Simplify model design in MDE approaches

Asbathou Biyalou-Sama

Simplify Model Design in MDE Approaches

Asbathou Biyalou-Sama
Univ. Lille, CNRS, Institut Mines-Télécom, UMR 9189 CRIStAL, Axellience/Ansys
Lille France
asbathou.biyalousama@univ-lille.fr

ABSTRACT

Many academic works prove that Model-Driven Engineering (MDE) improves the quality of the application's production in software engineering. However, its adoption remains limited for several reasons: the complexity of the tools makes them less productive, the modeling languages and the mechanisms manipulating the models are hard to grasp. In that sense, low-code platforms are an interesting proposition. These low-code platforms, despite their simplicity of use, do not allow the design of large systems because of the opacity of the various artifacts (models, code, generation...). In our work, we want to improve the adoption of MDE models, tools, and mechanisms in application design process through a more simplified interaction. Rather than constraining the developer in a closed environment as it is the case with the Low-code platforms, we wish to allow him to have access, if he wishes, to models as well as the generation mechanisms of the application.

We propose to build applications that are instrumented with modification actions on each element of their user interface. The applications are generated from models. A developer can modify or develop his application directly from the UI by calling upon the added modification actions. This makes his work easier by quickly locating (in the UI of the produced application) where to make the modifications and directly visualizing the result, rather than manually modifying each part of the model concerned. The instrumentation is based on a descriptive language for possible modification actions and a mechanism for injecting this language into the application generation chain. We have designed a demonstrator allowing us to conduct experiments, with various audience, in order to validate our work.

CCS CONCEPTS

• **Software and its engineering → Software design engineering** • **Software and its engineering → Integrated and visual development environments** • Software and its engineering → Visual languages

KEYWORDS

Software Engineering, MDE, Modeling interface, Tooling

ACM Reference format:

Asbathou Biyalou-Sama. 2022. Simplify Model Design in MDE Approaches. In *Proceedings of ACM Models conference (MODELS'22). ACM, New York, NY, USA, 6 pages.*
https://doi.org/10.1145/3550356.3558515

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10…$15.00
https://doi.org/10.1145/3550356.3558515

1 Problem

To design software applications, there are several design approaches with some of them based on models. We can cite generative programming [6], Model-Driven Engineering (MDE) [9], domain-specific languages [7], Model-Integrated Computing (MIC) [22], software factories [12], etc. A model is a simplified representation of an aspect of a system (note that the system can also be a model) that can be used instead of the modeled system, for example to reduce costs or decrease risk [18]. In this work, we are interested in Model-Driven Engineering (MDE). The MDE can be used to generate code from a descriptive model. In this case, the MDE can be decomposed in a chain made of five points (figure 1) which are: (1) modeling, (2) transformation, (3) generation, (4) maintenance and (5) execution.

(1) The application is first designed as conceptual models using some design patterns where they are applicable. These models allow to conceptualize and build the architecture of the application. They allow to focus on the essentials without worrying about implementation details or technical issues. However, at this stage, modeling tools and languages can be an obstacle to the adoption of MDE, as they are often complex to master [4]. (2) Once the application has been designed, the models can be used for generation, analysis or simulation. And for these uses, models are, sometimes, transformed to more adapted models. (3) From these models, all or parts of the application can be generated [15]. Model-based generation improves the quality of the software, on one hand by producing an application that conforms to the design models and on the other hand by benefiting from the expertise of technical experts encapsulated in the generation mechanisms. But these generation mechanisms are complex to implement, which makes the task of designing the

generation reserved only for modeling initiated and MDE experts. (4) In maintenance, the models make it possible to better understand the software architecture in order to locate the changes to make. Unfortunately, models quickly become obsolete, because in practice, it often happens that changes are made directly in the code without updating the initial models [17]. We have modeling approaches like in [16,25] which tend to resolve this problem, by using an intermediate textual representation of the model. This textual representation is always updated with the graphical model but for the moment is not reflecting changes done in the Java or PHP generated code. (5) The generated code can be executed.
As we can see, the use of MDE has many advantages to design software applications with a better quality, but requires a relatively long learning curve: it is necessary to know the modeling languages, to know how to use tools [5] and to understand the generation processes and the various mechanisms [4].

