Legacy System Evolution through Model-Driven Program Transformation

Jing Zhang and Jeff Gray
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, Alabama, USA
{zhangj, gray} @ cis.uab.edu

Abstract

Model-Driven Software Engineering (MDSE) is gaining increased adoption in the development of computer-based systems, especially in the area of distributed real-time and embedded (DRE) domains. There is also potential to apply MDSE to assist in the evolution of large enterprise legacy systems. However, the state-of-the-art MDSE techniques still lack support for advanced processes and constructive methods involved within the context of the development and evolution of software systems. This paper outlines our position on what is needed to provide a generative approach to support legacy system evolution with MDSE. A prototype of the Model-Driven Program Transformation (MDPT) technique has been implemented to perform large scale adaptation of legacy source code from transformation rules that are driven by the evolving features of the corresponding domain models.

1. Introduction

Model-Driven Software Engineering (MDSE) techniques are being adopted with more frequency in the development of computer based systems (CBS), especially in the domain of embedded control software (e.g., avionics and automotive control systems). Meta-configurable domain-specific modeling environments [6] [8] provide support for customization of modeling tools that enable domain experts to construct models in notations that are familiar to them. Such tools typically offer the ability to generate, or synthesize various artifacts from models (e.g., synthesis of input to analysis tools, or generation of source code from the models). This is accomplished by a model interpreter whose purpose is to traverse the internal representation of a model and generate new artifacts (e.g., XML configuration files, source code, or even hardware logic). The ability to describe properties of a system at a higher abstraction level, and in a technology-independent notation, can protect key intellectual assets from technology obsolescence. Domain-specific modeling also supports rapid evolution of computer-based systems when the hardware and software configuration is tightly coupled, but must frequently evolve to a new configuration schema (e.g., retooling in an automotive factory or reconfiguration of an avionics product-line) [11].

Likewise, MDSE has the potential for broad impact when applied to the evolution and maintenance tasks of legacy software systems. However, the state-of-the-art MDSE techniques still lack support for advanced processes and constructive methods involved within the context of the development and evolution of software systems.

This paper asserts our position on the key challenges that need to be solved in order to fundamentally advance the capabilities offered by MDSE to support legacy software evolution. The paper describes a Model-Driven Program Transformation (MDPT) technique to perform large scale adaptation of legacy source from transformation rules that are generated from the evolving features of the corresponding domain models. The paper outlines a few of the challenges of providing model-driven legacy evolution, as well as a proposed solution technique with our initial results.

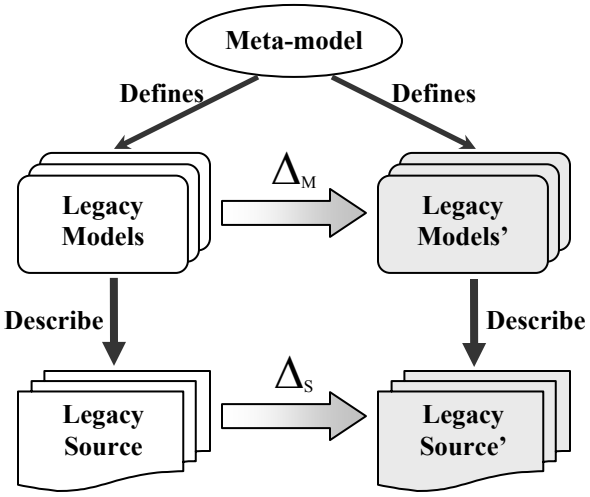
2. Technical Challenges

Legacy systems affect many aspects of our daily lives (e.g., software to control commercial transaction systems, defense systems, or even healthcare systems). As noted in [12], “any application system that is functioning in a production environment within an enterprise can be considered as a legacy system.” With increasing demands to evolve and maintain such

systems, future requirements will necessitate new strategies to support the requisite adaptations across different software artifacts.

Transformation of legacy software has many well-known technical obstacles [12]. With respect to model-driven evolution, the majority of MDSE tools are well-equipped to generate and synthesize new software artifacts. However, support for parsing and invasively transforming legacy source code from higher-level models is not well-represented in the research literature. This is because of the following three challenges:

1. It is often the case that even a slight change in the system requirements would necessitate extensive modifications that are widely spread over a large section of the source code. With the MDSE approach, this situation can be mitigated by providing a meta-configurable domain-specific modeling environment that is customized for the domain experts so that they are able to manipulate the system at a higher-level of abstraction, instead of the low-level source code. The desired result is to achieve modularization such that a change in a design decision is isolated to one location within the model. However, this approach alone cannot solve the problem because small changes in the models might still necessitate drastic changes throughout the source code. The question remains as to how the underlying existing legacy source is to be modified from the models. It is one of the key challenges for MDSE to maintain the fidelity between the mapping of model properties and the corresponding source code.
2. Many legacy systems are usually large (e.g., hundreds of thousands, or even millions, of lines of source code) and represented by a variety of programming languages. In order to transform such diverse systems, different parsers are needed for each language. In addition, if support for a new language is required, an individual new parser for this particular language must be necessarily included in the transformation toolsuite. Developing industrial-scale parsers to support all languages, and integrating them within the modeling tool, is really time-consuming (if not unfeasible).
3. Even if a mature parser is available and applicable for handling all of the languages in the underlying source, a full-blown program transformation engine is also required in order to perform the invasive [1] adaptations to the large legacy source base. This is also an arduous task.



