Joint Proceedings of Modellierung 2020 Short, Workshop and Tools & Demo Papers
22 Modellierung 2020: Short Papers

Generated Enterprise Information Systems: MDSE for Maintainable Co-Development of Frontend and Backend

Arkadii Gerasimov[1], Patricia Heuser[2], Holger Kettenißß[2], Peter Letmathe[2], Judith Michael[1], Lukas Netz[1], Bernhard Rumpe[1], Simon Varga[1]

Abstract: Universities, like any application domain and industry sector, have to establish a well functioning, reliable management accounting, and financial reporting software system. Currently, chairs have different technical solutions for their financial management such as commercial accounting software tailored to the needs of the central administration as well as the chairs' own data collections with additional information in other software tools. Previous work did not investigate the use of model-driven software engineering methods for the maintainable development of a full-size real-world enterprise information system. This paper shows the application of model-driven software engineering methods to create this system and support the maintainable co-development of frontend and backend written in different programming languages. We are using a variety of models and modeling languages in addition to an application generator that allows for continuous re-generation. Our approach can be easily adapted to other problem domains to create a functional prototype out of models with minimal manual effort.

Keywords: Controlling and Financial Management; Domain-Specific Modeling Languages; Enterprise Information Systems; Code Generation; Model-Driven Software Engineering; Prototyping

1 Motivation and Introduction

Motivation and relevance. Universities, like any application domain and industry sector, must drive the digital transformation of their processes. These processes include teaching, research, acquisition of third party funding and administration. Thus, appropriate and reliable software systems are an essential requirement. Chairs and institutes have to use commercial accounting software tailored to the needs of the central administration. Additionally, they have own data collections (including complex sheets and cross-references) using standard calculation tools such as Excel. Parts of this data has to be regularly synchronized with the university-wide information system by hand which is, clearly, error prone. Thus, a software solution is needed to improve the chairs' financial management accounting and planning. From a software engineering perspective, the application of Model-Driven Software Engineering (MDSE) methods on such a full-size real-world system is a great opportunity to investigate the maintainable development of such an Enterprise Information

[1] Software Engineering, RWTH Aachen University, Aachen, Germany {lastname}@se-rwth.de
[2] Controlling, RWTH Aachen University, Aachen, Germany {lastname}@controlling.rwth-aachen.de

System (EIS) which leads to the following research question: *How does model-based development support the maintainable co-development of frontend and backend written in different programming languages?*

Approach and main results. We show how MDSE methods can be used to create such a solution (an EIS) for planning, revision and governance of management processes and cost accounting. We state that MDSE methodologies suit best for the development of EIS for university chairs. Utilizing model-driven approaches allows for a strong interaction with end-users: Incremental releases of prototypes are discussed with them and their feedback is included in further improvement steps. From the technical perspective, generative software engineering enables continuous regeneration based on changing requirements and reduces the effort for producing handwritten code. Our approach uses a combination of models as input for an application generator which was simultaneously developed within this project and creates an EIS as output which can be extended by handwritten code.

Outline. The remainder of this paper is structured as follows. The next section describes the problem domain and shows the requirements for the resulting system. Section 3 presents how later users were involved, how model- based software engineering was applied and what domain specific languages and models were used. Section 4 discusses our approach and the results of the software solution. The last section concludes the paper.

2 The MaCoCo Project

To support small and medium-sized chairs at RWTH Aachen University, the interdisciplinary project MaCoCo (Management Cockpit for University Chair Management and Controlling) started in 2016 by two chairs: Controlling and Software Engineering. The main goal is to develop an enterprise information system for the planning, revision and governance of management processes and cost accounting [Ad18].