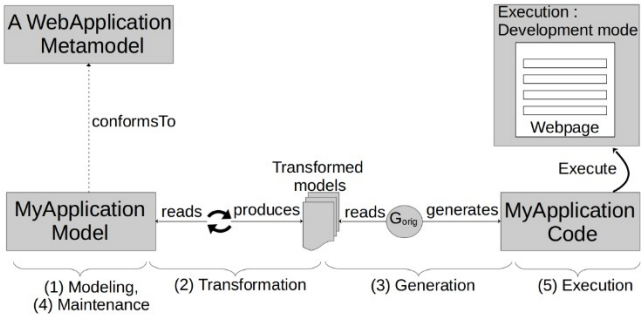

Figure 1: **The five main points of a MDE approach for a code production chain**

Designing applications using the MDE approach can be difficult as illustrated by the problems raised above. But there are other approaches to design applications that are easy to use. Among them, there is the Low-code/No-code approaches.

Low-code/No-code platforms have emerged in recent years [10,26]. They allow any profile/user, especially non-programmers, to create software applications by generating the different parts of the application. These platforms allow to design software applications through click-and-drop, add and modify actions in a dedicated environment. They allow not to write code or very little code to develop a software application [21]. Low-code platforms provide simplicity of use and facilitate the design and development of software applications. Despite the simplicity that these platforms bring, they have some drawbacks [8,14,21]: the developed applications constitute black boxes in which the generation mechanisms/processes are completely opaque to the user and very poorly configurable. The developed applications almost always run only in the dedicated environment of the platform or through third-party services linked to the platform. These software applications thus live in a very closed environment [21]. These limitations lead to a certain frustration for the designer and developer initiated respectively to programming and modeling and compromise the extensibility of tools and processes.

Even if the Low-code movement and the MDE community are trying to get closer [8], Low-code platforms do not offer the ability to manipulate models in the MDE sense (conforming to languages/metamodels), nor to exploit the possibilities offered by the MDE mechanisms (essentially based on Eclipse tools and technologies). Low-code platforms (LCDP) do not allow to take advantage of the code generation work done in the MDE community.

We end up with the power of models and generation processes on one side and the simplicity of using LCDP on the other side. However, for people initiated to programming and modeling, combining these advantages in a methodology or a tool would be great.

Our problem can be expressed as follows: Can we propose an approach combining both the ease of design of low-code (drag and drop and interactions on an interface close to WYSIWYG) and the power of MDE (abstraction, multi-target source code generation, transformation, open tools ...).

2 Proposed Solution

To deal with the complexity related to most of MDE tools and to provide answers to the various problems mentioned above, we propose simple ways to interact with MDE models and mechanisms. This proposal (figure 2) allows the designer to interact with the User Interface of the running application, in order to trigger MDE actions and mechanisms, including modification of the application model, generation mechanisms and re-execution of the application. In our proposal, just for sake of simplicity and tool accessibility, we are working with web applications. But our proposition is also targeting other kind of application. As in figure 2, we are manipulating an application obtained from a web application metamodel, the application here always makes reference to a web application.

The originality of the proposal is that the designer acts on the real application at runtime and not on a static model of it. To do this, we propose to instrument the user interface of the web application, so that an element of the interface that the designer wish to modify is provided with contextual actions allowing these modifications. These actions are of the CRUD (Create Read Update Delete) type and allow, depending on the intention, to create, modify and delete the element of the interface. These actions have an impact, not only on the user interface, but also on the application model. Moreover, depending on the application model, a CRUD action on a user interface element can trigger several modifications in the model. Thus, the designer acts directly on the application architecture through the user interface.

