GECO: Generator-Composition for Aspect-oriented DSLs

Reiner Jung

Kiel University
reiner.jung@email.uni-kiel.de

Abstract. Increasing size and complexity of software projects have triggered the use of domain-specific languages (DSL). Multiple DSLs, some with cross-cutting concerns, are used to describe software systems. In context of long-living software systems, requirements change over time causing an evolution of domains and subsequently the corresponding DSLs. Transformations are used to generate models and code from these DSLs combining information from different cross-cutting concerns. Due to the changes, the development and evolution of these generators become cumbersome and error-prone. The proposed GECO approach addresses this issue by introducing guidelines and tooling to ease generator composition and evolution. Furthermore, it allows parts of code generators to be developed and evolved separately reducing the overall complexity of code generation. In addition GECO fosters the reuse of DSLs and their generators in different projects.

1 Introduction

Model-driven engineering (MDE) addresses the complexity of software systems with a higher level of abstraction realized through models [1]. To compose models, domain specific languages (DSL) can be used, providing a concrete notation to abstract meta-models. Software systems comprise multiple concerns and views which are addressed by multi-view-modeling approaches [2] and aspect-oriented modeling (AOM) [3] with separate aspect and base models. Aspect and base models use the same meta-model (cf. [3]) or separate aspect and base meta-models with specific abstract syntaxes and semantics [4,5].

These source models of a software system are then transformed into target models or program code and supportive artifacts [6] by generators. Therefore, generators play a central role in MDE [7]. In an AOM context, generators may process multiple source models, representing different aspects, and integrate their information into target models (cf. [8,9]), making generators considerably complex artifacts, particularly because a generator may depend on multiple meta-models. Furthermore, depending on multiple meta-models, a transformation must be modified every time one of the meta-models is modified. Long-living software systems face changes due to alteration of requirements, technology and environment during their lifetime. Requirement changes may cause alterations in the domain of a specific DSL and require alterations to its syntax and semantic,

subsequently causing changes to the transformation. Technology and environment changes, induced by, e.g., service-oriented and cloud technologies [10], can affect how the semantic of the DSL is realized with the underlying technologies.

Frequent changes to complex transformations are cumbersome and can cause code and architecture degradation. Furthermore, DSLs can be altered quickly in comparison with transformations, and be reused in other software projects. However, such transformation, dependent on multiple source meta-models are harder to modify and reuse, as they process model elements from different meta-models in the same rule or operation. Therefore, separating transformation code for one specific meta-model and DSL is at least complicated or even unfeasible. This hinders software evolution and the reuse of tooling.

The goal of this research project is to provide an technology independent approach which supports the construction, evolution and reuse of complex generators, by dividing them into smaller and simpler fragments, targeting only one aspect or concern, and subsequently compose complete generators out of these fragments. Hence, the approach is named GECO for generator composition.

The remaining paper follows the proposed structure. Section 2 discusses the related work. Section 3 introduces the approach, and Section 4 the preliminary work to motivate, realize and evaluate the approach. The expected contributions are described in Section 5 and the evaluation in Section 6. The current status is summarized in Section 7 along with the timeline for the project.

2 Related Work

Only a few code generation approaches covering AOM have been published [7]. Most prominent are an FDAF [11] based approach [4], Theme/UML [12] based generator [13], and reusable aspect models (RAM) [14] which utilizes the *generic composition with Kermeta* weaver [15] for model weaving.

These approaches focus on UML and the generation of Java and AspectJ. They promise reusability of aspect models and code generators in different projects. While some approaches utilize stereotypes or profiles to describe aspects, they neither support UML profiles for their base and aspect models, nor do they address domain-specific languages. In these approaches, aspects are modeled with UML subsets. The weaving of aspects is controlled by direct references or model-subgraphs formulating pointcuts. However, these approaches do not address the construction of generators, as they see them as stable elements.

