Teaching Model Driven Architecture Approach with the Sirius Project

Ralph Maschotta, Sven Jäger, and Armin Zimmermann

Systems and Software Engineering Group
Computer Science and Automation Department
Technische Universität Ilmenau
Ilmenau, Germany
Contact: see http://www.tu-ilmenau.de/sse

Abstract. The OMG's Model-Driven Architecture (*MDA*) approach was published more than 10 years ago. There were many attempts to use this approach using different tools, which nevertheless has not lead to the widespread use of the *MDA*. The authors believe that among the reasons for this is the lack of an integrated tool chain that fully supports this approach, the high amount of specialized knowledge about specifications, as well as the lack of educational support for practitioners. However, the development of so-called meta tools has progressed considerably since the proposal of *MDA*, allowing to reduce the development effort significantly. It also allows to teach the *MDA* approach and necessary specifications using an available tool chain with the Eclipse Sirius project. This paper presents an experience report teaching the *MDA* approach to computer science students. The structure of the newly implemented course, the used tool chain and the students performances as well as the results of the first course evaluation are presented.

Keywords: MDA, OOM, Sirius, Eclipse, EMF

1 Introduction

Programming software systems has become easier, simpler and more effective over the last few decades. Writing simple sequences of instruction bits has long been replaced by more efficient assembler programs, which have a higher level of abstraction. This development continued with the development of higher-level programming languages and later object-oriented languages. Every time, when the abstraction level increased, the development of complex software became more efficient. As a result, highly complex software solutions are possible today with an acceptable effort. Another issue is expandability and reusability of software solutions, which calls for just another increase of the abstraction level. Model-based design of software systems is such a step that raises the abstraction level once more. The Unified Modeling Language (*UML*) [1] is the de-facto standard to describe structure and behavior of complex software systems today. These models are usually used in the design phase of software development processes or for documentation reasons. Sometimes, models are used to generate

the skeleton of a software system; or, in the case of code refactoring, models are generated based on existing code. Nevertheless, additional source code has to be written manually.

The Model-Driven Architecture (*MDA*) is seen as another step in this progress. This formal but still incomplete definition is a software design approach for the development of complex software system, which was defined by the Object Management Group (*OMG*) in 2001 and updated by the "MDA Guide Revision 2.0" in 2014 [2]. It proposes to start software development with a Platform-Independent Model (*PIM*) of an application's business functionality and behavior, constructed using a modeling language based on *OMG*'s MetaObject Facility (*MOF*) [3]. This *PIM* is converted to a Platform-Specific Model (*PSM*) using Model to Model (*M2M*) transformations and then to a working implementation using Model-to-Text transformations. This generative software development approach is expected to enhance development efficiency and to reduce software bugs and development cost [4–6]. It is supported by several specifications such as *UML* extended by profiles including the *UML* Profile for CORBA, the UML Profile for Quality of Service and Fault Tolerance (*QFTP*) and others. Further major specifications in this area include *XML* Metadata Interchange (*XMI*), Query-View-Transformation (*QVT*), *MOF* to Text (*MOF2T*), and Diagram Definition (*DD*). In order to obtain executable models of software systems, the "Precise Semantics Of UML Composite Structures" (*PSCS*) and the "Semantics Of A Foundational Subset For Executable UML Models" (*FUML*) are defined as the first step in this direction [7].

A supporting tool is necessary to apply these specifications for a software system [8]. Several *MDA* development tools have thus been developed [9–14]. These so-called meta CASE tools support the specifications in individual ways, with different complex workflows and tool chains. For instance, the Eclipse Modeling Project [8] supports model-based development technologies providing a unified set of modeling frameworks, tooling, and standards implementations. The different steps of the *MDA* approach are supported by diverse Eclipse projects and plugins [14, 8, 15]. The model transformation as one part of the *MDA* approach could be performed using 5 different tools. Hence, a comprehensive tool chain for the *MDA* approach is necessary.

The Eclipse Modeling Project has emergent as a standard in the field of software modeling: it provides a set of generative components and run-time infrastructures for developing graphical editors based on the Eclipse Modeling Framework (*EMF*) and the Graphical Editing Framework (*GEF*) [8, 12]. It simplifies the processes of defining metamodels and creating models using generated modeling software. The Eclipse Modeling Project includes the Graphical Modeling Framework *GMF*. It allows, among others, model components and editors to be automatically generated from a meta model. Moreover it provides models to describe graphical elements and to describe tools for their manipulation. The following Figure 2 gives an overview of some elements of the Eclipse *GMF*-based *MDA* approach.

