Domain-Specific Modeling Languages for Embedded System Development

Juha-Pekka Tolvanen, Steven Kelly

MetaCase
Ylistönmäentie 31, FI-40500 Jyväskylä, Finland
{jpt, stevek}@metacase.com

Abstract. Metamodels and generators are typically applied to define development environments that produce code and other software related artifacts. Based on our experience they can also be used to produce other non-software related artifacts, enable software and hardware co-design, support early level design space exploration, and automate testing activities. In this paper we describe – based on implementations done by users of MetaEdit+ tool – various kinds of embedded system development situations that benefit from metamodeling and generators.

1 Introduction

Metamodeling and code generators are typically applied to improve the productivity and quality of software development [13]. A metamodel defines one or more modeling languages for a specific area of interest raising the level of abstraction from the solution domain, code, world to the problem domain. Code generators provide then automation by reading the models created with the language to produce various kinds of artifacts like code, configuration, test data and documentation. Not all metamodels, though, lend themselves similarly to code generation as languages that do not focus on a specific domain being general purpose, such as UML and SysML, can not raise the level of abstraction and can't guarantee that the models created are complete and correct to enable code generation. Instead, the closer the language, and its metamodel, maps to the domain of interest, the more likely the models describe the system correctly and generators can then be applied efficiently. Often the most powerful modeling languages for code generation are those applied within a single company only.

The creation of domain-specific languages for modeling has become more widely used because modern metamodeling tools, for a review see [3], have removed the burden of tool creation and maintenance from the language creator: they provide fully functional modeling environments for a given language without any particular tool development activities. Industry-strength tools also enable experimentation with the language as it is built and can update models based on metamodel change: Work is not lost when moving to newer language versions.

Since metamodels allow defining domain-specific languages for various kind of situations we have seen the approach applied in many domains, including consumer electronics [5, 12], industrial automation [11], railway systems [8], mobile devices [1], telecommunication [1], and home automation [1, 9]. Not all these cases deal with code generation only. We start by describing the benefits that metamodels and code generators have provided – as reported by the users of MetaEdit+ [6] tool. Then we move beyond code generation and present other typical areas that benefit from metamodels and generators too.

2 Industry experiences

Raise in the level of abstraction provides many benefits and among the industry experience reports improved productivity is perhaps the most often claimed. While many companies report or provide anecdotes about productivity improvements, luckily some companies have also reported publicly their data. Kieburtz et al. [3] compared the use of domain-specific languages and code generators in experiments covering more than 130 development tasks in the domain of message translation and validation. Comparison to manual coding practices showed 300% increase in productivity. In the study the differences in the average performance of the subjects are statistically significant at confidence levels exceeding 99%. Usually most companies do not have resources and time to conduct as comprehensive studies and need to rely on more practical approaches. Panasonic [12] compared languages and generators they build for developing touch screen applications against manual programming by building most of the functions of the same touch screen device twice. The comparison to manual programming showed that modeling and code generation was over 400% faster. Polar [5] implemented languages for their sports computer applications along with code generators. In a laboratory study 6 engineers build individually a typical sports computer feature. After recording the development time they measured in average 900% increase in productivity. After extending the comparison by building a large portion of the sports computer functionality, similarly to [12], productivity increase was measured to be over 1000%.

Metamodels do not only capture domain concepts, but they may also cover domain rules. This allows languages to detect errors in the models and even prevent errors to occur in the first place. Since the impact of development approaches to the quality is harder and more laborious to measure than productivity, companies often rely on their engineers' opinions: at EADS it was seen that the quality of the generated code is clearly better because the modeling language rules out errors, eliminating them already in the design stage [1]. At Polar, programmers compared the generated code to the available manually written code and found the generated code having clearly better quality [5]. Kieburtz et al. [3] studied the quality of the code by analyzing errors found in generated code and in manually written code. Their statistically significant data showed that generated code had 50% less errors.

While other desirable characteristics have also been reported, like easier understanding of models [1, 8] and easier introduction of new developers [5], we focus in the following to the use of generators for producing other than plain code.

