Challenges and Research Directions for Successfully Applying MBE Tools in Practice

Francis Bordeleau*, Grischa Liebel†, Alexander Raschke‡, Gerald Stieglbauer§ and Matthias Tichy‡
*CMind Inc., Canada, Email:francis.bordeleau@cmind.io
†Software Engineering Division, Chalmers | University of Gothenburg, Sweden, Email: grischa@chalmers.se
‡Institute of Software Engineering and Programming Languages, Ulm University, Germany,
Email: alexander.raschke@uni.ulm.de, matthias.tichy@uni-ulm.de
§AVL List Gmbh, Graz, Austria, Email: gerald.stieglbauer@avl.com

***Abstract*—Model Based Engineering aims to improve efficiency and effectiveness of software engineering. Success in industrial practice of MBE does not only depend on the modeling languages and constructive or analytical approaches, like code generation or model checking. It is also heavily influenced by the quality and, particularly, usability of the selected tools. In this position paper, we discuss challenges experienced in applying MBE in practice both from academic as well as industrial viewpoints. Based on the research challenges, we discuss future research directions to improve the chances for the success of MBE in industrial practice.**

I. Introduction

Model-Based Engineering (MBE) has been successfully used in industrial domains and contexts, though mostly embedded systems [1], over the last decades to manage the increasing complexity of modern software intensive systems, and increase developer productivity and product quality. Since then, many different aspects of this field were addressed in research and industry: creation and representation of models, model transformation, code generation, and tool support to mention only a few.

Several tools (commercial and open source) have been developed to handle textual and/or graphical models. In the beginning, the focus was on fixed modeling languages like UML while in the last decade, the support for modeling domain specific languages (DSL) raised. This resulted in frameworks and tools to generate (textual and graphical) modeling tools out of meta-models, like GMF, Sirius, MPS, and Xtext.

However, despite the undoubtable strengths of MBE, its adoption in the industry, particularly outside the embedded systems domain, often remains limited. A substantial body of work has empirically investigated MBE adoption in industry, both using quantitative methods [1], [2], [3] and qualitative methods [4], [5], [6], [7]. Consistently, obstacles reported in this context are related to modeling tools and their shortcomings. Commonly named shortcomings are, e.g., inadequate usability [1], [8], [9], lack of interoperability [1], [3], [4], [5], [10], high complexity [4], [6], [11], and high training effort [1], [6].

In this position paper, we enumerate several obstacles that are limiting the broad adoption of MBE in industry. From different points of view (two industrial, three academic), we present our experiences regarding the application and development of MBE tools not only, but mainly in industry. In each section, we describe the experienced challenges in specific settings and propose research directions to overcome these limitations.

Section 2 discusses challenges in industrial use while Section 3 discusses challenges with respect to introduction of modeling tools into industrial environments. Section 4 focuses on use of modeling tools in education. Sections 5 and 6 discuss technical challenges in developing modeling tools, particularly, with respect to development of editors. We conclude with a summary.

II. Industrial Use of MBE

In spite of the fact that MBE has been used in the industry since the 90s, the tools have not significantly evolved over the last decades. Existing MBE tools are still much too complex to use and customize for domain specific needs. Moreover, the lack of evolution of the MBE tools in the last years and the fact that MBE tools still lack key capabilities has led several development units to seriously re-consider the use of MBE in spite of the all of the benefits it provides.

A. Experienced Challenges

From our industrial experience, we identify three main areas that need to be improved to enable a broader adoption of MBE tools.

Customization and DSL support. In order to enable the increase of productivity, it is essential that MBE tools support customization and Domain Specific Languages (DSL) to allow adapting the tool to the specifics of the domain/context in which they are used. MBE tools should also allow combining graphical and textual modeling in a complementary manner in a single DSL.

This is one aspect that essentially all commercial UML modeling tools have failed to address. UML tools typically present the whole UML language to the user with very little capabilities to reduce the set of concepts and diagrams to the subset the user needs, and essentially no support for textual modeling. This means that the overall tool environment is much more complex than it used to be. While most people associate this problem with UML, it is not a problem with UML but the UML tools. Team support and collaborative modeling To enable efficient team collaboration, MBE tools

must provide first-class support for the following aspects: versioning, model diff/merge, model review, and document generation. While versioning is well supported, much better support is required for model diff/merge, model review, and document generation. If we compare to coding environments where diff/merge and review are well supported, MBE tools still lack proper support.