To the best of our knowledge, literature indicates that construction, evolution and reuse of generators have not had much attention in the modeling community [7]. While this is different in the compiler community, they do not address the specific nature of evolution and reuse, as the evolution steps of programming languages and their compilers happen in years not weeks. However, two modeling approaches address the construction and reuse of generators. The first approach focuses on product lines, where highly adaptable generators are required to support, e.g., different functionalities of the target domain [16]. It utilizes higher order transformations [17] to combine domain-specific transformation templates

for specific products. The second approach, Genesys [18], focuses on correctness of code generators which is ensured by composing generators from correct fragments, providing features, like loops or allocation. Both approaches allow the construction of single generators out of smaller fragments and can be used to construct fragments for GECO. However, they do not support AOM and are not prepared to support model traceability, which is necessary to resolve join points.

3 The GECO Approach

The general aim of GECO is to provide an approach and methods to construct code generators and support their evolution and reuse. The key challenges for this research project are the notation of pointcuts in DSLs, their resolvement to target model join points, and the decomposition of generators along concerns reflected in meta-models and facets of meta-models, like typing and expressions.

3.1 Basic Generator Scenarios

Code generation for software systems utilizing AOM involves different base and aspect models which are transformed into target models. In projects with multiple DSLs, like MENGES [8], the generation involves multiple generators processing and combining information from different models. However, based on the AOM paradigm, models play the role of an aspect or base model resulting in aspect to base model relationships [5].

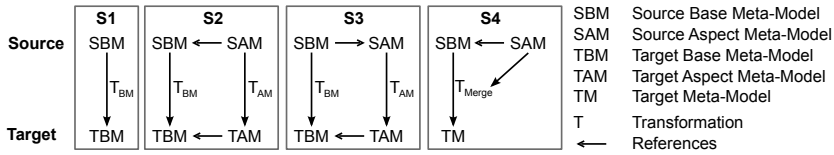

Fig. 1. Four different mega-model pattern for base and aspect meta-model with their respective transformations and target meta-models.

On that basis, the complex relationship graph of meta-models and generators can be split into four different mega-model [19] pattern (see Fig. 1). S1 is a simple transformation with one source and one target model. S2 describes that source model references are mapped to target model references, resembling the aforementioned Java/AspectJ generator approaches in Section 2. S3 reflects the situation, where the direction of the references is inverted from source to target models. This may happen to express function calls to an aspect model. Finally, S4 represents model or code weaving where a compatible aspect model is integrated into the base model. GECO utilizes different weavers, like GeKo [15] for models and AspectJ[1] for Java code.

[1] `https://www.eclipse.org/aspectj/`

3.2 Generator Composition

Depending on the described generator mega-model pattern, different information must be exchanged between generator fragments. In Fig. 2, the combination of fragments is illustrated along the two central pattern. In S2, the generator fragment $\mathsf{TF}_{\mathsf{BM}}$ produces as main output a model conforming to TBM. Similarly, $\mathsf{TF}_{\mathsf{AM}}$ produces an output conforming to TAM. To do so, $\mathsf{TF}_{\mathsf{AM}}$ must resolve the reference destinations in TBM based on the references expressed in SAM. This task requires trace information for the generated model nodes which are stored in a trace model conforming to a TRM. A trace model can be generated by $\mathsf{TF}_{\mathsf{BM}}$ or can be computed by a separate transformation $\mathsf{TR}_{\mathsf{BM}}$. Depending on the transformation language, the generation of the trace model must be explicitly specified or automatically added to a generator (see [20]).

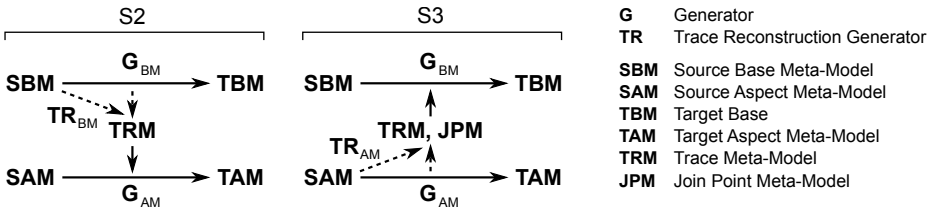

Fig. 2. Illustration of generator fragment compositions

S3 requires a different approach to fulfill its two tasks. First, it has to invert the direction of the references. And second, the references must be mapped from source to target level. Therefore, $\mathsf{TF}_{\mathsf{AM}}$ or a supplement $\mathsf{TR}_{\mathsf{AM}}$ produces a trace model for the aspect, and additionally relationships for source model join points stored in a join point model. The latter is required to infer the inverse references, to be placed in the target base model. And the trace model is used to determine target aspect model nodes corresponding to their source aspect model nodes.