The software solution needs to be tailored for the needs of small and middle- sized chairs at the university. The phrase *small and middle-sized* is not related to a certain amount of funding, staff or accounts (there are no technical restrictions on this level) but to their organizational structure. In contrast, large chairs often have further administrative structures, are more workflow oriented, and have other needs regarding their financial affairs. Thus, they already use systems for accounting and sometimes even workflow systems similar to companies in the private sector. In addition to the functional requirements of such an EIS, e.g. to create/read/update/delete accounts, budgets, bookings, staff or contracts, the following *goals* were identified.

1. **Provide high adaptability.** The system and underlying regulations for universities management accounting underlie continuous changes. Thus, frequent changes should be applicable.

2. **Ensure consistency.** Changes, which should affect all chairs should be automatically available for them, e.g., changes of the salaries due to negotiations of the labour union.
3. **Data sovereignty.** Each chair should have the sovereignty over their data. This means that no access mechanisms for the central administration should be provided.

The competence, what concepts should be developed in the EIS is enclosed in the university staff. Thus, their expertise and competency needs to be continuously included into development and testing processes which supports the need for an agile process. Consequently, the software engineering process follows an agile paradigm, which strongly involves future users in the conceptualization process.

3 Approach

In order to fulfill the requirements, we follow a model-driven and generative approach together with strong user involvement. Following this user-centered approach, MaCoCo included a group of *lead users* giving feedback upon the end-user experience and helping in the development of concepts and system functions. Each member of the lead user group works in the target domain of MaCoCo on a daily basis and represents a larger group of end users. Additionally a *steering committee* supervised and evaluated the concepts and overall project progress.

3.1 Model-driven Software Engineering of Enterprise Information Systems

One way to incorporate user requirements is the direct involvement of domain experts in all the phases of a project. Abstract models can be created to ease the communication basis and reduce misunderstandings. Model-driven Software Engineering (MDSE) is the general description of software projects which use models as an essential part to describe the abstract specification *i.e.,* data structure or program flow for the software. Those models are suited for the domain's application context. Through this specialization domain experts can easily read them and thus are directly included in the development process.

Models as abstract structure. Models that provide an abstracted view of the real world, help to focus on aspects in isolated and structured manner [HM08; St73]. A model always represents only a limited scope of aspects of the real world. Thus a *set of multiple models* is used to provide a manageable representation. In MDSE such different kind of models can be used to describe the application structure. They are created at the analysis and design phase or even throughout the development of the application. Based on these models the software can be evaluated, planned and developed. Models are used as specification or guideline for the developer, for documentation purposes, static analysis, automated tests, rapid prototyping and for code generation.

Depending on the given domain, specialized **Domain Specific Languages** (DSLs) [Vö13] can be used to define models, that isolate individual aspects of the target domain. DSLs are used, to provide an abstract view in the context of the given domain and reduces the miscommunication with domain experts. They enable the domain experts to model a specific problem and provide a standardized specification within a project. For software engineering, DSLs are mostly based on UML, e.g. class or sequence diagrams.

3.2 Models in the development process

A software engineer typically implements an application based on models derived from the requirements. Based on the information content of a model, it can be used to derive repetitive parts of the implementation directly. In the following we present the set of elementary models used to generate the core of the enterprise information system which is the backbone of MaCoCo. The used generator framework *MontiGem* [Ad19] is based on the MDSE experiences of the SE group of RWTH Aachen University and the developed MontiCore language workbench and code generation framework [HR17; KRV10]. Models, created with UML/P [Ru16] inspired modelling languages, are used as input for this framework.

CD4A

```
1 package de.macoco;
2 classdiagram Example
      {
3   class Person {
4     String name;
5     int age;
6   }
7 }
```

List. (1) Example for CD4A representing the class `Person` with attributes `name` and `age`.

Aggregate-CD4A..

```
1 class PersonDetail {
2   String name;
3   List<long>
        changesInSalary
        ;
4 }
```

List. (2) Example aggregate based on the cd in List. 1.

GuiDSL

```
1 webpage SimplePage()
      {
2   row (c) {
3     button "Do
          something" {
4       click ->
            something
            ()
5     }
6   }
7 }
```

List. (3) Example gui model with a callback in a button.