Tool capabilities. In spite of years of research in essentially all technical aspects, existing MBE tools still lack capabilities regarding a number of aspects that are considered key (or essential) by software and system engineers, including model-based tracing and debugging, model validation/verification, model executability, and many others (as discussed in the other sections).

While technical evolution is needed regarding many different MBE aspects, we believe that the most effective way to get access to more and better capabilities is to increase technology transfer. In spite of major investment in MBE research over the last decades, very little has been made available in industrial tools. If we want the MBE tools to evolve and provide more capabilities, we need to increase technology transfer. To achieve effective technology transfer, the MBE community needs to collaborate together on the development of an open source MBE tooling platform that can be used by both industrial developers and researchers to develop tools and capabilities. We believe that it is the only pragmatic way to obtain the ever-evolving set of capabilities that we need.

B. Proposed Research Directions

Regarding research directions, we believe that the following three topics are key to broaden the use of MBE in the industry.

- **DSL support**. We need more research focused tool support for DSL, in particular on UML-based DSL so that we can get in the same MBE tools the full benefits of UML as a standard modeling language and the strength of DSL. Also, we need to research focus on how to combine different DSLs in the same modeling environment/tools.
- **Model diff/merge**. We need to increase focus on model diff/merge to address existing issues. In particular, we need scalable model diff/merge support for DSL.
- **Model review**. MBE tools currently dont provide real capabilities for model review. Typically, people do model review by generating PDF documents from models and providing comments in the PDF documents. Firstclass model review is required to enable efficient use of MBE. Finally, we need to better understand how to develop and establish vibrant open source communities to foster innovations and enable efficient technology transfer.

III. Incremental Introduction of MBE

As one reaction to limited MBE tool capabilities, the industry is favoring alternative development strategies such as *agile development* in combination with traditional development technologies. This is despite a common agreement within the modeling community that agility and the *application* of MBE are not contradictory at all. However, there is an observed difference between *applicatio*n (e.g. the act of modeling) and *introduction* of MBE (e.g. learning a modeling language) and according to some experiences, the latter is considered as much more critical within agile environments.

Within agile environments, so-called *development sprints* have shrunk the length of traditional development cycles from months to weeks. The support of frequent product updates, rapid prototyping and continuous integration became an inherent part these sprints. In contrast, most established MBE tools originate from MBEs first hype during the 90s of the previous century when agile environments were less prominent. Many traditional MBE *introduction* strategies are thus not intentionally designed to comply with weekly sprints and due to a limited tool evolvement, MBE tools do not provide an inherent support for an agile-oriented introduction strategy.

A. Experienced Challenges

In contrast to weekly sprints and their central paradigm of a gradual change, traditional strategies for introduction of MBE usually emphasize on a methodological paradigm shift. It is argued that this shift needs extensive MBE training programs for employees and investments in new tool suites both supervised by experienced modeling advocates. These programs are planned for a longer period and are promoted to the management with the promise of a drastic improvement of the development efficiency. The size of these programs, however, turns out to be a classical showstopper in practice for the following simple reason: Upcoming deadlines remain demanding and the agile development team is moving forward, while the modeling advocates are still focusing on various long-term investigations. Once the modeling advocates are presenting the initial MBE-based approach, they usually miss the latest requirements of the next upcoming deadline. The result will thus not be accepted by the development team and prevents the MBE approach to achieve critical momentum.

B. Proposed Research Directions

Instead of a radical paradigm shift from traditional development methods to MBE, there is a strong need for a strategy about a gradual introduction of MBE. In [12], a corresponding MBE introduction strategy is proposed by the definition of so-called MBE *micro injections*. These MBE micro injections are intended to go along agile development processes and the related effort of one MBE micro injection is manageable within one development sprint. Furthermore, the benefits and drawbacks of a single MBE micro injection have to be quantifiable in relation to a traditional approach to convince the development team. If the concrete numbers favor the MBE approach, it has to be capable of being integrated to the main development trunk within the next development sprint. To ensure sustainability, the development team takes over responsibility from the modeling advocate during this integration. Adequate tool support is fundamental to support the

co-existence MBE micro injections and traditional approaches during the gradual introduction phase.