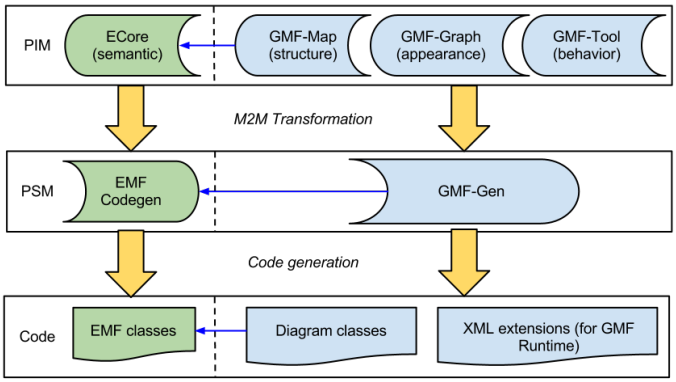


Fig. 1: The Eclipse *GMF*-based *MDA* approach [16]

In-depth knowledge of different *OMG* specifications and tools and tool chains is necessary to use the *MDA* approach in the generation of a graphical editor based on a self-defined metamodel using different meta case tools. A significant amount of time is necessary to familiarize oneself with the *MDA* approach. This may be one reason for the recently diminishing interest in using the MDA approach. On the other hand, there are still several endeavors to use the *MDA* in the development of complex software systems. To apply the *MDA* approach in real software systems in the future, the development effort has to be decreased considerably by improving the tool support. On the other side, educational institutions should teach the *MDA* approach, the necessary specifications, and the practical applications using current tools.

There are several approaches to considerably reduce the development effort using the *MDA* approach [17, 5, 18, 19]. The evaluation of different meta case tools based on the task to generate a simple BPMN editor has been presented in [17]. The required time to generate the editor is reduced from 25 days using Eclipse EMF to five days using Obeo Designer, and to 0,5 days using MetaEdit+. This short time allows the inclusion of a practical applications of the *MDA* in small educational projects for students. Knowledge of *MDA* specification principles and workflows is however also required and must be an additional subject of a corresponding lecture.

This paper presents an experience report of teaching the *MDA* approach as a part of an object-oriented modeling class. The structure of the lecture, the seminar, and the tool chain used are based on the recent Eclipse Sirius project and will be presented step by step in the subsequent sections. In Section 5, student performance and the results of the class evaluation are presented before the paper is summarized in Section 6.

2 Course Design

The semester course should qualify students to model structure and behavior of systems using standard modeling techniques and notations like *UML*. Knowledge of the *MDA* approach should also be integrated into this lecture. The lecture is scheduled as a Master-level lecture for students of computer science and related curricula. Knowledge of object-oriented programming techniques and at least one object-oriented programming language are required as well as basic knowledge of UML class diagrams.

The course is composed of a lecture (1.5 hours per week), where the theoretical knowledge is conveyed; a seminar, where the practical methods and tools are presented by a lecturer (1.5 hours every two weeks); and a homework session, where the students have to solve practical tasks in small groups.

2.1 Structure of the Lecture

The necessary fundamentals of *MDA* are taught during the first part of an existing lecture on Object-Oriented Modeling. It spans two lectures, each 1.5 hours in length. The introduction contains a short historical survey as motivation; the general content of the *MDA* and some additional fundamentals. The second lecture presents the *MOF* metamodel layer structure using the layered metamodel architecture of the *UML*. To clarify the membership of an element to an appropriate metamodel level, the different metamodel levels are associated with a specific color, which is retained both in the lecture and the seminar.