Fig. 1: Example of Models used in MaCoCo

Class diagram for analysis (CD4A). CD4A is a textual DSL, which enables to model class diagrams intended for analyses[Ob17]. They are based on UML-CD features and have a Java-like syntax [Ru16]. In MaCoCo, CD4A models are used to describe the data structure (see List. 1). Line 1 describes, the package name and is similar handled to the package structure in Java. The keyword `classdiagram` signals the start of the class diagram and the name should be the filename via convention. A CD4A model can have multiple classes, interfaces, and many more definitions. One such example definition is the `Person` class in line 3 with two attributes namely `name` (4) and `age` (5). Visibility modifier such as `+/public` can be omitted and leave an underspecification of the model. The generator

then decides how to handle such cases. Context conditions check, *e.g.,* unique class- or attribute-names. Available types for the attributes are either predefined Java types, like `String, int, long, Date`, or imported types of other class diagrams.

Aggregates. Aggregates are based on the CD4A-Language (see List. 2). They are used for two things: (1) The data exchange between the application frontend and backend and (2) as view models for the frontend itself. Currently, it is necessary to write the data acquisition by hand. This could be further improved by adding expressions, which describe how each value in an aggregate is constructed. Aggregates are used to keep data sovereignty on the backend. This way, the frontend just provides filters or show/hide options but doesn't need any logic for the data itself. List. 2 shows a simple aggregate definition which has the exact same syntax as the CD4A DSL. Using a stronger correlation to classes of the domain class diagram, some parts, *e.g.,* types, could be omitted. Adding expressions, which describe how an attribute is build based on the domain class model, more logic could be extracted or even the complete aggregate could be generated.

GuiDSL. A Graphical User Interface (GUI) description language was developed and used to simplify and generify the GUI-creation for the frontend. List. 3 shows an example GuiDSL model. It is intended to describe the view of a (web)page. Included are interactions, such as click- or write-events. The language abstracts the complete visual design and data correlation of a page. Basic communication, *e.g.,* fetching of the data is already generated. Line 1 in List. 3 shows the views name. A `button` is used at line 3 with a `click`-callback. Additional handwritten code is needed for the full implementation of the callback.

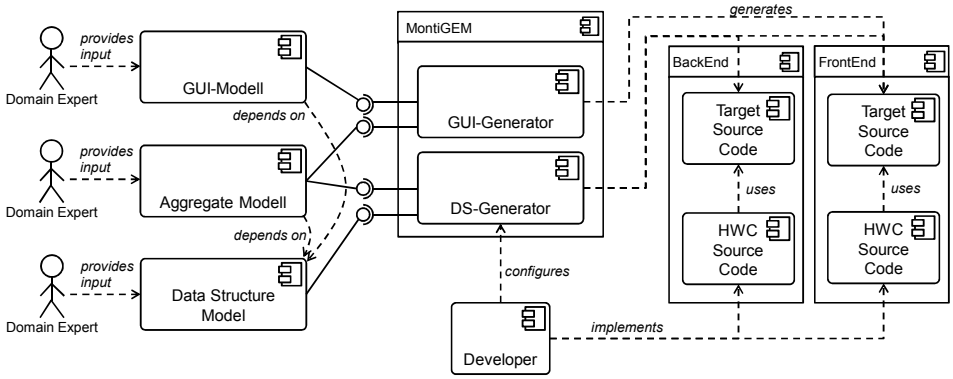

Fig. 2: Simplified overview of the main components of MontiGem

Generator environment. The generator processes provided models (in case of Figure 2 gui-, aggregate- and data structure models) and generates source code based on them. This code can be extended by handwritten code and additional business logic implementations. Compared to existing tooling which only support the development in certain aspects, the generator framework MontiGem has the goal to provide a functional prototype with minimal

manual effort. Additionally, a lot of boilerplate code is omitted and moved to the generated code. Changes in a model cascade naturally through the entire code base.

