Heinrich C. Mayr, Stefanie Rinderle-MA, Stefan Strecker (Hrsg.): 40 Years EMISA
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2020 59

Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project

Kai Adam[1], Judith Michael[1], Lukas Netz[1], Bernhard Rumpe[1], Simon Varga[1]

Abstract: The development of domain-specific information systems, especially web information systems, takes a certain amount of time, needs intensive testing to ensure a certain quality and lacks the consistency of front- and backend. Using model-based strategies for the creation of information systems helps to overcome these problems by fastening the development process, facilitating testing and ensuring consistency-by-construction. In practice, however, they are still rarely used. In this paper, we show that model-based engineering is beneficial for the creation of an enterprise information system and improves the quality of the resulting product. We present the basic functionalities of our Generator for Enterprise Management (MontiGEM) and discuss identified problems and lessons learned in a project in practice. The generator was developed simultaneously with and for an enterprise management system. Our research shows that the use of generative methods and MBSE improves the adaptability and reusability of parts of the application on the one hand but on the other hand, there are still obstacles that slow down its broad application in practice.

Keywords: Agile Development · Data-Intensive Enterprise Information Systems · Domain-Specific Modeling Languages · Generative Software Engineering · Model-Based Software Engineering · MontiGEM · Web Information System Engineering

1 Introduction

Context. To effectively create an enterprise information system (EIS), investigating and modeling the domain in focus is essential to create a usable and reliable system. Using models decreases the gap between software abstractions on the problem level and its implementation. In Model-based Software Engineering (MBSE) models play an important role. It supports strong stakeholder involvement and the *constructive generation* or *manual synthesis of code* from models is among the first steps of model-based development processes. Thus, one or more modeling languages are the central notation and replace the programming language as much as possible. For domain-specific needs, MBSE relies on the use of domain-specific modeling languages (DSML) [Vö13] as a central notation. DSMLs can be developed to be used for several purposes such as designing, programming and testing software systems or for describing the behavior of a system or processes.

Motivation and Relevance. MBSE increases efficiency and effectiveness in software development and its adoption in the software industry is foreseen to grow exponentially in

[1] RWTH Aachen University, Software Engineering, Germany {surname}@se-rwth.de

the near future [BCW17]. Using MBSE approaches for web applications improves their development as well [MCM14]. In order to impart knowledge about MBSE approaches, more tools and experience with modeling languages as well as more successful examples for practical application in EIS are needed.

Goals. In this paper, we discuss lessons learned and obstacles that slow down the broad application of generative MBSE approaches for EIS in practice in order to identify starting points for future research and improvements.

Approach and Main Results. To be able to discuss the lessons learned in detail, we introduce the practical application of MBSE in an agile EIS development project. As part of a MBSE project, we developed the Generator for Enterprise Management (MontiGEM) based on the language workbench and code generation framework MontiCore [HR17]. The realization was accompanied by strong stakeholder involvement processes: As usual in scrum, we define the use cases together with future users. Interested future users were a part of the the engineering process who provided feedback on implemented features. Thus, it was important to consider and react to changes quickly. In [Ad18] we already presented our approach for model-based generation of data-intensive EIS at a glance. This paper (1) presents an improved version of (MontiGEM) and basic functions to handle different DSMLs and (2) shows lessons learned and obstacles for using MBSE for creating EIS. With the help of a generator, some problems, such as data inconsistency on frontend (FE) and backend (BE), can be avoided. Nevertheless, we state additional improvements for the „ease of use“ for engineers.

Outline. The paper is structured as follows: Section 2 presents our approach for MBSE of EIS and the generator MontiGEM. In section 3, we discuss advantages and disadvantages of our approach as well as lessons learned and obstacles in the practical realization in a software engineering (SE) project. Section 4 discusses related work in comparison to our approach and the state-of-the-art of MBSE for web-EIS. The last section reviews the current progress and highlights further goals and next steps for our approach.