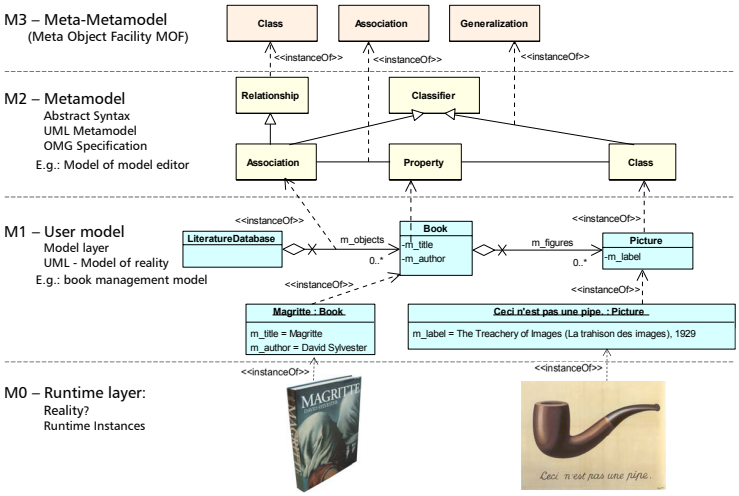


Fig. 2: Example slide of the lecture, presenting the *MOF* metamodel layer structure using the layered metamodel architecture of the *UML*.

The general approach of transformation from *PIM* over *PSM* to the code is repeated thereafter. The involved models are covered and the relationship to the different *MOF* metamodel levels is shown (model / metamodel / meta-metamodel). A practical example using the selected tool chain (see Section 3) is presented using a simple domain-specific language (*DSL*), and the generated editor for model families is used as an example. This step-by-step example in relation to the theory of *MDA* approach helped to clarify the theory of the *MDA* approach for the students.

The Ecore meta-metamodel with the major elements and relations is introduced next, in order to lay the foundations for the practical usage of it in the seminar (see Section 2.2).

The *UML* metamodel is presented first in the remaining part of the lecture, followed by the presentation of all UML models necessary to describe structure and behavior of a system, which is inspired by standard literature (e.g. [20]).

2.2 Structure of the Seminar

The seminar of this course starts with the newly designed *MDA* section. It is split into three parts and follows the general workflow of the *MDA* and the selected tool chain (see Section 3). It consists of the creation of a metamodel of a *DSL*, the definition of visualization properties of used model elements, and the definition of the behavior of modeling tools. This totals three seminars, each 1.5 hours in length. Further steps of the *MDA* approach such as model-to-text transformations are not covered.

The seminar task is kept very open to promote high creativity and interest of the students: The participants may define their own selected *DSL*. An editor to build models based on this metamodel should then be generated by the students. This seminar project has to be realized in a small group of two students.

Workflow and tool chain are presented by a lecturer during the scheduled seminars. The students start with the different steps of the tool chain and have the opportunity to ask questions and get help with arising issues. The students finally have to finalize the seminar project as a home work. A documentation of the results has to be written and submitted as well,both parts form the basis for grading the seminar.

3 Tool Chain

As described in Section 1, there are several tool chains available that have been evaluated in [17]. They have key properties with different advantages and disadvantages. The time required to create an editor for a *DSL* is one of the major aspects of the seminar — MetaEdit+ shows the best result in [17]. On the other hand, MetaEdit+ is based on a proprietary meta modeling language *GOPPRR* and uses a special notation syntax to define the metamodel, and is thus not widely used.

The relatively new Eclipse project Sirius [15, 13] and its closed-source counterpart Obeo Designer [21] are based on the common *EMF* and *GMF*. The Ecore metamodel is thus used to specify the metamodel of a *DSL*. Moreover, there are class diagram editors for Ecoremetamodels available. In contrast to other Eclipse-based *MDA* tool chains, Sirius interprets the model of the graphical editor. Hence, the generation step of *GMF* is not necessary, and it is possible to view the resulting editor directly while modeling it. This accelerates the development cycles of a modeling tool significantly [21] and allows a hands-on success experience for the students. Due to these reasons and the wide usage of Eclipse IDE we decided to use the Eclipse Sirius project for our seminar.

The basis of the tool chain is the latest Eclipse IDE for Java Developers with additional Eclipse Modeling Tools. The components of Sirius project have been installed for the seminar. The *EMF* includes the Ecore metamodel that is used to define the metamodel of the *DSL*. To enable the graphical creation of the *DSL*we also use the "EcoreTools - Ecore Diagram Editor. The Sirius project does not force the use of a specific language to write queries or expressions. We chose the common Acceleo Query Language (*AQL*) and the Object Constraint Language (*OCL*) for these tasks.

Subversion is used as a simple version control system, where each student group gets its own project repository to allow group collaboration. The online learning management system Moodle [22] is an additional environment for the class, where students receive the latest information about the course and can select a seminar group.