In order to prevent an ordinary user from modifying the application, the modification actions will only be available on the user interface when the application is in 'development' mode. In 'production' mode, the application will not have any actions (not even the action code). These two different modes are obtained thanks to the MDE principle: we generate 'development' code and 'production' code from the same input model. In development

mode, the application augmented with appropriate actions will be obtained by injecting a description of the actions into the generation process. This description will be made thanks to a description language that we propose. This description language will describe the possible CRUD actions on each concept and how these actions are executed.

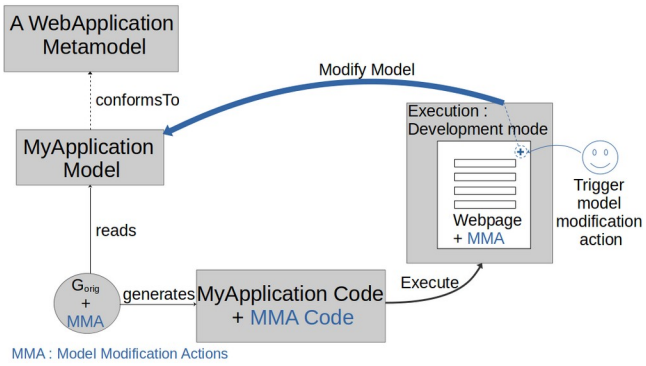

Figure 2: **Our proposal: model modification from the application user interface**

In summary, our proposal allows, from an action of the designer on the user interface of an application, to generate a set of actions on this application: the modification of the model; the automatic generation of the code containing the modifications made to the model; the re-execution of the code of the augmented application.

3 Related Work

In this section, we will present some related works according to two aspects: (i) those which deal with notions and technologies related to MDE close to our approach and which will be used in the realization of our approach and (ii) the works which try to answer the more general question of the complexity during modeling.

3.1 Notions and technologies

In the notions and technologies that enter in one way or another in our approach, we have: Live Modeling/Live Programming and Models at Runtime in relation with Adaptive Systems.

According to [1], "in the context of Model-driven Development (MDD) models can be executed by interpretation or by the translation of models into existing programming languages, often by code generation". Live Modeling is then a technique allowing users of executable modeling languages to edit models during their execution, thus helping to better understand the impact of design choices.

In [24], the authors make a parallel between Live Modeling and Live Programming, Live Modeling being seen as a transposition of the concepts of live programming to modeling languages. They bring the possibility to edit live models written with these modeling languages. They focus on strategies to instrument modeling languages in order to add live modeling to them. The modification of models during execution becomes possible, without necessarily stopping the execution.

In Live Modeling, the question of interaction with the models is not addressed. The model is modified and we are interested in how to apply the changes during the execution of this model. And this regardless of the means by which the model has been modified. In our work, we aim more at improving the ways the designers interact with the model. In order to update the running application after the model is updated by our proposal, we will use the strategies of dynamic update offered by the live modeling.

Blair et al [3] define models@run.time as a causal self-representation of the associated system, which focuses on the structure, behavior, or goals of the system and can be manipulated during runtime for specific purposes. Models@run.time can thus be used to maintain different aspects of a system. They have been used very often in the context of work on self-adaptation or the evolution of software applications [2]. The works around self-adaptation use models@run.time to have an updated view of the system on which the changes of the real system can be followed in real time (Monitoring) and conversely the real system can also be updated (Execute) from this view on which analyses have been performed [23]. This corresponds to the famous MAPE-K feedback loop [11].

The interest of this work in our thesis is to exploit on the medium term, the analysis possibilities offered by models@run.time. We will be able to analyze the impact of the changes on the models and at different levels of our system. Our goal is not to realize self-adaptive systems or to propose new strategies around them. We are much more interested in the user interaction with the models and in the complexity during the exploitation of MDE tools.

