Mixed Generative and Handcoded Development of Adaptable Data-Centric Business Applications

Pedram Mir Seyed Nazari Alexander Roth Bernhard Rumpe

Software Engineering
RWTH Aachen University, Germany
{nazari,roth,rumpe}@se-rwth.de

Abstract

Consistent management of structured information is the goal of data-centric business applications. Model-driven development helps to automatically generate such applications. Current approaches target full or one shot generation of business applications and often neglect simplicity and adaptability of the code generator and the generated code. Inspection of the generated code is required to add functionality. Thus, here we discuss mechanisms for a code generator to generate a lightweight and highly customizable data-centric business application that is targeted for a variety of users including generated application users, tool developers, and product developers. We achieve simplicity by reducing the mapping of the input model to the generated code to a minimal core of easily understandable concepts. High customizability is achieved by providing a variety of mechanisms to extend the generator and the generated code. These include template overriding and hook points to extend the code generator; and hot spots and additional manual extensions to extend the generated code. It is even possible to fully control the code generator and the entire generation process via a scripting language.

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools and Techniques —Computer-aided software engineering (CASE)

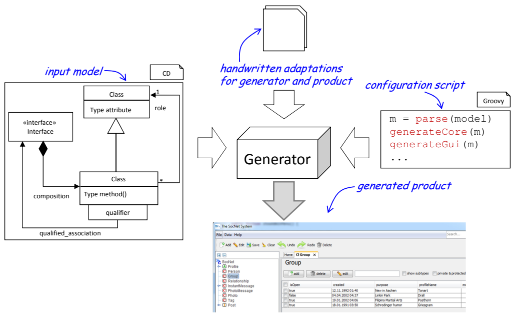
Keywords Data-Centric Business Application, Generative Development

DSM'15, October 27, 2015, Pittsburgh, PA, USA
© 2015 ACM. 978-1-4503-3903-2/15/10...$15.00
http://dx.doi.org/10.1145/2846696.2846707

1. Introduction

Data-centric business applications provide management functionality for structured and consistent information. They offer CRUD (create, read, update, and delete), search, and persistence functionality [11, 12]. Existing model-driven development approaches allow nearly full code generation [9]. Such generators can be powerful tools when used by experienced users. However, developers not familiar with such approaches hardly accept them, because of the complexity and the loss of control [7, 10]. Consequently, adapting and customizing the code generator or the generated output becomes a labor-intense and time-consuming task.

Even if nearly full code generation is achieved, simplicity (the amount of languages needed to describe the business application and the amount of approaches to integrate handcoded extensions), ease-of-use, and adaptability is not much addressed by current research [1, 2, 4, 13]. Instead an infrastructure for generating enterprise applications has been proposed [6, 8].



```
m = parse(model)
generateCore(m)
generateGui(m)
...
```

Figure 1. Overview of generation process.

We present a generator that aims at demonstrating the power of the generative software development methodology using the generator framework MontiCore [5]. Our main contribution is a demonstration of easy-to-use generation of almost ready-to-use business applications from abstract models as shown in Fig. 1. This approach is different to existing work as it only requires one input language to describe the data to be managed, provides clear customization approaches for the code generator and the generated systems, and presents a code generator that is designed to automatically integrate handwritten and generated code. The generated applications provide a graphical user interface to manage instances of the modeled system. Furthermore, they allow to persist instances in the cloud and share them among users, which may have different roles and rights.

2. Generation of Application from UML Class Diagrams

The input language for our code generator is a reduced variant of UML class diagrams provided in textual form. Certainly, it does not provide much application-specific functionality. Therefore, various extension and adaption mechanisms are introduced to extend the functionality of generated products. Nevertheless, the input language is sufficient to describe the managed data and generate a working application.

The generated application is a typical 3-layered architecture composed of the graphical user interface, the application core, and the persistence management to structure its products. The application core realizes only business functionality. The layers are independent and can easily be exchanged by different implementations. Each layer has its own runtime environment and standard components for accessing predefined not generated functionality. The generated applications allow for creation of users, roles, and definition of CRUD operations for each role. It is even possible to define very fine grained rights on attribute and association level.