3 Beyond plain code generation

The use of metamodels is obviously not limited to automating software development tasks, but can also be applied for automating and supporting various system development tasks. We describe below some of these as been implemented in MetaEdit+ metamodeling and generator environment.

Generation of non-software artifacts. One obvious approach has been to extend the domain-specific languages to cover also parts of the non-software system. For example, in cases of developing industrial automation systems (e.g. fishing farm [11]) and home automation systems (heating system [14], Figure 1), the generators also produce other than software related artifacts, like hardware mappings, network device configuration, wiring plans, material usage, installation guidelines and even labels for a wiring cabinet. Generation of these artifacts from the same and single source brings major benefits, including already mentioned productivity and quality improvements.

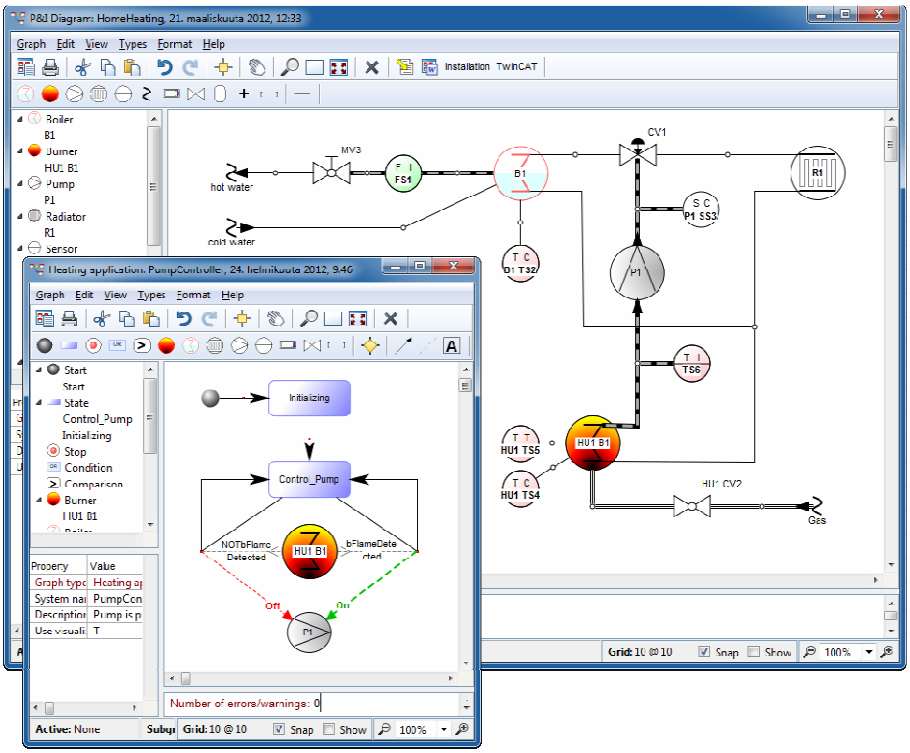

Fig. 1. Model of the system structure (right), model of the application behavior (left) [14]

The metamodels are in these cases extended to cover the system characteristics of interest. As an example, in Figure 1 the characteristics of the pipes and instruments of the heating system (right) are specified in addition to the behavioral models (left) used for code generation. These two models are not made with separate languages but the

languages share the same concepts and allow sharing the same model elements sicne they are based on common metamodel. Also domain rules are checked with the combined models.

Generators may also use the combined models to analyze the system in more detail. A particularly important part in embedded system development is generation of simulations as it enables addressing situations that would not be practically even possible to reach with the real system – being for example too dangerous or expensive to realize. Also analysis of safety characteristic and techniques like failure analysis can be addressed easier as generators decrease the cost for analysis and simulation.