3.3 Computing Target Model Join Points

In aspect-oriented DSLs and their meta-models, join points can be expressed as direct references pointing from an aspect model node to a base model node [5] or they can be determined with pointcuts specified as subgraph patterns [15] and model queries [21]. These join points refer to source base model nodes. However, to retain the join points on the target model level, they must be translated accordingly. Based on the trace model, all source model nodes in the join point collection are replaced by their respective target model nodes. As a generator might create multiple target model nodes for one source model node, the resulting target join point model might be significantly larger than the source join point model. For example, a source model node describing a filter is represented on the target model level by multiple public operations.

Based on current approaches to determine and represent traces [22,23], trace models also contain traces to nodes which are semantically not suited for weaving

behavior or structure. This is especially true when trace construction is automatically introduced to transformations [20]. For example, a join point representing an injection of a monitoring probe should reference an operation and not the parameter or return type of the operation. Therefore, only suitable nodes in the target model must be selected. In GECO this is realized by a subgraph pattern match. Based on this selection a set of remaining join points can be determined.

3.4 Obtaining Model Traces

In current research, different methods and approaches have been described to obtain and store model traces [24]. They can be categorized as constructive and recovery approaches. The latter use either deterministic algorithms or heuristics [25] to find matches. Heuristic approaches produce non-predictable results which is not sufficient for generators. Deterministic based recovery strategies require attributes which are considered identical in source and target models. However, this assumption cannot be guaranteed for all transformations, resulting in incomplete join point models. Therefore, only constructive approaches can determine suitable trace models, where either traces are generated by the fragment itself or by a supplement trace model generator. The first approach leads to more complex generators from a developer's perspective if the code must be added by hand. However, for some transformation languages, this feature can be added automatically [20]. The second approach circumvents this complexity issue by defining a separate generator, which also allows to integrate legacy generators were code alterations are not possible.

4 Preliminary Work

MENGES [8] aimed to develop a DSL for railway control centers, which resulted in eight different DSLs describing communication, logic, deployment, configuration and monitoring. The DSLs were developed in an agile setting where all features and changes have been recorded and documented. Thus, it is a good basis for a comparative evaluation for generator construction and evolution.

Based on experience in application monitoring, we developed an instrumentation aspect language [21] with a query based pointcut model capable of being used with any EMF meta-model. The tooling provides already a generator for AspectJ configuration files and partial code generation for instrumentation code.

A key problem of meta-models is the understanding of the syntactical and semantical aspects of their different features which must be established prior to writing code generators. We first addressed structural properties of DSLs in form of type-systems and instances of types [26]. Particularly important is the mapping of the source to the target type-system, which may involve special runtime code to support source level type concepts on the target level. Further, we explored the semantic properties of meta-models and typical features of specific meta-models types [5] induced by their AOM role and application context, like meta-model pattern for pointcuts, typing, expressions, data, state, and traces.

5 Expected Contributions

The main goal of GECO is to provide both: a method and a process to guide DSL and generator development, based on the above described approach. This includes meta-model patterns induced by the domain the meta-model is used for, application context, and role in AOM, as well as, the partitioning of generators along these patterns [5]. The EMF based tooling will support the composition of generators accompanied by a framework for DSL development supporting type-system integration. Furthermore, a reusable aspect DSLs for monitoring and application measuring will be created in cooperation with the Kieker project[2].

6 Evaluation Scenarios

The evaluation is built on two generator construction scenarios based on the Common Component Modeling Example (CoCoME) [27] and the aforementioned DSLs for railway control centers (RCC). The CoCoME scenario resembles a Java-based enterprise software system, while RCC project targets embedded systems. In both scenarios, DSL users will be involved to steer DSL feature changes by contributing feature requests.