MontiGem processes the input models and produces an application that reflects each aspect defined by domain experts which can describe *e.g.,* the data structure or GUI features. Thus, models form the common basis for discussions on change requests. The generator-layer supports the iterative and incremental development of the application by allowing continuous re-generation using the provided models. The described models only describe the generated structure, but do not contain specific business logic. The generated code can be adapted by adding handwritten code [Ha15]. It is still kept separated from generated code, while integrating it in the product and enabling repetitive generation. The handwritten code extends the generated structure and provides business logic. This is achieved by logic in the generator called TOP-mechanism [HR17], which checks if there are handwritten parts present and handles overwritten parts.

3.3 Architecture

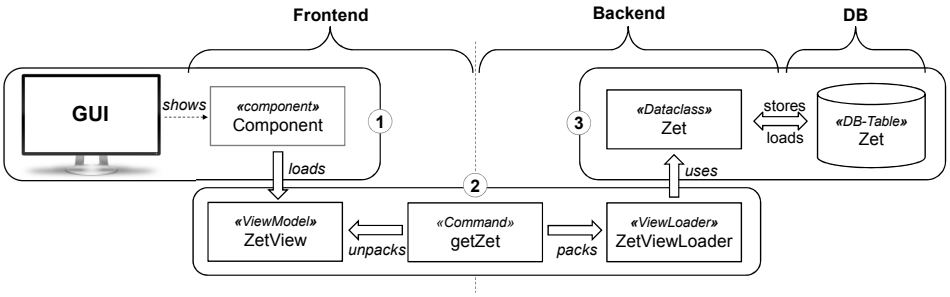

Fig. 3: Simplified overview of the main artefacts generated by MontiGem: (1) components from GuiDSL models, (2) view-models from Aggregate models, (3) data-classes from CD4A models.

Application Generator. Based on the provided models, MontiGem creates the complete application infrastructure, the application's backend, and frontend (Figure 2). It substitutes parts of the otherwise handwritten code. Figure 3 shows a simplified overview of the generated data structure and corresponding artifacts from the database up to the views. The different areas in Figure 3 correspond to languages described in subsection 3.2. The application is split into three parts. A *thin frontend client*, which provides a tailored view of the underlying data structure. The *application backend* contains the business logic and connects to the *database*. The system architecture enables to provide MaCoCo as a web application. A 3-tier client-server architecture in combination with the Model View View-Model (MVVM) pattern is applied to fit the requirements (section 2).

Persistence. MaCoCo's persistence tier contains and manages all the user data and provides general settings for all the instances. It is managed in a MySQL relational database

management systems (RDBMS) due to scalability purposes. The communication with the backend is handled with Hibernate and fully generated by MontiGem.

Scaleability and Reuseability. Each component is wrapped inside a docker container and kept stateless. This enables an easy version management for each of the software components. For instance, the component related container can be easily updated and restarted individually. Furthermore it is possible to have multiple backends and/or frontends running at the same time and thus easily perform load-balancing.

4 Discussion

Until now, more than 140 instances of MaCoCo are deployed for chairs of RWTH Aachen University. To answer the *research question*, model-driven development supports the maintainable [Zh13] co-development of frontend and backend written in different programming languages by providing a generator framework which uses the same models to create target code in several languages. Further detailed discussions and lessons learned about the simultaneous development of the application and the generator can be found in [Ad19].