3.2 Works around complexity in modeling

Concerning the complexity during modeling, various works have been carried out to first measure this complexity and evaluate the reasons for this complexity [5,19]. Other works have gone further by proposing approaches to reduce this complexity [13,20] and lead to a better adoption of MDE in application design.

Hill [13] measures the modeling effort when using domain-specific modeling languages (DSMLs). He evaluates this effort in terms of user actions. He proposes metrics to make this evaluation, and then looks at various techniques to reduce this modeling effort. Its work is based on external artifacts (model observers, model decorators and model solvers) present in DSML environments that can be used to reduce the modeling effort.

Pourali, in his thesis [20], does an interesting analysis of the difficulties that modelers encounter when using UML modeling tools. In particular, he points out that one of the challenging tasks is to locate, understand and fix errors in a model. From these analyses, he extracts a set of recommendations for modeling tools in order to reduce the difficulties while using these tools. He proposes reduction techniques and makes implementations to verify some of these proposals.

We can see from the various works that the question of the complexity of modeling tasks is real and needs to be addressed. Many works focus either on modeling languages or on modeling tools. But very few address this issue from the point of view of the goal of the modeling tasks we perform, of the reason why we model. In our case, the aim of the designers is to design a software application. We are thus specifically interested in the modeling tasks that are performed in the design of a software application and we look for ways to simplify these tasks.

4 Plan for Evaluation and Validation

The evaluation and validation of our work is done in two steps: 1) the realization of a demonstrator and 2) the validation through experiments.

A first demonstrator has been realized. It allows to validate that it is possible to interact on the model of the application from the UI of this application, to re-generate and re-execute the code after the modification. The demonstrator is described in section 6: Current Status.

For the experiments, a first step will be to define evaluation criteria. These criteria will be based on questions we asked ourselves during the design of the demonstrator and on some tests done by first-time users in the laboratory.

The objective of the experiments is to validate our proposal. Is it easier for an application designer to act directly on the application's interface compared to a classic MDE approach? Is it more efficient? Is the possibility of accessing the models and generation mechanisms useful for application designers compared to low-code tools that do not allow it?

The demonstrator will be available to two different user profiles: learner profiles which are beginners in modeling and professionals more experienced in modeling. We will observe the users in the execution of modification tasks (addition, deletion of elements...) that we will ask them to complete.

Regarding the learners, we will separate them into two groups, one will have to perform model modification tasks without using the demonstrator, the other will perform the same modifications, but using the demonstrator, and thus acting directly on the UI. We will be able to evaluate in which situations they can manage more easily, if they understand more easily the application architecture and if our approach helps them to integrate a new project.

Expert will probably want to act directly on the models, the source code, or the various elements of the generation chain. As for the beginners, we will propose to two groups of experts to realize modifications of the application, for the ones in a classic MDE approach, for the others through our demonstrator. This will allow us to validate the fact that even experts in modeling appreciate using the interface to modify the application while having a look at the evolution of its models.

We also want to evaluate and validate the implementation of the instrumentation. To do so, we will ask experts to instrument an MDE-based generation application other than the one we used. We want to evaluate the speed with which this implementation can be done and the efficiency of this instrumentation. The validation of such properties is crucial as they are determinant for the adoption of this type of model-based approaches in projects.

5 Expected Contributions

One of our objectives is to improve the adoption of MDE (models, tools, mechanisms) in the design of applications by trying to reduce the learning curve. We propose to act on models of an application by interacting with its user interface. This approach simplifies the understanding of complex models and MDE mechanisms. The direct interaction with the application interface can be found in Low-Code platforms. However, these platforms are often closed and opaque. By proposing a reproducible and easy to implement approach, we want to allow the designer to always have access to the models, code and generation mechanisms of his application.

Our approach favors model modification over code modification, making the model the preferred "source of truth". This could lead to reduce model obsolescence.