Standard PCs with write-protected virtual machines, a Windows operating system and the installed tools of the presented tool chain are used in the seminar. The students may get the complete Eclipse environment with all the installed tools for their own environment alternatively.

Based on our experiences with the first class it is not recommended to deal with more than six or seven student groups in one seminar appointment, because there are still a lot of pitfalls and sources of errors with the chosen tool chain. It is necessary to provide the possibility to answer questions immediately during a supervised group work in the seminar.

4 Seminar Workflow

Inspired by common existing *MDA* workflows [8, 17, 15], the seminar workflow includes three steps: creating a metamodel, defining the graphical representation and defining the editor functionality.

The practical example that was presented in the lecture is shown here again step by step. Another simple example of a university management *DSL* is used in addition to that to demonstrate the *MDA* workflow using Eclipse and Siriusstep by step. It starts with the creation of a new Eclipse project and the additional steps to build a modeling project. The easier steps to create a new "Ecore Modeling Project" are demonstrated afterwards.

4.1 Metamodel Creation

Creating the *DSL* using Ecore as a metamodeling language is the first step of the workflow. A quite simple example of a university management *DSL* is presented (see Figure 3). It includes four classes and one enumeration only. The classes have simple properties such as name or course type. Moreover, the required composition relationship and one additional reference are used to allow different graphical relations of model elements in the final example editor.

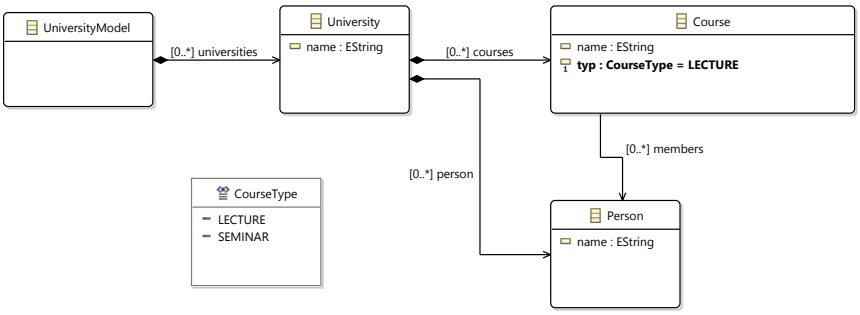


Fig. 3: Presented example of a university management *DSL*.

After this presentation the students have to create their own simple *DSL*. The model validation, the creation of the generator model and the generation of the Eclipse editor components using *EMF* is demonstrated by a lecturer afterwards. As a result, it is possible to run an Eclipse IDE that supports the creation of a model based on the formerly defined *DSL* using a standard tree editor (see Figure 4). At the end of the first seminar appointment the students are thus already able to run their first simple tree editor of a self-defined *DSL*, and can improve their *DSL* until the next seminar individually. This turned out to be a motivational experience for the participants without prior knowledge in model-based software generation.

4.2 Definition of the Graphical Representation

The validation of the *DSL*s defined by the students is the first task in the next seminar. The necessary steps to define a graphical representation of a *DSL* using the Sirius project are presented and demonstrated by a lecturer afterwards.

A new viewpoint specification project is created for this reason and the definition of a visualization of one meta class as a simple graphical node is presented. Sirius does not force the use of a specific language to write queries or expressions to get access to the properties of the metamodel elements, which is necessary, for example, to display the name of an object. Therefore, the essential structure and usage of common used Acceleo Query Language (*AQL*) and Object Constraint