2 Model-based Generation of EIS with MontiGEM

With MontiGEM it is possible to generate large parts of data centric business applications: The datastructure and database communication, functions, access control, and the graphical user interfaces (GUIs). The generator MontiGEM uses different kinds of models as input and uses them as the base for code generation of the BE and FE in different target languages. The internal architecture of the generator includes three main processes: read, transformation and generation. A model loader loads all input models and handles their transformation in an internal accessible structure, the abstract syntax tree (AST), by using parsers and a library of data structures and functions. The central part of MontiGEM transforms several input ASTs to output ASTs. The template engine processes each output AST together with standardized as well as project-specific templates, which describe the concrete shape of the

resulting artifacts, and produces as an output the code files for BE and FE. The components model loader and template engine are generated with MontiCore[2].

CD4A (class diagram for analysis [Ob17]) is used, to describe the basic domain model (data structure). Purple boxes represent all the classes generated out of the one given *Classes.cd*, see fig. 1, 2 and 3. Each of these classes are used for a specific task, e.g., the database communication done with hibernate or network transportation with a REST infrastructure. This domain model can be written by domain experts.

Fig. 1: Model files with their resp. number of occurrences (*: any number, 1: exactly one)

Based on the domain class diagram, an *OCL* [Ma17; Ob14] model is used to restrict the attributes of defined classes to describe valid objects (yellow boxes). Resulting validators need different levels of detail, especially FE and BE have distinct demands. Error messages or even specific expressions can be used to adapt objects to the exact behavior required. Through the generation process both, the validators on the FE and BE, are consistent and thus check for the same validity.

Aggregate models (blue boxes) summarize attributes of specified domain classes and/or calculated values to limit the view to the data needed in the FE. For each dataclass multiple constellations of aggregates are viable to use. The models and thus the generated code have little information about the logic with which the aggregate object is generated, so a considerable part of it currently has to be written by hand.

Based on the previous described DSMLs, the GuiDSL is used to create views for the FE (GUIs) and fill them with the corresponding data defined by aggregates. For each page there is an affiliated model. This model separation improves the configurability and the separation of concerns. The model describes the usage of the aggregate and generates communication and validity checking. The model is the base for the logic component, which handles the data collection and user interaction, and a HTML view (red boxes).

Boxes with a *hwc* addition indicate, that handwritten parts are necessary to work properly. [Gr15] describes the used integration of handwritten and generated object-oriented code. A constant regeneration based on the given models is possible. We expect to be able to (nearly) fully generated these missing parts in the future by improving the DSMLs.

A command infrastructure and generated commands for each domain and aggregate class are used to handle the entire communication between BE and FE. This is based on the command processor pattern [BHS07]. The aggregates selected in the GuiDSL are requested and can be written back by commands on the FE. Through this mechanism data consistency is ensured.

[2]http://www.monticore.de/

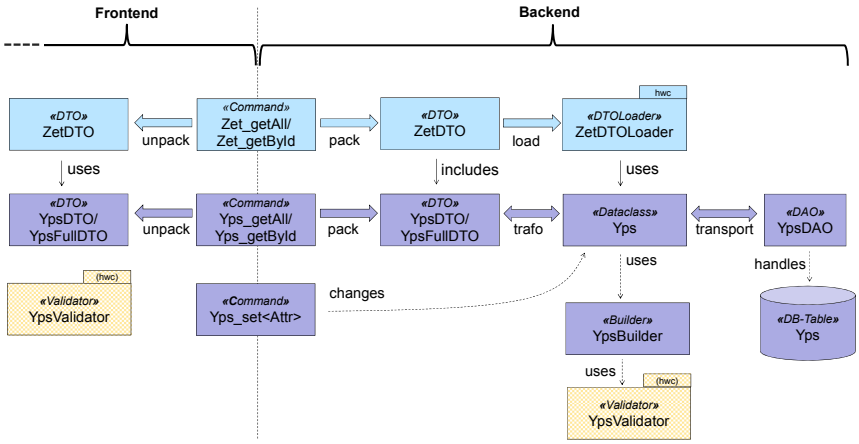

Fig. 2: Generation of BE files for the example class Yps based on the model files from Fig. 1