We will also need a description language for the modification actions proposed to the user and a mechanism to inject the elements of this language into the application generation process. In a first step, we will define an action description language specific to the design of web applications. By projecting this work in another application domain, we will extract a set of generalizable criteria allowing the definition of a higher-level action description language, to finally define an action description language metamodel. In parallel, we will contribute to the definition and implementation of an efficient and fast injection mechanism. The dynamic re-execution of the application will be based mainly on existing work on Live Modeling.

As stated in section 4, our work will be validated through case studies and experiments. Then, we will extract from these case studies and experiments a set of use cases in which our approach has the most significant impact compared to other existing approaches.

6 Current Status

6.1 Work Done

One of the main tasks that we have performed is the development of a demonstrator to implement our solution.

To realize this demonstrator, we started from an existing application generator, WebSiteGen, built on the principles of MDE. WebSiteGen allows to generate websites from a model describing the site to generate. The generated code is in React and Spring.

We have modified this generator so that it produces 'instrumented' applications with actions allowing to modify the model from the UI of the application. Currently, this modification on the generator is 'manual': we have added the necessary code in the generation templates in an ad hoc way.

The demonstrator is composed of the modified generator and a basic model describing a web application with a simple home page (figure 3).

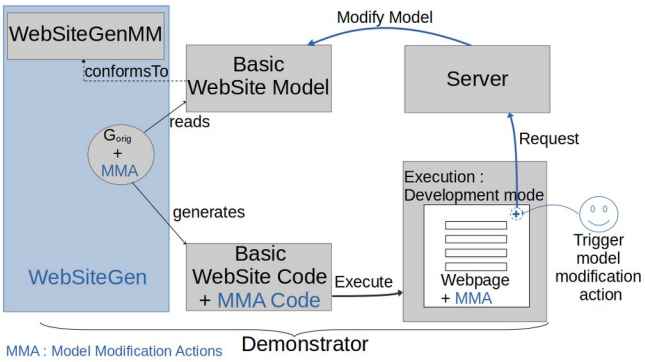

Figure 3: **Demonstrator made of an existing MDE application generator, a basic model, and 'Model Modification Actions' (MMA). A server is used to receive modification requests and to modify the model.**

With the demonstrator, the (generated) page of the web application contains actions allowing the addition and deletion of pages or fields in a form. These actions are accessible from the page when the application is running and as we mention already are contextual. In figure 4, we are displaying 'add field' and 'update field' actions (blue buttons) on a form page. When an action is triggered, a request is sent to a server, which processes the action and modifies the model. The MDE chain re-generates the code, the application is restarted, and the user can then see the result of his action.

As the code is re-generated each time the model is modified, the 'manual' code of the actions is inserted into the generation templates. Thus, the actions are also re-generated.

Thanks to our demonstrator, we have a first complete loop allowing us to test various properties related to model modification from the user interface of a running web application.

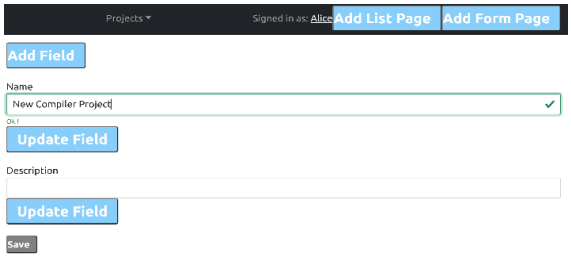

Figure 4: **A form page with some modification actions (blue buttons)**

6.2 Work to be done

For the next step, we will use and submit our demonstrator internally within the laboratory in order to get feedback. Then, tanks to discussions and observations, we will define and refine the evaluation criteria that we will use later. These evaluation criteria will be used to prepare a more formal experimentation with a larger audience outside the laboratory. We will thus define the tasks to be carried out during the experiments and the questions to be asked in order to reach our objectives. In parallel to the experiments, we will start working on the definition of the description language.