A code generator becomes helpful, when it effectively assists developers to speed up their work. This is only possible, when the generator actually takes some burden from the developer. For example, by making certain decisions and generating corresponding functionality. Our generator for example targets desktop applications with a layered architecture. Based on that choice, it embodies a variety of automatically generated additional functionality.

It is an intrinsic property of a good generator to be able to adapt either the generation process or the generated code. In particular, for algorithms that usually cannot be described in a more abstract form than the implementation of the algorithm itself, manual implementation is necessary. We provide explicit hook points, which are dedicated spots in templates intended to be customized and extended. A more detailed level of customization is provided by allowing to replace every template of the code generator with a custom one. In order to give developers full control of the generation process, which includes parsing models, checking context conditions, and generating code, we a scripting language for generator control. Finally, for the generated applications, we offer hot spots as a dedicated spot in the generated code, which is usually known from frameworks as provided methods that have to be overridden, and concepts to extend the generated classes [3]. We strictly separate handcoded artifacts from generated artifacts to allow complete regeneration without loss of the customizations and adaptations.

References

[1] W. L. d. S. Carlos Eduardo Cirilo, Antonio Francisco do Prado and L. A. M. Zaina. *Interactive Multimedia*, chapter Building Adaptive Rich Interfaces for Interactive Ubiquitous Applications. 2012. ISBN 978-953-51-0224-3.

[2] A. Cicchetti, D. Di Ruscio, and A. Di Salle. Software customization in model driven development of web applications. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1025–1030, New York, NY, USA, 2007. ACM. ISBN 1-59593-480-4.

[3] T. Greifenberg, K. Hölldobler, C. Kolassa, M. Look, P. Mir Seyed Nazari, K. Müller, A. Navarro Perez, D. Plotnikov, D. Reiss, A. Roth, B. Rumpe, M. Schindler, and A. Wortmann. A Comparison of Mechanisms for Integrating Handwritten and Generated Code for Object-Oriented Programming Languages. In S. Hammoudi, L. F. Pires, P. Desfray, and J. F. Filipe, editors, *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development*, pages 74–85, Angers, Loire Valley, France, February 2015. INSTICC and ESEO, SciTePress.

[4] S. Kelly and J. Tolvanen. *Domain-Specific Modeling - Enabling Full Code Generation*. Wiley, 2008. ISBN 978-0-470-03666-2.

[5] H. Krahn, B. Rumpe, and S. Völkel. MontiCore: Modular Development of Textual Domain Specific Languages. In *Proceedings of Tools Europe*, 2008.

[6] V. Kulkarni and S. Reddy. Model-driven development of enterprise applications. In N. Jardim Nunes, B. Selic, A. Rodrigues da Silva, and A. Toval Alvarez, editors, *UML Modeling Languages and Applications*, volume 3297 of *LNCS*, pages 118–128. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-25081-4.

[7] V. Kulkarni and S. Reddy. A model-driven approach for developing business applications: Experience, lessons learnt and a way forward. In *Proceedings of the 1st India Software Engineering Conference*, ISEC '08, pages 21–28, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-917-3.

[8] V. Kulkarni and S. Reddy. A model-driven approach for developing business applications: Experience, lessons learnt and a way forward. In *Proceedings of the 1st India Software Engineering Conference*, ISEC '08, pages 21–28, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-917-3.

[9] V. Kulkarni, R. Venkatesh, and S. Reddy. Generating Enterprise Applications from Models. In J.-M. Bruel and Z. Bellahsene, editors, *Advances in Object-Oriented Information Systems*, volume 2426 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44088-8.

[10] V. Kulkarni, S. Reddy, and A. Rajbhoj. Scaling up model driven engineering experience and lessons learnt. In D. Petriu, N. Rouquette, and y. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6395 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16128-5.

[11] R. Mohan and V. Kulkarni. Model driven development of graphical user interfaces for enterprise business applications experience, lessons learnt and a way forward. In A. Schrr and B. Selic, editors, *Model Driven Engineering Languages and Systems*, volume 5795 of *LNCS*, pages 307–321. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04424-3.

[12] A. Schramm, A. Preuner, M. Heinrich, and L. Vogel. Rapid ui development for enterprise applications: Combining manual and model-driven techniques. In D. Petriu, N. Rouquette, and y. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 271–285. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16144-5.

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