Maintainable model-driven development. As the resulting code is directly derived from the provided models, the models themselves serve as an important part of the documentation and are always up to date. Multiple generated parts are originated in a single predefined structure (template) and can be easily changed collectively and thus reduce complexity. An additional concern is the type preserving communication between server and client. In our approach both, the data structure and communication endpoints on the frontend and backend originate from the same models and thus are consistent-by-design. The resulting (programing) languages can be switched by adapting templates and those templates can be reused in similar projects. This improvement limits the number of possible errors. Overall, the implementation based on a generative approach with usage of models written in different languages allows for the separate development of application frontend and backend. Furthermore, models represent independent components of the application and can be added to extend the data structure or the user interface without affecting the current implementation [JJ12; Ko08]. At the same time, dependencies across the entire code base are introduced as a result and need to be carefully handled [Ad19]. The need to manually define behavior results in dependencies between models and the hand-written code. MontiGem provides a solution using class inheritance which is not always applicable. In this case, the code is either completely generated or completely handwritten.

5 Conclusion

This paper shows the application of MDSE methods on a full-size real-world project: Until now, more than 140 instances of MaCoCo are delivered for chairs of RWTH Aachen

University, proving scalability and validity of the presented concept. This paper shows how to generate an enterprise information system which improves the maintainability of software projects. We have applied MDSE methods to create this system with the generator framework MontiGem which allows the maintainable co-development of frontend and backend written in different programming languages. Now it is possible to describe a web application with a small set of models. Only around 13% of the application's backend and 30% of the frontend is handwritten (including the runtime environment), the other parts are fully generated. With our approach, the end user benefits from fast handling of changes, easy adaptability to new guidelines and quick implementation of feature requests. Changes of the models are easily passed on to the source code and decrease development time. Our approach can be easily adapted to other problem domains using a different set of models to create a functional prototype with minimal manual effort.

References

[Ad18] Adam, K.; Netz, L.; Varga, S.; Michael, J.; Rumpe, B.; Heuser, P.; Letmathe, P.: Model-Based Generation of Enterprise Information Systems. In (Fellmann, M.; Sandkuhl, K., eds.): Enterprise Modeling and Information Systems Architectures (EMISA'18). Vol. 2097. CEUR Workshop Proceedings, CEUR-WS.org, pp. 75–79, 2018.

[Ad19] Adam, K.; Michael, J.; Netz, L.; Rumpe, B.; Varga, S.: Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In: Digital Ecosystems of the Future: Methods, Techniques and Applications (EMISA'19). LNI, (in press), 2019.

[Ha15] Haber, A.; Look, M.; Mir Seyed Nazari, P.; Navarro Perez, A.; Rumpe, B.; Völkel, S.; Wortmann, A.: Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components. In: Model-Driven Engineering and Software Development Conference (MODELSWARD'15). SciTePress, pp. 19–31, 2015.

[HM08] Hesse, W.; Mayr, H. C.: Modellierung in der Softwaretechnik: eine Bestandsaufnahme. Informatik-Spektrum 31/5, pp. 377–393, 2008.

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

[JJ12] Jia, X.; Jones, C.: AXIOM: A model-driven approach to cross-platform application development. ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends/, pp. 24–33, Jan. 2012.

[Ko08] Koch, N.; Knapp, A.; Zhang, G.; Baumeister, H.: UML-Based Web Engineering: An Approach based on Standards. In. Pp. 157–191, Jan. 2008.

[KRV10] Krahn, H.; Rumpe, B.; Völkel, S.: MontiCore: a Framework for Compositional Development of Domain Specific Languages. International Journal on Software Tools for Technology Transfer (STTT) 12/5, pp. 353–372, Sept. 2010, ISSN: 1433-2779.

[Ob17] Object Management Group: OMG Unified Modeling Language, V2.5.1, 2017.

[Ru16] Rumpe, B.: Modeling with UML: Language, Concepts, Methods. Springer International, 2016.

[St73] Stachowiak, H.: Allgemeine Modelltheorie. Springer-Verlag, Wien, New York, 1973.

[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.

[Zh13] Zhang, F.; Mockus, A.; Zou, Y.; Khomh, F.; Hassan, A. E.: How Does Context Affect the Distribution of Software Maintainability Metrics? In: IEEE Conf. on Software Maintenance. Pp. 350–359, 2013.