Domain-specific languages (DSLs) are promising candidates to fulfill the requirements of MBE micro injections. However, a DSL design step and the development of a related end-user tool to promote the DSL must be feasible during a sprint to be accepted by the development team. Rapid prototyping must thus come along with a *high degree of automation* (e.g. DSL editor generation) besides further aspects such as a *high tool matureness* (in terms of stability, usability and documentation), *straightforward integration* in existing industrial development tool chains (e.g. Visual Studio), sophisticated *collaboration features* and *composability* of intermediate and distributed MBE approaches across working groups and departments. Despite of promising candidates (e.g. the Eclipse eco-system), none of the established MBE tools are currently mature enough in all mentioned fields. A missed deadline due to this, however, contradicts the vision of a MBE micro injection and undermines the fragile trust between the modeling advocate and the development team.

IV. Usage of MBE Tools in Education

For MBE to be adopted successfully in industry, appropriate university education on the topic is instrumental. Our experience is drawn from four years of teaching a Bachelor-level course on model-based software development at Chalmers University of Technology and Gothenburg University in Gothenburg, Sweden. During each year, approximately 120 to 150 students took this 8-week long course. The course consists of several lectures on the topic of modeling, in particular UML, and a group project. While the syntax of several UML diagrams is introduced, the course focus is on the semantics of the produced models and several different purposes of using models, from informal models for communication purposes to models for code generation.

Three different tools have been used throughout the four course years. In the first year, we used a commercial MBE tool that supports executable models (the academic license agreement prohibits us to name the tool). In the remaining three years, we used Papyrus. In the fourth year, we additionally used Yakindu to allow students to explore executable state machines. From the second year on, the students were required to build a running system from the generated code.

We provided in all four years tool support through a dedicated teacher instead of relying on official tool documentation. However, the amount of support we could provide varied depending on resources.

A. Experienced Challenges

Industrial MBE tools are often reported to contain many bugs and have low usability. Additionally, we often observe that students have difficulties to distinguish actual tool errors from mistakes in their model. For example, when we used code generation students would often attribute errors during code generation to bugs in the modeling tool. When asking the teachers for help, we discovered that the errors were rather related to mistakes they had made in their models. In the course project, students work in teams of 6 to 8 people. Due to the weak support for collaboration, as discussed in Section 2, we observe that students typically resort to work on a single computer or develop a complex manual process to deal with changes on different computers.

Finally, a challenge we encountered regularly is a lack of tool support. Official documentation is often outdated or incomplete. Due to its rapid evolution during the last three years, this is especially true for Papyrus. Additionally, in official documentation for industry-grade modeling tools it is usually assumed that the reader is a modeling expert, which is clearly not the case for students. Similarly, in order to be able to ask questions on a support forum, the students often lack knowledge of how to phrase their questions, or even to distinguish a tool-related question from a language-related question. Therefore, we have during the last years resorted to giving dedicated tool support during lectures and, recently, in the form of short videos that introduce different topics in a concise way.

B. Proposed Research Directions

Several of the challenges we encounter are identical to the challenges in industry. For example, we also see the clear need for *customization and DSL support*. While simplified, education-specific tools are clearly a way to tackle the complexity our students encounter with industrial MBE tools, we often experience that these tools are built with a specific course design in mind. Hence, they might be restricted to a certain process and diagram types. Therefore, we believe that to address complexity of MBE Tools, tool developers and researchers should investigate how industrial tools can effectively be customized in a quick fashion without expert tool knowledge. In particular, it should be possible to adapt a tools customization after it has been installed, e.g., to re-distribute it to students or employees when changes are necessary. Furthermore, we need better *team support and collaborative modeling* to allow students to effectively work in groups.