The evaluation process is divided into three phases, where each one has its specific objectives. Phase one: Two existing code generators for the application (partial CoCoME) and a monitoring aspect [21] are integrated to evaluate the feasibility of all techniques, notations, and methods. The resulting generator and pointcut notations are thought to be used by other researchers of the priority program[3] in their case studies. The success of the first phase will be determined by its ability to deliver a working generator and the user feedback, based on interviews, regarding practicability of the different pointcut notations.

Phase two: Existing and newly constructed DSLs supplement the initial CoCoME models in order to specify all aspects of the CoCoME application. For these DSLs, generator fragments are developed by me and other researchers who utilize CoCoME in their software evolution scenarios. Based on the researchers scenarios, modifications to the DSLs and generators are performed simulating evolution. GECO will successfully complete this phase if it is able to provide stable generators, and the development time does not increase with every change. Furthermore, user feedback will determine the applicability of GECO.

The final phase will utilize the RCC DSLs in a setting performed together with an industry partner. Success will be determined similarly to phase two. However, in this scenario results from a previous generator development project for the same DSL will be used to evaluate an eventual cost benefit of GECO. Furthermore, the results and the approach will be reviewed by means of expert interviews of industry partners to evaluate its applicability in their specific domains and software development processes.

[2] `http://www.kieker-monitoring.net`
[3] `http://www.dfg-spp1593.de`

7 Current Status

Currently, the evaluation scenarios and partners have been selected. Still, details have to be discussed with industry partners. The monitoring DSLs [21] are implemented including the tooling. However, the generators for the instrumentation probes are only partial realized. Further, different weavers are currently investigated and the first evaluation phase setup is being prepared.

The final milestone for my doctorate is mid 2015. The next step is to final the setup for the first evaluation scenario which requires all technical parts of the approach in a working order. This includes the necessary generator fragments for the instrumentation probes. The weaving will be realized with AspectJ. The results for this first phase should be available in September 2014. Subsequently, the additional DSLs for the CoCoME scenario will be finalized and the generator fragments shall be available in December, providing the basis for the second phase of the evaluation. From January to April 2015, the third phase based on the RCC scenario will be conducted. The remaining time is reserved to finalize the thesis, tooling and documentation.

References

1. Stahl, T., Völter, M.: Model-Driven Software Development – Technology, Engineering, Management. Wiley & Sons (2006)
2. Kienzle, J., Abed, W.A., Klein, J.: Aspect-oriented multi-view modeling. In Sullivan, K.J., Moreira, A., Schwanninger, C., Gray, J., eds.: AOSD, ACM (2009) 87–98
3. Kramer, M.E., Kienzle, J.: Mapping aspect-oriented models to aspect-oriented code. In: Proceedings of the 2010 international conference on Models in software engineering. MODELS'10, Berlin, Heidelberg, Springer-Verlag (2011) 125–139
4. Bennett, J., Cooper, K., Dai, L.: Aspect-oriented model-driven skeleton code generation: A graph-based transformation approach. Science of Computer Programming **75**(8) (2010) 689 – 725 Designing high quality system/software architectures.
5. Jung, R., Heinrich, R., Schmieders, E., Strittmatter, M., Hasselbring, W.: A method for aspect-oriented meta-model evolution. In: Proceedings of the 2Nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling. VAO '14, New York, NY, USA, ACM (2014) 19:19–19:22
6. Biehl, M.: Literature Study on Model Transformations. Technical Report ISRN/KTH/MMK/R-10/07-SE, Royal Institute of Technology (July 2010)
7. Mehmood, A., Jawawi, D.N.: Aspect-oriented model-driven code generation: A systematic mapping study. Information and Software Technology **55**(2) (2013) 395 – 411 Special Section: Component-Based Software Engineering (CBSE), 2011.
8. Goerigk, W., Hasselbring, W., Hennings, G., Jung, R., Schneider, C., Schultz, E., Stahl, T., von Hanxleden, R., Weik, S., Zeug, S.: Entwurf einer domänenspezifischen sprache für elektronische stellwerke. In Jähnichen, S., Küpper, A., Albayrak, S., eds.: Software Engineering. Volume 198 of LNI., GI (2012) 119–130
9. Giacinto, D., Lehrig, S.: Towards integrating java ee into protocom. In: KPDAYS. (2013) 69–78