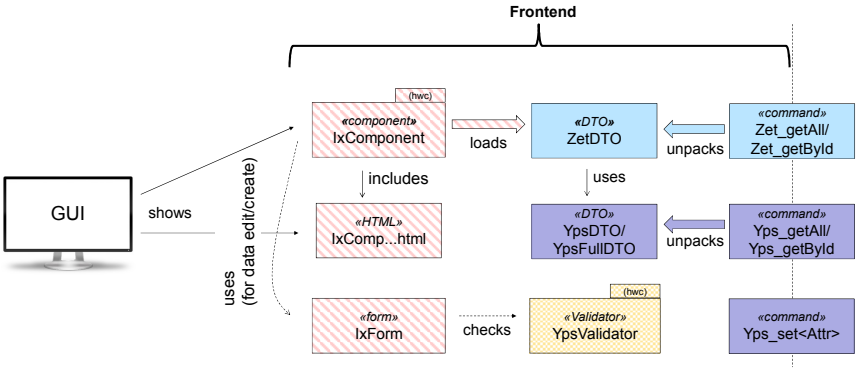

Fig. 3: Generation of FE files for the example class Yps based on the model files from Fig. 1

To sum up, MontiGEM provides us strong support in our practical development project, where we have to consider and react to changes in the requirements quickly. All input models are defined by UML/P [Ru11] inspired DSMLs.

3 Lessons learned in practice

To engineer EIS for research purposes strongly differs from software engineering in industrial-strength projects: in most cases it is enough for academia to have a demonstrator or prototype which is stable for one browser and one screen resolution, there is no need to catch all user errors as oneself and the research team are the only users and performance issues are only important in some cases, e.g. for data-intensive systems.

We developed and evolved MontiGEM in an industrial-strength project for RWTH Aachen university and clearly noticed these differences. The following *lessons* we have *learned during the MBSE process* lead to research gaps that should be addressed in the future.

The simultaneous development of the application and the generator itself is more time consuming than to do that separately. Clearly, to develop a generator without a specific use case is less effective, but to start a project with a basic generator for similar purposes significantly facilitates the development process. If the development of the generator is completed, the generation of similarly structured applications, e.g. Web-EIS which present data in various ways, in addition with minimal handwritten code is well manageable. Setting up a generator for the first time takes a while. Depending on the complexity of the project, the time taken to configure the generator can consume the time gained from code generation. Reusing a configured generator with prefabricated languages and generator components mitigates this problem.

There are strong dependencies between models of different DSMLs. For example the GUI model requires references to the aggregate model in order to display data. Typical IDEs are not able to check these calls, therefore the developer has to ensure validity. This is highly error prone and is only confirmed after a rebuild. Possible solutions are, e.g., composition on model level based on the symbol table to create a consistent set of models in different DSMLs.

There are strong dependencies of the project on the generator. Changes in the generator may lead to inconsistencies in the GUI. It is possible that validators and handwritten code does not completely fit into the generated files any more. One approach could be to implement generator composition in order to describe more logic in related models and to minimize the need for handwritten code.

Changes in the main data-classes model results in changes of the data-structure and requires migration. As the main data-classes model is generating the database, changes in the model have strong effects on it. If there already exists a running system with real data, this results in regular migrations of the database and change or addition of data, e.g., if required fields are added, existing fields are changed to be required or relations are changed. For a system with only one database this might lead to additional work with manageable time expenditure, but this time expenditure increases for large scale projects with several databases including different data. It even increases more, if several different versions of the EIS exists. In these cases it is important to develop strategies to (semi-)automate the migration process, to ensure that no data is in an inconsistent state or lost (at least they should be recoverable from a backup).