In the actual demonstrator, we manually define the actions and changes that are performed on the model. Our goals being to be able to easily express and use these actions, a next step will be to set up a description language for these actions. For this, we will use the demonstrator to extract a base of simple and minimal actions. We will define (in a formal way) a semantic allowing to express these actions. Thus, the actions that we have written manually in the demonstrator will be defined using the description language.

We also plan to study the way the MDE compilation chain is done: What are the best solutions to integrate the compilation language, to generate action code, to re-execute the generated application? For that, we will exploit works done around MDE code generation.

We will explore how Live Modeling can be used to provide the best dynamic execution strategy to re-execute the application.

Once the compilation chain is finalized, we will study how to use more complex and more abstract modification actions, but also the possibility of having modification actions on elements of the model that have no visible counterparts in the page. Among these elements we can mention the architecture of the cache in the services, the configuration of the different services, the internal processing operations, the relations between entities in the database... A possible way to explore is to make these elements visible in a 'configuration' page.

While improving the language, we will publish our results. We will carry out a compilation and organization of our results in order to write our PhD thesis.

ACKNOWLEDGMENTS

This work is supported by IRCICA research institute. This work also benefits from the financial support of the Hauts-de-France Region.

REFERENCES

[1] Mojtaba Bagherzadeh, Karim Jahed, Benoit Combemale, and Juergen Dingel. 2021. Live modeling in the context of state machine models and code generation. *Softw Syst Model* 20, 3 (June 2021), 795–819. DOI:https://doi.org/10.1007/s10270-020-00829-y

[2] Nelly Bencomo, Sebastian Götz, and Hui Song. 2019. Models@run.time: a guided tour of the state of the art and research challenges. *Software & Systems Modeling* (January 2019). DOI:https://doi.org/10.1007/s10270-018-00712-x

[3] Gordon Blair, Nelly Bencomo, and Robert France. 2009. Models@ run.time. *Computer* 42, (November 2009), 22–27. DOI:https://doi.org/10.1109/MC.2009.326

[4] Francis Bordeleau, Grischa Liebel, A. Raschke, Gerald Stieglbauer, and Matthias Tichy. 2017. Challenges and Research Directions for Successfully Applying MBE Tools in Practice. *MDETOOLS 2017* (2017). Retrieved July 19, 2022 from https://research.chalmers.se/en/publication/500685

[5] Nelly Condori-Fernández, Jose Ignacio Panach, Arthur Iwan Baars, Tanja Vos, and Óscar Pastor. 2013. An empirical approach for evaluating the usability of model-driven tools. *Science of Computer Programming* 78, 11 (November 2013), 2245–2258. DOI:https://doi.org/10.1016/j.scico.2012.07.017

[6] Krzysztof Czarnecki. 2005. Overview of Generative Software Development. In *Unconventional Programming Paradigms* (Lecture Notes in Computer Science), Springer, Berlin, Heidelberg, 326–341. DOI:https://doi.org/10.1007/11527800_25

[7] Arie van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.* 35, 6 (juin 2000), 26–36. DOI:https://doi.org/10.1145/352029.352035

[8] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. 2022. Low-code development and model-driven engineering: Two sides of the same coin? *Softw Syst Model* 21, 2 (April 2022), 437–446. DOI:https://doi.org/10.1007/s10270-021-00970-2

[9] Jean-Marie Favre, Jacky Estublier, and Mireille Blay-Fornarino. 2006. *L'ingénierie dirigée par les modèles : au-delà du MDA*.