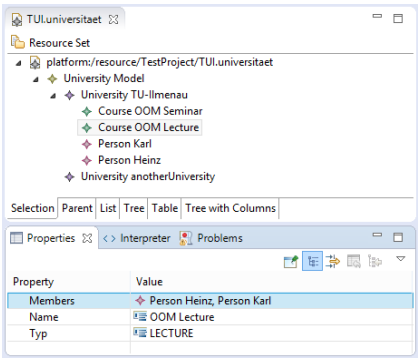
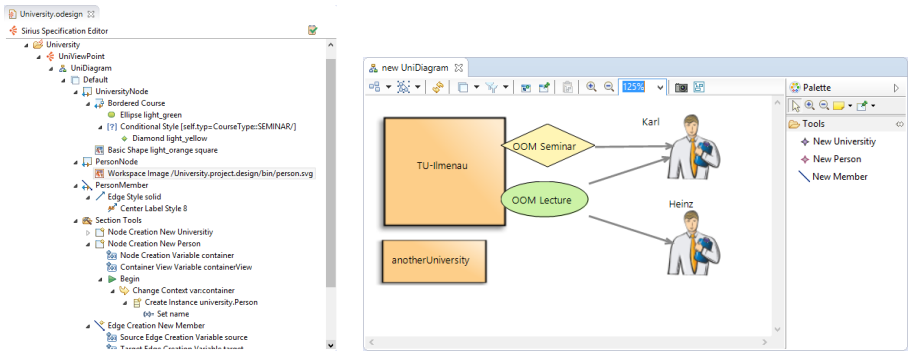


Fig. 4: *EMF*tree view and property sheet for *DSL*s.

Language (*OCL*) are presented to solve this aspect. This is used to demonstrate and implement conditional visualization styles. Additionally, import and use of images is demonstrated. Other useful elements covered are bordered nodes and container nodes to visualize containment relations. The definition of this kind of elements is demonstrated subsequently.

The definition of a relationship between nodes is the final element for the second seminar. There are two different kinds of edges: Element-Based Edge and Relation-Based Edge. The created metamodel includes one reference (see Figure 3) that is used to demonstrate the definition of a Relation-Based Edge.



(a) (b)

Fig. 5: Sirius viewpoint specification (a) and the resulting model editor for the created model (b) (see Figure 4).

Similarly to the first seminar, the students repeat the presented steps using their own *DSL*. At the end of the second seminar date, the students are thus

able to visualize models in a self-defined way, which could be created with the simple tree editor defined during the first seminar date.

4.3 Creation of a Toolbox

The topic of the last seminar is the definition of different tools to create and edit model elements. The steps to create a tool to create nodes are demonstrated first. This includes the necessary steps to set values of attributes of a model element. A tool to create connections between the defined nodes is the last step of this seminar. The final editor is shown in Figure 5.

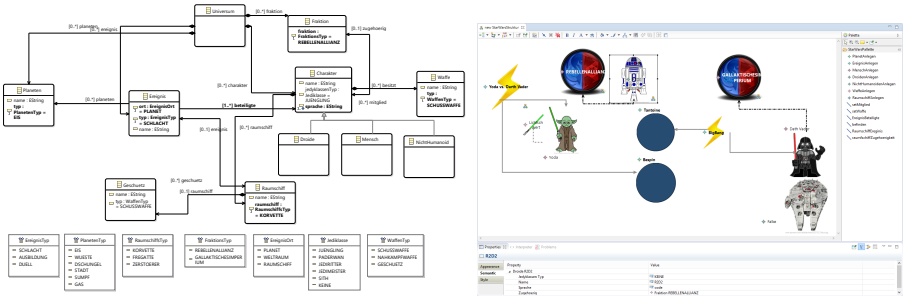
The students should enhance their editor during the rest of this seminar and had the opportunity to ask questions and solve any remaining issues. The editor and the documentation of seminar task had to be finalized as homework. The results are presented in an additional appointment.

The remaining seminar is related to the other lecture parts on details of object-oriented methods. In this part of the seminar, the students should model a system using all structural and behavioral *UML* diagrams.

5 Results

The new course was taught to a smaller number of 14 students (3 information technology sciences, 4 information technology engineering and 5 business information science) for the first time.

The additional *MDA* content of the lecture includes 20 new slides. The new seminar content consists of 30 new slides. The development environment for the tool chain could be reused, using results of former scientific work [23, 24]. The prepared practical example is a simplified one. Hence, it does not need much effort to create it. In the result the effort that went into teaching preparation and teaching itself was comparable to other practically oriented courses. The created slides and the prepared examples can be reused easily in the future.



(a) (b)

Fig. 6: Metamodel of an example *DSL*(a) and the corresponding complete editor including defined tool bars (b).

5.1 Student Results

All students were able to define their own *DSL* and to generate an editor for models based on a self-defined *DSL*. The students were very motivated and interested during the whole semester, and the results were quite impressive. Most of the students spent extra time to present complex and very good model editors. The Figures 6 to 8 display sample results of the seminar.