Additionally, we also see multiple directions for research aimed specifically at modeling education. Based on the challenge to distinguish tool errors from mistakes resulting from a lack in understanding, we propose to research how students can be provided with feedback on their models in a constructive way. Similar work has been conducted in the area of programming education, e.g., in [13], [14]. It is important to note that this research is not only restricted to tool features or automated feedback, but could also be in terms of novel course designs or processes that ensure that students receive sufficient and constructive feedback, such as in [14].

V. Development of more usable model transformation tools

We discuss the challenges during development of model transformation tools with a higher usability using Henshin as a case. Henshin is an Eclipse-based Model Transformation

Language based on the Graph Transformation formalism. It uses a declarative way to specify model transformations by defining pre- and postconditions. Henshin and predecessors have been around since 2009 and heavily use tooling of the Eclipse Modeling Framework ecosystem, e.g., generation of graphical and tree editors. Recently, we developed several usability improvements with respect to syntax checks when executing model transformations from plain Java code, a textual syntax, and generation of transformation rules [15].

A. Experienced Challenges

A major challenge during the development of Henshin is a rather big gap between the abstract syntax and the concrete syntax. Graph transformation based model transformations define a precondition (called Left Hand Side) and a postcondition (called Right Hand Side) on the to-be-transformed model part. Henshin and most graph transformation languages use a shorthand notation as concrete syntax, where the left hand side and right hand side is combined and the changes to transform the model from the pre-condition into the post-condition are explicitly annotated. However, the abstract syntax of Henshin is based on the explicit distinction between the left and the right hand side. This leads to a big gap between the concrete syntax and the abstract syntax and is out-of-the-box not well supported by frameworks for graphical. Hence, we had to add many manual changes to the generated code of the graphical editor. First, these manual changes are difficult to correctly do and require a rather big effort. Second, these manual changes complicate further usage of the frameworks code generation capabilities and, thus, hampers development.

Furthermore, one of our recurring experiences in MBE tools is that they often focus on functionality. Particularly, we believe that we, as a community, much more value functionality over usability. A researcher will get a paper on new language features or performance improvement accepted, but work on simply improving usability of existing tools is difficult to measure and to publish. For example, the MODELS 2017 main program does only have 2 papers which cover usability, 7 mention it somewhere as something important or future work, and 27 ignore it. Hence, we believe there is a systematic bias against usability improvements to MBE tools. This has been also our experience, while working in industry, developing modeling tools.

With respect to model transformations, there have been numerous papers on search plan optimization and other kinds of automatic performance improvement. However, to the best of our knowledge there does not yet exist an easy way for developers of graph transformations to understand how transformation rules are executed and the specific impact of changes in their transformation rules have on the performance of model transformation executions.

Similarly, often modeling tools do not provide easy to use debugging capabilities. Particularly, for declarative model transformation languages like Henshin, it is very difficult for developers to understand why a model transformation rule does not work as intended, particularly, since the declarative nature makes it more difficult to understand how rules are executed and the sequences of actions are not explicitly expressed by the developer but automatically inferred and optimized by the engine.

B. Proposed Research Directions

We see a need for a more systematic usability engineering support in MBE tool development. Particularly, we are wondering whether existing usability engineering techniques [16] can simply be reused, and are maybe often ignored, or specific techniques for MBE need to be researched. For example, there could be a significant difference in usability requirements between the developer of MBE tools and expert users on the one side and non-expert users and domain experts on the other side.

Furthermore, while we developed a textual editor for Henshin [15] for an easier specification of transformation rules, we still see the value of graphical editors. However as discussed in Section 2, we need both types of editors highly integrated, as shown in an early prototype [17], and the corresponding development frameworks which makes those editors easy to develop.

VI. User Experience of Graphical Modeling Tools

The user experience plays an important role in the acceptance of a tool. If the creation and editing of a model is painful and time-consuming, the advantage of MBE by shortening the development time can melt dramatically.

Besides MBE tools for fixed models like UML, a set of frameworks were built to support development and/or generation of graphical modeling editors for a DSL. These frameworks (e.g. GMF, Sirius, MPS, and graphiti) reduce the effort for the developer tremendously. Unfortunately, these frameworks are optimized for fast development, and thus helpful for the introduction of MBE (see Section 3), but not for a comfortable and user-optimized usage. In the following, we focus on our experiences with these generator frameworks.