Hardware and software co-design. Concurrent development of both hardware and software has always been of great importance. Metamodels may cover these two worlds to support co-design. Such approaches enable SW/HW allocation work and facilitate examination of different integration approaches. Figure 2 below shows an example of two different kind of architecture models from automotive: one addressing functional architecture (left) and another addressing hardware architecture (right). To support allocation among these, a dedicated language (shown as an allocation matrix in model level below) has been defined [7]. Such metamodels can cover the rules of allocation and generators can be applied to analyze different allocation options or produce the allocation information into existing analysis tools.

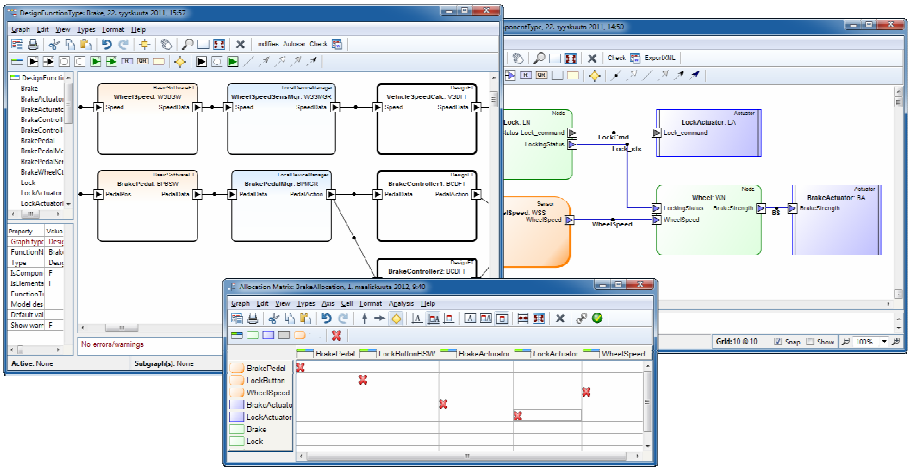

Fig. 2. Functional architecture (left), hardware architecture (right), and allocation modeling (bottom)

Design space exploration. Another area of using metamodels and generators is then defining completely new languages for the specific system development tasks. Some of the cases we have seen companies addressing deal with defining modeling languages for early-phase design space exploration, verification and validation as well as performing failure analysis. The metamodels are here defined to capture the needed characteristics of the system and generators produce the output in the format needed by a particular tool (e.g. ABSOLUT, TLA, SPIN, UPPAAL) used to perform the simulation and/or analysis.

The results of the analysis can be provided as a feedback to the models by updating the models or by annotating the models. In Figure 3, the results of failure analysis are shown by annotating the functions in the chain: failure in one port is traced and model shows how it is influencing in the whole system. In this example the analysis is performed by the generator of the modeling tool, but the models could also be used as a front-end for an existing analysis and simulation tool (as e.g. in [15]).

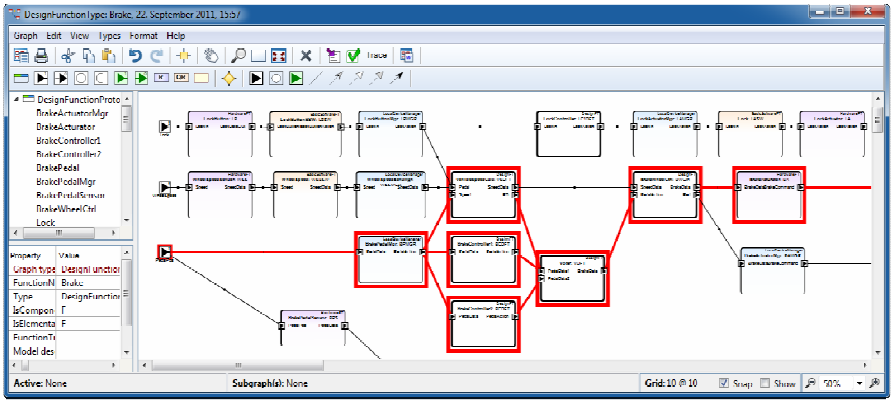

Fig. 3. Annotating failure trace directly in the model.