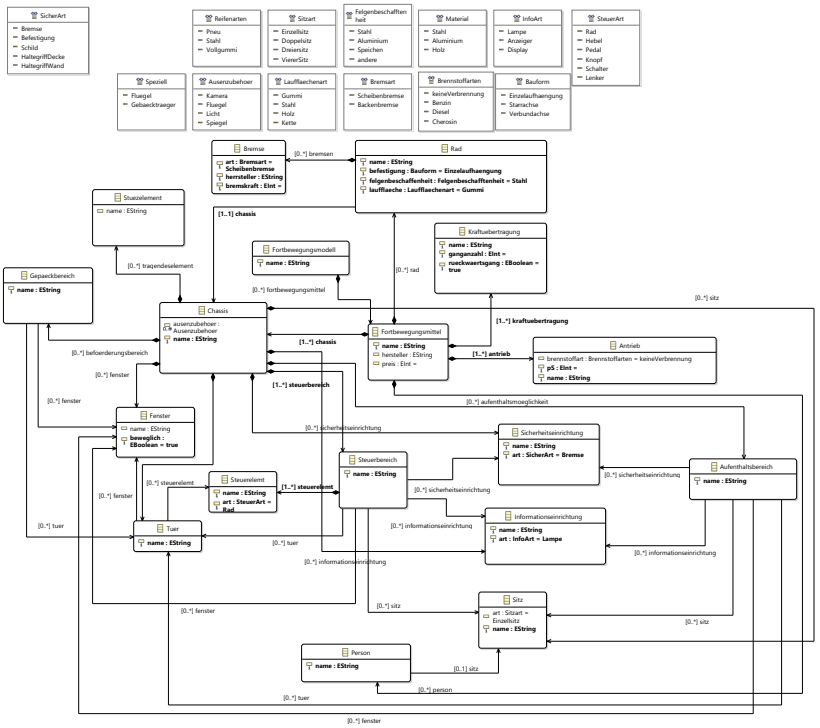
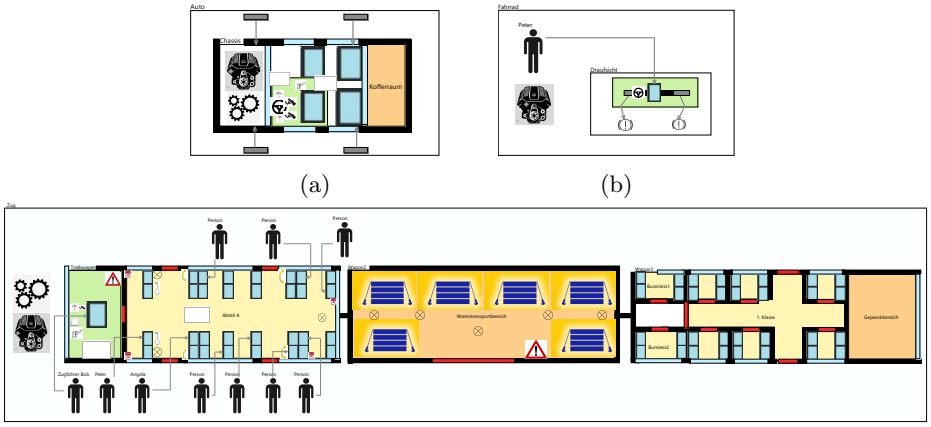


Fig. 7: A slightly more complex metamodel for a vehicle *DSL*.

Figure 6 shows a created model editor for a Star Wars university. It shows the model and the developed tools to create the different model elements. In addition to drawing different model elements, it is possible to set different properties and references to other model elements as it is specified in the defined metamodel using the standard *EMF* editor. This editor is displayed at the lower end of Figure 6b. The corresponding metamodel for this *DSL* is visualized in Figure 6a.

As another example, Figure 7 shows a metamodel for a vehicle *DSL* and some examples of developed models based on this *DSL* (see Figure 8). It is possible to define all vehicles using the defined *DSL*. In Figure 8b the students create a bicycle where Peter is assigned as engine of the bicycle.



(a) (b)

(c)

Fig. 8: Some examples of developed models based on a vehicle *DSL* (see Figure 7). (a): a car model, (b): a bicycle model, and (c): a model of a train.