A. Experienced Challenges

One of the main problems is the correct-by-construction approach of the underlying GEF framework. This means that any editing step must result in a (syntactically) correct model. Obviously, this is useful from the developer view, because any diagram is a representation of the model. It is not necessary to deal with two kinds of models: one (possibly incorrect) for the graphical editing and one for further processing.

Considering the users view, it is often painful to ensure that any editing operation does not contain any intermediate, incomplete model. In the simplest case, the tool enforces the user to do some editing in a specific order. For example, if an edge exists between two nodes and the user wants to add another node in the middle of the edge, then, first, she has to create the new node, attach the start or the end point of the existing edge to the new node and then create a new edge, correspondingly. It is not possible to first detach the edge from one of the existing nodes.

Another example is to remove the surrounding state of a hierarchical state in a state chart, i.e., to move the contained states one level up. In most tools deleting the surrounding state results in a deletion of the containing states as well. At least, any edge to or from the surrounding state is removed, because otherwise these edges are not connected to two nodes. In this case, the user has to redraw the edges again or she has to move all contained states to the same level as the surrounding state, modify the edges, remove the surrounding (and now empty) state, and rearrange the states to their original position.

Problems like these are the reason why simple drawing tools like Microsoft Visio or even Powerpoint are often preferred for creating graphical models. These tools do not restrict the users in their workflow. The avoidance of incomplete graphical representations of models is in contrast to the textual representation. In a text editor, most of the time, the actual text does not represent a correct model. Only at specific times, when the text is parsed, it is necessary that the text is syntactically correct.

The need for incomplete graphical models in MBE tools becomes even larger, if textual and graphical models are used in a hybrid view, where the underlying model can be edited either by using a textual or graphical representation and it should be possible to change between these views at any time as discussed in Section 2.

B. Proposed Research Directions

In order to improve the user experience of graphical editing tools, we propose the following three research directions:

- Comprehensive studies with industrial users have to be conducted to gain more insight into the required features enabling a faster editing of models.
- Based on the results of these studies, improved prototypes of editors supporting different kinds of graphical editing tasks have to be evaluated with industrial users.
- Finally, frameworks incorporating the results of the preceding studies have to be developed.

VII. Conclusions

In this position paper, based on our cumulative experience in industry, research, and education, we present current challenges related to MBE tooling.

We discuss the introduction of MBE tools in industry, which is often hindered by outdated assumptions on the process and a slow return on investment. With respect to tool usage in industry, we see that there are several key features currently missing that effectively prevent the use of MBE tools. These are, e.g., a lack of collaboration facilities and customization support. Mirroring the industrial use of MBE tools, several similar challenges can be seen in modeling education, which suggests that the topic of MBE education should not only be studied in isolation. Using and building MBE tools in an academic environment, we observe that the user experience and usability of existing tools and tool-building frameworks are low, but at the same time that academic policies do not effectively encourage improvements within academia.

To tackle existing challenges, we outline directions for future work in the area of MBE tooling. These directions can be separated into directions for research, e.g., improved model diff/merge capabilities and novel MBE introduction strategies, directions for industry and communities like improved technology transfer and initiatives to start and foster open source communities around MBE, and directions for academic policy, e.g., an increased focus on tool creation or improvements in tenure procedures.

Acknowledgment

This work was partially funded by the German Research Foundation (DFG) as part of the DFG Priority Programme 1593 (TI 803/2-2 and TI 803/4-1).

References

[1] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain - an industrial survey on the state-of-practice," *Software & Systems Modeling*, Mar. 2016.

[2] L. T. W. Agner, I. W. Soares, P. C. Stadzisz, and J. M. Simão, "A brazilian survey on UML and model-driven practices for embedded software development," *Journal of Systems and Software*, vol. 86, no. 4, pp. 997–1005, 2013.