[10] FED. The Forrester Wave™: Low-Code Development Platforms, Q2... *Forrester*. Retrieved July 17, 2022 from https://www.forrester.com/report/The-Forrester-Wave-LowCode-Development-Platforms-Q2-2016/RES117623?iOS=

[11] Holger Giese, Nelly Bencomo, Liliana Pasquale, Andres J. Ramirez, Paola Inverardi, Sebastian Wätzoldt, and Siobhán Clarke. 2014. Living with Uncertainty in the Age of Runtime Models. In *Models@run.time: Foundations, Applications, and Roadmaps*, Nelly Bencomo, Robert France, Betty H. C. Cheng and Uwe Aßmann (eds.). Springer International Publishing, Cham, 47–100. DOI:https://doi.org/10.1007/978-3-319-08915-7_3

[12] Jack Greenfield. 2004. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. In *Software Product Lines* (Lecture Notes in Computer Science), Springer, Berlin, Heidelberg, 304–304. DOI:https://doi.org/10.1007/978-3-540-28630-1_19

[13] James H. Hill. 2011. Measuring and Reducing Modeling Effort in Domain-Specific Modeling Languages with Examples. *2011 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems* (2011). DOI:https://doi.org/10.1109/ECBS.2011.22

[14] Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé. 2020. Challenges & opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. Association for Computing Machinery, New York, NY, USA, 1–10. Retrieved July 18, 2022 from https://doi.org/10.1145/3417990.3420204

[15] John Klein, Harry Levinson, and Jay Marchetti. Model-Driven Engineering: Automatic Code Generation and Beyond. 51.

[16] Timothy C. Lethbridge, Andrew Forward, Omar Badreddin, Dusan Brestovansky, Miguel Garzon, Hamoud Aljamaan, Sultan Eid, Ahmed Husseini Orabi, Mahmoud Husseini Orabi, Vahdat Abdelzad, Opeyemi Adesina, Aliaa Alghamdi, Abdulaziz Algablan, and Amid Zakariapour. 2021. Umple: Model-driven development for open source and education. *Science of Computer Programming* 208, (August 2021), 102665. DOI:https://doi.org/10.1016/j.scico.2021.102665

[17] T. Mens and T. Tourwe. 2004. A survey of software refactoring. *IEEE Transactions on Software Engineering* 30, 2 (February 2004), 126–139. DOI:https://doi.org/10.1109/TSE.2004.1265817

[18] Pierre-Alain Muller. 2006. De la modélisation objet des logiciels à la metamodélisation des langages informatiques. thesis. Université Rennes 1. Retrieved August 24, 2022 from https://tel.archives-ouvertes.fr/tel-00538525

[19] Jakob Pietron, Alexander Raschke, Michael Stegmaier, Matthias Tichy, and Enrico Rukzio. 2018. A study design template for identifying usability issues in graphical modeling tools. In *Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October, 14, 2018* (CEUR Workshop Proceedings), CEUR-WS.org, 336–345. Retrieved from http://ceur-ws.org/Vol-2245/mdetools_paper_4.pdf

[20] Parsa Pourali. 2020. A User-Centric Approach to Improve the Quality of UML-like Modelling Tools and Reduce the Efforts of Modelling. University of Waterloo. Retrieved from http://hdl.handle.net/10012/15595

[21] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 171–178. DOI:https://doi.org/10.1109/SEAA51224.2020.00036

[22] J. Sztipanovits and G. Karsai. 1997. Model-Integrated Computing. *Computer* (1997). DOI:https://doi.org/10.1109/2.585163

[23] Marcello Thiry and Roger A. Schmidt. 2017. Self-adaptive Systems Driven by Runtime Models. 248–253. DOI:https://doi.org/10.18293/SEKE2017-168

[24] Yentl Van Tendeloo, Simon Van Mierlo, and Hans Vangheluwe. 2019. A Multi-Paradigm Modelling approach to live modelling. *Softw Syst Model* 18, 5 (October 2019), 2821–2842. DOI:https://doi.org/10.1007/s10270-018-0700-7

[25] Umple: Merging Modeling with Programming. Retrieved August 24, 2022 from https://cruise.umple.org/umple/

[26] The Forrester Wave™: Low-Code Development Platforms For Professional Developers, Q2 2021. Retrieved July 17, 2022 from https://reprints2.forrester.com/#/assets/2/228/RES161668/report