10. Hasselbring, W., Heinrich, R., Jung, R., Metzger, A., Pohl, K., Reussner, R., Schmieders, E.: iobserve: Integrated observation and modeling techniques to support adaptation and evolution of software systems. Research report, Kiel University, Kiel, Germany (October 2013)
11. Dai, L.: Formal Design Analysis Framework: An Aspect-oriented Architectural Framework. PhD thesis, The University of Texas at Dallas (2005)
12. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design. Addison-Wesley Professional (2005)
13. Hecht, M.V., Piveta, E.K., Pimenta, M.S., Price, R.T.: Aspect-oriented code generation. In: Simpsio Brasileiro de Engenharia de Software. (2005)
14. Klein, J., Kienzle, J.: Reusable aspect models. In: 11th Workshop on Aspect-Oriented Modeling, AOM at Models'07,. (2007)
15. Morin, B., Klein, J., Barais, O., Jézéquel, J.M.: A generic weaver for supporting product lines. In: Proceedings of the 13th International Workshop on Early Aspects. EA '08, New York, NY, USA, ACM (2008) 11–18
16. Kapova, L., Goldschmidt, T., Happe, J., Reussner, R.H.: Domain-specific templates for refinement transformations. In: MDI '10: First International Workshop on Model-Drive Interoperability, New York, NY, USA, ACM (2010) 69–78
17. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., Bzivin, J.: On the use of higher-order model transformations. In Paige, R., Hartman, A., Rensink, A., eds.: Model Driven Architecture - Foundations and Applications. Volume 5562 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 18–33
18. Jörges, S.: Construction and Evolution of Code Generators - A Model-Driven and Service-Oriented Approach. Volume 7747 of LNCS. Springer (2013)
19. Favre, J.M.: Foundations of model (driven) (reverse) engineering – episode i: Story of the fidus papyrus and the solarus. In: Post-Proceedings of Dagstuhl seminar on model driven reverse engineering. (2004)
20. Jouault, F.: Loosely coupled traceability for atl. In: In Proceedings of the European Conference on Model Driven Architecture workshop on traceability. (2005) 29–37
21. Jung, R., Heinrich, R., Schmieders, E.: Model-driven instrumentation with kieker and palladio to forecast dynamic applications. In Becker, S., Hasselbring, W., van Hoorn, A., Reussner, R., eds.: KPDAYS. Volume 1083 of CEUR Workshop Proceedings., CEUR-WS.org (2013) 99–108
22. Galvao, I., Goknil, A.: Survey of traceability approaches in model-driven engineering. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International. (2007) 313–313
23. Grammel, B., Kastenholz, S.: A generic traceability framework for facet-based traceability data extraction in model-driven software development. In: Proceedings of the 6th ECMFA Traceability Workshop, New York, NY, USA, ACM (2010) 7–14
24. Vanhooff, B., Van Baelen, S., Joosen, W., Berbers, Y.: Traceability as input for model transformations. In Oldevik, J., Olsen, G.K., Neple, T., eds.: Third ECMDA Traceability Workshop 2007 Proceedings,, SINTEF (June 2007) 37–46
25. Saada, H., Huchard, M., Nebut, C., Sahraoui, H.: Recovering model transformation traces using multi-objective optimization. In: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. (Nov 2013) 688–693
26. Jung, R., Schneider, C., Hasselbring, W.: Type systems for domain-specific languages. In Wagner, S., Lichter, H., eds.: Software Engineering (Workshops). Volume 215 of LNI., GI (2013) 139–154
27. Rausch, A., Reussner, R., Mirandola, R., Plasil, F., eds.: The Common Component Modelling Example (CoCoME). Volume 5153 of Lecture Notes in Computer Science. Springer Verlag Berlin Heidelberg (2011)