[3] A. Forward and T. C. Lethbridge, "Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals," in *International Workshop on Modeling in Software Engineering, MiSE 2008, Leipzig, Germany, May 10-11, 2008*, J. M. Atlee, R. B. France, G. Georg, A. Moreira, B. Rumpe, S. Völkel, and S. Zschaler, Eds. ACM, 2008, pp. 27–32.

[4] S. Kirstan and J. Zimmermann, "Evaluating costs and benefits of model-based development of embedded software systems in the car industry–results of a qualitative case study," in *Proceedings Workshop C2M: EEMDD "From code centric to model centric: Evaluating the effectiveness of MDD" ECMFA*, 2010.

[5] P. Baker, S. Loh, and F. Weil, "Model-driven engineering in a large industrial context - motorola case study," in *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings*, ser. Lecture Notes in Computer Science, L. C. Briand and C. Williams, Eds., vol. 3713. Springer, 2005, pp. 476–491.

[6] J. E. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, R. N. Taylor, H. C. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 471–480.

[7] J. E. Hutchinson, M. Rouncefield, and J. Whittle, "Model-driven engineering practices in industry," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, R. N. Taylor, H. C. Gall, and N. Medvidovic, Eds. ACM, 2011, pp. 633–642.

[8] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernández, B. Nordmoen, and M. Fritzsche, "Where does model-driven engineering help? experiences from three industrial cases," *Software and System Modeling*, vol. 12, no. 3, pp. 619–639, 2013.

[9] S. Karg, A. Raschke, M. Tichy, and G. Liebel, "Model-driven software engineering in the openetcs project: project experiences and lessons learned," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016*, B. Baudry and B. Combemale, Eds. ACM, 2016, pp. 238–248.

[10] P. Mohagheghi and V. Dehlen, "Where is the proof? - A review of experiences from applying MDE in industry," in *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*, ser. Lecture Notes in Computer Science, I. Schieferdecker and A. Hartman, Eds., vol. 5095. Springer, 2008, pp. 432–443.

[11] A. Forward, O. Badreddin, and T. C. Lethbridge, "Perceptions of software modeling: a survey of software practitioners," in *Proc. of C2M:EEMDD*, 2010.

[12] G. Stieglbauer and I. Roncevic, "Objecting to the revolution: Model-based engineering and the industry - root causes beyond classical research topics," in *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2017, Porto, Portugal, February 19-21, 2017.*, L. F. Pires, S. Hammoudi, and B. Selic, Eds. SciTePress, 2017, pp. 629–639.

[13] M. J. Lee and A. J. Ko, "Personifying programming tool feedback improves novice programmers' learning," in *Proceedings of the Seventh International Workshop on Computing Education Research, ICER 2011, Providence, RI, USA, August 8-9, 2011*, K. Sanders, M. E. Caspersen, and A. Clear, Eds. ACM, 2011, pp. 109–116.

[14] A. Vihavainen, M. Paksula, and M. Luukkainen, "Extreme apprenticeship method in teaching programming for beginners," in *Proceedings of the 42nd ACM technical symposium on Computer science education, SIGCSE 2011, Dallas, TX, USA, March 9-12, 2011*, T. J. Cortina, E. L. Walker, L. A. S. King, and D. R. Musicant, Eds. ACM, 2011, pp. 93–98.

[15] D. Strüber, K. Born, K. D. Gill, R. Groner, T. Kehrer, M. Ohrndorf, and M. Tichy, "Henshin: A usability-focused framework for EMF model transformation development," in *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*, ser. Lecture Notes in Computer Science, J. de Lara and D. Plump, Eds., vol. 10373. Springer, 2017, pp. 196–208.

[16] A. Seffah, J. Gulliksen, and M. C. Desmarais, *Human-Centered Software Engineering - Integrating Usability in the Software Development Lifecycle*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[17] S. Maro, J. Steghöfer, A. Anjorin, M. Tichy, and L. Gelin, "On integrating graphical and textual editors for a UML profile based domain specific language: an industrial experience," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, SLE 2015, Pittsburgh, PA, USA, October 25-27, 2015*, R. F. Paige, D. D. Ruscio, and M. Völter, Eds. ACM, 2015, pp. 1–12.