5.2 Evaluation

In order to verify the success of this course, it was evaluated by the Central Institute of Education of TU Ilmenau based on their regular evaluation scheme. In this section, an excerpt of the results of this evaluation is presented. The results are based on a survey of 12 participants of this course that returned their evaluation sheets.

The required time for preparation and follow-up processing for the whole course is displayed in Figure 9. The mean value of 3.8 hours per week, not including lecture and seminar dates, is just a proper value, regarding 5 credit points for this course, especially considering the significant amount of practical seminar tasks. Some students spend some more time and energy to achieve these terrific results (see Figure 8).

The results of the seminar-specific questions are shown in Figure 10. It shows that the seminar is seen as being useful to understand the complex content that

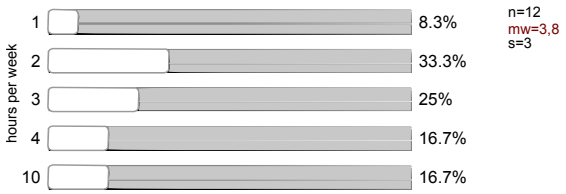


Fig. 9: Required time for preparation and follow-up processing for the whole course. Estimated by the course participants at the end of the semester. (n: number of persons, mw: mean value, s: standard deviation)

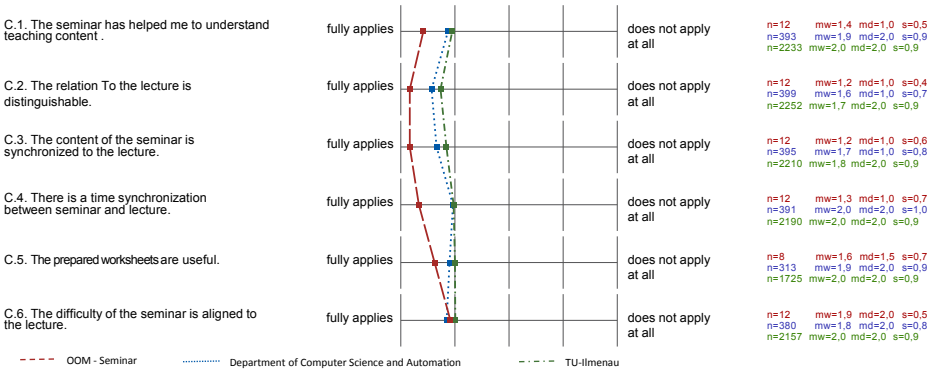
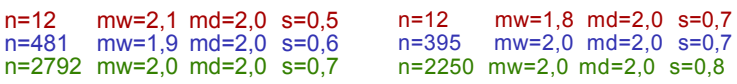


Fig. 10: Detailed evaluation results of the seminar-specific questions. (n: number of persons, mw: mean walue, md: median, s: standard deviation)

was briefly outlined in Section 2.1. The quality of the seminar workflow and the prepared slides and examples were also positively evaluated. These values are (with statistical significance) better than the average results for the whole university and also of the Department of Computer Science and Automation. Even though the topic is quite complex, the difficulty of the seminar was not rated higher than other courses at our university.

The overall grading of the course and the seminar is shown in Figure 11. The course is deemed similar to the average of all other courses of the university. The seminar, however, is graded slightly better than the average of the other courses, despite the higher work load caused by the corresponding tasks.

The students followed lecture and seminar very attentively, showing interest and curiosity. They took a lot of pleasure in practicing the presented *MDA* workflow, which became apparent through the very good results of the seminar presented in this section. As a result, we observed an increasing demand for *MDA*-related topics for student projects and final theses.



(a) (b)

Fig. 11: Overall grading of the course (a) and the seminar (b). (course evaluation results of: red - the seminar, blue - department, green - university, n: number of persons, mw: mean value, md: median, s: standard deviation)

6 Conclusion

This paper presented an experience report of using the *MDA* approach as part of an object-oriented modeling class. The structure of the course, lecture and student project as well as the used tool chain have been presented together with results of student performance and evaluations. The course was rated to be very successful with excellent results of the students and the very positive evaluation responses. In conclusion we can say that the model-driven approach can be taught effectively to computer science students based on the recently developed meta case tools and tool chains. The increased demand for *MDA*-related topics for student projects and Master thesis encouraged us to expand the topic course into an upcoming complete *MDA* course.

