the assignment.

Overall, we received mixed feedback from these first exploratory experiments with students. On the one side, many noted that the tool was confusing especially with respect to the offered multi-level concepts. On the other side, the instantiation of models at run-time appeared to support students effectively in their learning process. For one, creation of object diagrams seemed to be intuitive to many students. Without any explicit prompt, students created objects already in the first in-class assignment. Students noted that the creation of object diagrams provided valuable feedback on the correctness of the corresponding class diagram. For example, modeling object diagrams clarified whether the multiplicity of an association was suited for linking objects.

4.2 Related Work

Teaching object-oriented modeling is linked to the overarching goal of developing abstraction skills [8]. Some researchers propose to focus more on modeling objects than classes alone as a means to make students better understand the various dependencies between classes and objects (e.g., [1] and [2]). As noted by [12], UML modeling tools fail to adequately support modeling object and classes since class and object diagrams are not well integrated. We conducted a preliminary survey on current tool support for UML class and object diagrams. While some consistency checks are often provided (e.g., class name in object corresponds to the name of a class), they cannot imitate an executable run-time environment. To the best of our knowledge, no self-standing modeling tool allows for the specification and execution of operations within diagrams or validates constraints on the object level.

5 Conclusions and Future Work

In retrospect, UML-MX© is the result of an unusual, yet fortunate coincidence of technology push and demand pull forces. As a side effect, so to speak, our many years of research on multi-level language architectures have resulted in the opportunity to realise a modeling tool that makes a clear difference to conventional tools and promises considerable advantages, both in teaching and in practical application. To the best of our knowledge, no other UML editor is available that would allow for a common representation of class and object diagrams or would even enable the execution of models without the need for code generation. Preliminary results from using a predecessor of UML-MX© in modeling courses indicate that the additional features implemented especially to support students are suited to add further value. We also assume that using UML-MX© is suited to serve as a door opener for appreciating higher levels of abstraction as they are provided by multi-level language and model editors such as the XModeler$^{\mathrm{ML}}$. The smooth transition from the UML-MX© environment to the XModeler$^{\mathrm{ML}}$ makes it all the easier to get started with multi-level modeling. Our future research is aimed at adding other UML diagram types such as use case diagrams and add further support for students and teachers. We also plan on conducting more sophisticated experiments with UML-MX© in the future.

References

1. Andrianoff, Steven K., Levine, David B.: Role Playing in an Object-Oriented World. In: Gersting, J., Walker, H.M. (eds.) SIGCSE '02: Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, pp. 121–125 (2002)
2. Brinda, Torsten: Student Experiments in Object-Oriented Modeling. In: Cassel, L., Reis, R.A. (eds.) Informatics Curricula and Teaching Methods: IFIP TC3 / WG3.2 Conference on Informatics Curricula, Teaching Methods and Best Practice (ICTEM 2002), pp. 13–20. Springer, Berlin, Heidelberg (2003)
3. Frank, Ulrich: Multi-level Modeling: Cornerstones of a Rationale. Software and Systems Modeling **21**, pp. 451—480 (2022)
4. Frank, Ulrich: Multilevel Modeling. Toward a New Paradigm of Conceptual Modeling and Information Systems Design. Business and Information Systems Engineering **6**(6), pp. 319—337 (2014)
5. Frank, Ulrich, Clark, Tony: Multi-Level Design of Process-Oriented Enterprise Information Systems. Enterprise Modelling and Information Systems Architectures (EMISAJ) **17**, pp. 1—50 (2022). DOI: 10.18417/EMISA.17.10
6. Clark, Tony, Frank, Ulrich, Gulden, Jens, Töpel, Daniel: An Extended Concept of Delegation and its Implementation within a Modelling and Programming Language Architecture. Enterprise Modelling and Information Systems Architectures (EMISAJ), **21** (2024) .
7. Clark, Tony, Sammut, Paul, Willans, James: Applied Metamodelling: A Foundation for Language Driven Development. 2nd edn. Ceteva (2008)
8. Engels, Gregor, Hausmann, Jan Hendrik, Lohmann, Marc, Sauer, Stefan: Teaching UML Is Teaching Software Engineering Is Teaching Abstraction. In: Bruel, J.-M. (ed.) 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, pp. 306–319. Springer, Berlin, Heidelberg (2006)
9. Frank, Ulrich, Clark, Tony: Language Engineering for Multi-Level Modeling (LE4MM): A Long-Term Project to Promote the Integrated Development of Languages, Models and Code. In: Font, Jaime, Arcega, Lorena et al. (eds): Proceedings of the Research Projects Exhibition at the 35th International Conference on Advanced Information Systems Engineering (CAiSE 2023), pp. 97—104. CEUR, 3413 (2010)
10. France, Robert B., Rumpe, Bernhard: Model-driven Development of Complex Software: A Research Roadmap. In: Briand, Lionel C., Wolf, Alexander L. (eds.): Workshop on the Future of Software Engineering (FOSE '07). International Conference on Software Engineering (ISCE 2007). pp. 37—54, IEEE CS Press
11. Atkinson, Colin, Kühne, Thomas: The Essence of Multilevel Metamodeling. In: Gorgolla, Martin, Kobryn, Chris (eds.): UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 1-5, 2001. Proceedings. LNCS, vol. 2185, pp. 19–33. Springer, Heidelberg (2016)
12. Moisan, Sabine, Rigault, Jean-Paul: Teaching Object-Oriented Modeling and UML to Various Audiences. In: Ghosh, S. (ed.) Models in Software Engineering: Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009. Reports and Revised Selected Papers, pp. 40–54. Springer, Berlin, Heidelberg (2010)
13. Bock, Alexander, Frank, Ulrich, Kaczmarek-Heß, Monika: MEMO4ADO: A Comprehensive Environment for Multi-Perspective Enterprise Modeling. In: Modellierung 2022 Satellite Events, pp. 245–255 (2022)