Automating testing. Test cases are typically written using a common scripting language to describe different aspects of the overall behavior of the system under test. These test cases share many common properties while exhibiting some degree of variation. Such items are perfect candidates for domain-specific languages which describe a set of variants with a common set of properties.

Here metamodeling is used to specify what are tested and the created models are used for test case generation. Different testing approaches exists as models created for specification can be used as a basis for test design and test case generation, models can be created to specify individual test cases, or models can specify test logic from which multiple test cases are then been generated. Industrial experiences of model-based testing (e.g. [10]) show remarkable improvements in productivity and coverage of test cases.

4 Concluding remarks

Experienced developers have always been looking how to automate repeating development tasks. Modern metamodeling tools make this task cost effective and provide industrial strength modeling tools for the engineers developing embedded systems. Typically the emphasis has been on generating the code, either for production systems or for prototypes. Metamodeling and related tools can also be applied to support other kind of development work as been presented in this paper: support design space exploration, hardware/platform and application co-design and automate testing activities. While the above mentioned approaches may require new

languages – new metamodels to define the languages - often the most straightforward approach is to extend the generators to produce various non-software related artifacts from the existing models, or from models which expression power has been extended via metamodels. Such non-software artifacts include simulation data, hardware mappings, device configuration, wiring plans, material usage calculations, installation guidelines, etc. Having a single source, models, to produce multiple different kinds of outputs brings many benefits as the design data is entered, checked and reviewed only once. The metamodels can also be defined so that the models are kept consistent and in-completeness information can be reported directly in the models.

References

1. Lan Cao; Ramesh, B.; Rossi, M., Are Domain-Specific Models Easier to Maintain Than UML Models, IEEE Software, July/Aug (2009)
2. Kelly, S., Tolvanen, J.-P., Domain-Specific Modeling: Enabling Full Code Generation, Wiley (2008) http://dsmbook.com
3. Kieburtz, R. et al., A Software Engineering Experiment in Software Component Generation, Procsof 18th International Conference on Software Engineering, IEEE Computer Society Press, March (1996)
4. Kouhen El, A., Dumoulin, C., Gerard, S., Boulet, P., Evaluation of Modeling Tools Adaptation, (2012) http://hal.archives-ouvertes.fr/hal-00706701
5. Kärnä, J., et al. Evaluating the Use of Domain-Specific Modeling in Practice, Proceedings of the 9th Workshop on Domain-Specific Modeling, HSE Print, B-108 (2009)
6. MetaCase, MetaEdit+ 5.0 (2012), http://www.metacase.com
7. MetaCase, EAST-ADL Example, (automotive architecture description language) (2012)
8. Mewes, K., Domain-specific Modelling of Railway Control Systems with Integrated Verication and Validation, PhD thesis, University of Bremen (2009)
9. Puolitaival, O.-P., Home Automation DSL case, Code Generation Conference (2011)
10. Puolitaival, O.-P., Kanstren, T., Rytky, V.-P., Saarela, A., Utilizing Domain-Specific Modeling for Software Testing, Proceedings of VALID, (2011)
11. Preschern, C., Leitner, A., Kreiner, C., Domain Specific Language Architecture for Automation Systems: An Industrial Case Study, Procs of Graphical Modeling Language Development, DTU (2012), http://www.dsmforum.org/events/GMLD12/
12. Safa, L., The Making Of User-Interface Designer, A Proprietary DSM Tool, Proceedings of the 7th Workshop on Domain-Specific Modeling, Technical Reports, TR-38, University of Jyväskylä (2007)
13. Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D., What Kinds of Nails Need a Domain-Specific Hammer?, IEEE Software, July/Aug (2009)
14. Tolvanen, J-P., Language Workbench Challenge 2012 implementation in MetaEdit+ (2012), http://www.metacase.com/support/50/repository/LWC12_MetaEdit.pdf
15. Vatjus-Anttila, J, Kreku, J., Tiensyrjä, K., Domain-specific front-end for virtual system modeling, Procs of Graphical Modeling Language Development, DTU (2012), http://www.dsmforum.org/events/GMLD12/