References

1. OMG, "Unified Modeling Language (UML), Version 2.5," Object Management Group, Tech. Rep., June 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/PDF
2. ——, "Model Driven Architecture (MDA) - MDA Guide rev. 2.0," Object Management Group, Tech. Rep., 2014. [Online]. Available: http://www.omg.org/mda/
3. ——, "OMG Meta Object Facility (MOF) Core Specification 2.5," Object Management Group, Tech. Rep., 2015. [Online]. Available: http://www.omg.org/spec/MOF/
4. S. J. Sewall, "Executive Justification for Adopting Model Driven Architecture (MDA)," Object Management Group, Tech. Rep., 2003. [Online]. Available: http://www.omg.org/mda/presentations.htm
5. J.-P. Tolvanen and S. Kelly, "Model-Driven Development Challenges and Solutions: Experiences with Domain-Specific Modelling in Industry," in *4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016.
6. OMG, M1 Global Solutions, "MDA Success Story - M1 Global Solutions - Model Driven Software Development and Offshore Outsourcing," 2016. [Online]. Available: http://www.omg.org/mda/mda_files/M1Global.htm
7. OMG, "Omg specifications," 2016. [Online]. Available: http://www.omg.org/spec
8. R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.
9. S. Kelly, K. Lyytinen, and M. Rossi, *Advanced Information Systems Engineering: 8th International Conference, CAiSE'96*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, ch. MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment, pp. –21. [Online]. Available: http://dx.doi.org/10.1007/3-540-61292-0_1
10. IBM, "Rational Software Architect Family," 2016. [Online]. Available: http://www-03.ibm.com/software/products/en/rational-software-architect-family
11. J. Davis, "GME: The generic modeling environment," in *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '03. New York, NY, USA: ACM, 2003, pp. 82–83. [Online]. Available: http://doi.acm.org/10.1145/949344.949360

12. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2009.
13. V. Viyovic, M. Maksimovic, and B. Perisic, "Sirius: A rapid development of DSM graphical editor," in *Intelligent Engineering Systems (INES), 2014 18th Int. Conf. on*, July 2014, pp. 233–238.
14. S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, "Papyrus: A UML2 tool for domain-specific language modeling," in *Proceedings of the 2007 Int. Dagstuhl Conf. on Model-based Engineering of Embedded Real-time Systems*, ser. MBEERTS'07. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 361–368. [Online]. Available: http://dl.acm.org/citation.cfm?id=1927558.1927582
15. Eclipse Sirius Project, "Sirius — the easiest way to get your own modeling tool," 2016. [Online]. Available: http://www.eclipse.org/sirius/
16. Eclipse Foundation, "GMF tooling - model driven architecture approach to domain of graphical editors," 2014. [Online]. Available: http://www.eclipse.org/gmf-tooling/
17. A. El Kouhen, C. Dumoulin, S. Gerard, and P. Boulet, "Evaluation of modeling tools adaptation," 2012. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00706701
18. D. Amyot, H. Farah, and J.-F. Roy, *System Analysis and Modeling: Language Profiles*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ch. Evaluation of Development Tools for Domain-Specific Modeling Languages, pp. 183–197. [Online]. Available: http://dx.doi.org/10.1007/11951148_12
19. V. Pelechano, M. Albert, J. Muñoz, and C. Cetina, "Building tools for model driven development. Comparing microsoft DSL tools and Eclipse modeling plug-ins," in *Actas del Taller sobre Desarrollo de Software Dirigido por Modelos*, 2006. [Online]. Available: http://ceur-ws.org/Vol-227/paper11.pdf
20. C. Rupp, S. Queins *et al.*, *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. München: Hanser, 2007.
21. E. Juliot and J. Benois, "Viewpoints creation using Obeo designer or how to build Eclipse DSM without being an expert developer?" Obeo, Tech. Rep., 2010, Obeo designer whitepaper.
22. moodle project, "moodle," 2016. [Online]. Available: http://www.moodle.org/
23. S. Jäger, R. Maschotta, T. Jungebloud, A. Wichmann, and A. Zimmermann, "Creation of domain-specific languages for executable system models with the eclipse modeling project," in *IEEE Int. Systems Conference (SysCon 2016)*, April 2016, pp. 303–310.
24. ——, "An EMF-like UML generator for C++," in *4th Int. Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016, pp. 309–316.