Model changes require manual test changes. Tests need to check the functionality in different browsers and a large variety of screen resolutions. Changes in data-structure or dummy data could lead to failed tests, e.g. check for a certain fixed result. With the goal to support software engineers as much as possible, the generation of the test infrastructure is

an important aspect for time reduction. MBSE should and could improve that by automatic generation of dummy data with all peripheral cases including tests for them.

It is a challenge to identify the generateable parts of a software project. A generator is well suited to apply patterns and rules to a set of models. Thus we can generate many lines of code from a few lines of a model. A more diverse code without common structure requires further adjusted models, which in the worst case match the size of the generated code. It is recommended to model the software project with these similarities in mind, and to adjust with handwritten code. Project with high portions of algorithmic diversity (e.g. SatSolvers) are unlikely to be efficiently generated.

To sum-up: The use of a generator only pays off if (a) a predefined and reusable generator exists (small adaption), (b) the generator is adaptable, (c) language and generator components are modular and (d) you use it for more than only one project (increase reuse).

4 Discussion and Related Work

Model-driven software engineering, model-driven architecture and MBSE are more and more used in practice. There exist several surveys and case studies for its application in industry [HRW11; Li18; MD08; To11] and the generation of specific parts such as for automated test generation [Ar18]. Nevertheless, the maturity of tool environments is critizised and is perceived as unsatisfactory for large-scale industrial adoption [MD08].

There exist several approaches and tools for Model-Driven Web Engineering [MRV08] but the paradigm shift from code-centric to generative approaches that has been expected in industry for years is still to come [MKH18]. There are attempts to explain this phenomenon, such as the impact of personality on the intention to adopt an MBSE method [MKH18]. The findings in this paper in section 3 refer to improvements for „ease of use“ as intention to adopt to MBSE approaches.

In comparison to the seven categories of web-based applications proposed by [GM01] our system generated with MontiGEM is mainly *informational*, *interactive*, and a *basic collaborative work environment*. It is an *informational* EIS, which provides data view-specific based on underlying domain data objects. It is *interactive* as the views and forms provide the possibility to interact with (parts of) the domain data. Our EIS is a *basic collaborative work environment* as several users can work on the data with a defined role and right structure.

Following [Of02] the most important quality criteria for web application success from the point of view of managers and practitioners are reliability, usability, security, availability, scalability, maintainability and time-to- market. MontiGEM generates a reliable system which ensures consistency-by-construction as much as possible, MBSE approaches improve the usability for the developer, the maintainability is improved as well and time-to-market is reduced for generative approaches.

5 Conclusion

Our lessons learned and obstacles show, that there are still several research gaps to work on in future in academia and practice, to provide good tools and facilitate the engineering process of developers in MBSE processes for EIS. Current trends on ICT technologies for EIS face four big challenges [El16] for the next generation of EIS: (1) Data Value Chain Management, (2) Context Awareness, (3) Interaction and Visualization and (4) Human Learning. Using MBSE approaches strongly support (2) and (3).

In future, we will show the practical application of MontiGEM for other domains, e.g. Cyber-Physical-Systems (CPS) with data from sensors, working stations or data from workers in the Internet of Production.

The usage of a generator for complex enterprise applications offers benefits, but also includes custom parts which are only usable in this specific case. A modular and configurable generator is a necessary condition to be able to adapt and reuse the generator to more sophisticated variants such as other coding languages or infrastructures. To further improve the generator and reduce handwritten parts an own aggregate model language with expressions to describe the logic to calculate/ get the values out of domain objects could be added. A *workflow-engine* could be used to completely describe the process within the application (BE/FE), the navigation and simplify the overview for programmers and domain experts.

References

[Ad18] Adam, K.; Netz, L.; Varga, S.; Michael, J.; Rumpe, B.; Heuser, P.; Letmathe, P.: Model-Based Generation of Enterprise Information Systems. In: EMISA'18. CEUR 2097, pp. 75–79, 2018.

[Ar18] Arcuri, A.: An experience report on applying software testing academic results in industry: we need usable automated test generation. Empirical Software Engineering 23/4, pp. 1959–1981, 2018.

[BCW17] Brambilla, M.; Cabot, J.; Wimmer, M.: Model-Driven Software Engineering in Practice. Synthesis Lectures on SE 3/1, pp. 1–207, 2017.

[BHS07] Buschmann, F.; Henney, K.; Schimdt, D.: Pattern-oriented Software Architecture: on patterns and pattern language. John wiley & sons, 2007.

[El16] El Kadiri, S.; Grabot, B.; Thoben, K.-D.; Hribernik, K.; Emmanouilidis, C.; von Cieminski, G.; Kiritsis, D.: Current trends on ICT technologies for enterprise information systems. Computers in Industry 79/, pp. 14–33, 2016.

[GM01] Ginige, A.; Murugesan, S.: Web engineering: An introduction. IEEE Multimedia 8/1, pp. 14–18, 2001.

[Gr15] Greifenberg, T.; Hölldobler, K.; Kolassa, C.; Look, M.; Mir Seyed Nazari, P.; Müller, K.; Navarro Perez, A.; Plotnikov, D.; Reiß, D.; Roth, A.; Rumpe, B.; Schindler, M.; Wortmann, A.: Integration of Handwritten and Generated Object-Oriented Code. In: Model-Driven Engineering and Software Development. Vol. 580. Comm. in Comp. and Inf. Science, Springer, pp. 112–132, 2015.

[HR17] Hölldobler, K.; Rumpe, B.: MontiCore 5 Language Workbench Edition 2017. Shaker Verlag, 2017.

[HRW11] Hutchinson, J.; Rouncefield, M.; Whittle, J.: Model-driven engineering practices in industry. In: ICSE 2011. ACM, New York, N.Y., p. 633, 2011.

[Li18] Liebel, G.; Marko, N.; Tichy, M.; Leitner, A.; Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Software & Systems Modeling 17/1, pp. 91–113, 2018.

[Ma17] Maoz, S.; Mehlan, F.; Ringert, J. O.; Rumpe, B.; von Wenckstern, M.: OCL Framework to Verify Extra-Functional Properties in Component and Connector Models. In: Proc. of MODELS 2017. Workshop EXE. CEUR 2019, 2017.

[MCM14] Martínez, Y.; Cachero, C.; Meliá, S.: Empirical study on the maintainability of Web applications: Model-driven Engineering vs Code-centric. Empirical Software Engineering 19/6, pp. 1887–1920, 2014.

[MD08] Mohagheghi, P.; Dehlen, V.: Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In: Model Driven Architecture: Foundations and Applications. Springer Berlin Heidelberg, pp. 432–443, 2008.

[MKH18] Mikkonen, T.; Klamma, R.; Hernández, J., eds.: Web Engineering. Springer International Publishing, 2018.

[MRV08] Moreno, N.; Romero, J. R.; Vallecillo, A.: An Overview Of Model-Driven Web Engineering and the Mda. In: Web Engineering: Modelling and Implementing Web Applications. Springer London, pp. 353–382, 2008.

[Ob14] Object Management Group: Object Constraint Language, 2014, URL: https://www.omg.org/spec/OCL/2.4/PDF.

[Ob17] Object Management Group: OMG Unified Modeling Language (OMG UML), 2017, URL: https://www.omg.org/spec/UML/2.5.1/PDF.

[Of02] Offutt, J.: Quality attributes of Web software applications. IEEE Software 19/2, pp. 25–32, 2002.

[Ru11] Rumpe, B.: Modellierung mit UML. Springer Berlin, 2011.

[To11] Torchiano, M.; Tomassetti, F.; Ricca, F.; Tiso, A.; Reggio, G.: Preliminary Findings from a Survey on the MD State of the Practice. In: Int. Symp. on Empirical Software Engineering and Measurement. Pp. 372–375, 2011.

[Vö13] Völter, M.; Benz, S.; Dietrich, C.; Engelmann, B.; Helander, M.; Kats, L. C. L.; Visser, E.; Wachsmuth, G.: DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. dslbook.org, 2013.