Δ_M: The changes made to the legacy models
Δ_S: The changes reflected in the legacy source

Figure 1. The evolution of the legacy system in terms of models and source code

Figure 1 illustrates legacy system evolution in terms of models and the corresponding source code. This figure represents the concept that was originally termed a *maintenance delta* by Baxter in [2]. Generally speaking, a meta-model defines the valid concepts and rules for constructing a model in a specific domain. The meta-model is relatively stable and seldom changes during the process of the system evolution; however, the meta-model may need modifications or extension if it cannot represent concepts described in new system requirements. The legacy models are dependent on the meta-models and describe the specific designs/requirements of the underlying legacy system.

During the evolution process of a legacy system, there may exist some difference, Δ_M, between the old models and the new ones, which capture additional system requirements. Additionally, Δ_M should correspond to the difference (represented by Δ_S) between the original source code and the adaptations needed to update the legacy code. However, conventional model-driven techniques are inadequate for maintaining the consistency between a design model and the corresponding program code in the case of invasive changes to legacy code. Naturally, a robust program transformation mechanism is needed as the "bridge" to enforce the conformity between models and code. Although program transformation systems have been under development for several decades, there is little investigation into the merging of mature transformation systems within model-driven tool-suites.

3. Model-Driven Program Transformation

With respect to the three challenges enumerated in the previous section, we have been investigating the feasibility of utilizing the power of a mature program transformation system to support parsing and source-level transformation of legacy code. We provide a generative approach [4] to enforce a *causal connection* between models and the corresponding legacy source through an approach that we call model-driven program transformation (MDPT). A causal connection occurs whenever any modifications are made to the models, such that there are some accordant changes applied to the source code.

An overview of the concept for model-driven program transformation is shown in Figure 2. In our initial experiment, domain models are represented in the Generic Modeling Environment (GME) [9]. The GME is a domain-specific modeling tool that provides meta-modeling capabilities to configure the instance models from the meta-level specifications. Our approach synergistically extends the GME modeling process by incorporating the Design Maintenance System (DMS) [3] as the underlying program transformation engine. The core component of DMS is an Abstract Syntax Tree (AST) term rewriting engine that supports powerful capabilities for pattern matching and source transformation. DMS provides pre-constructed domains for several dozen languages (32 languages were supported at the time of our experimentation). These domains are very mature and have been used to parse several million lines of code in various domains. Furthermore, an important feature of DMS is the source-to-source transformation rules that can be applied to modify a large cross-section of a code base.

In the MDPT approach, the key contribution is to construct domain-specific model interpreters that are able to make the comparison between the old and new models and then produce the DMS transformation rules from the evolving features of GME models, i.e., generate the legacy source-to-source transformation delta (represented by Δ_S in Figure 1) according to the model-to-model transformation delta (represented by Δ_M). Therefore, the changes made in the models and the resulting generated transformation rules are one-to-one mappings. The corresponding legacy source code, along with the generated transformation rules, serves as the input to the underlying DMS engine. As a result, the legacy source programs will be modified and adapted to the new requirements that are reflected in the model changes.

It is worth noting that the domain experts are not required to understand the complicated DMS transformation rules. Domain experts modify the domain-specific models according to any new requirements for the system. After that, the transformation rules will be generated automatically. As a consequence, the legacy source code will be transformed into a new version. The whole process is transparent and auto-driven by the MDPT model interpreters. Detailed examples of the rules generated by the model interpreters can be found in [6].

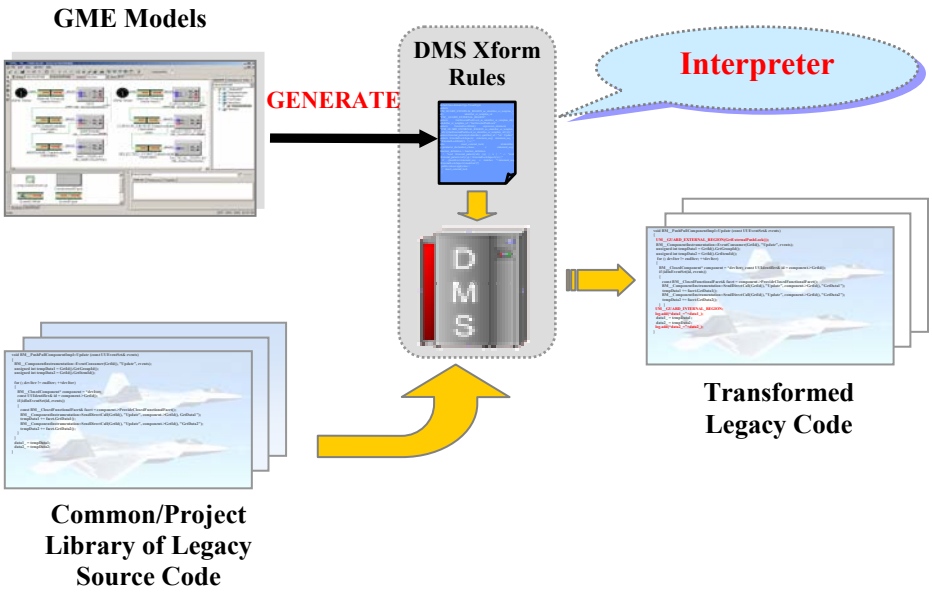


Figure 2. Overview of Model-Driven Program Transformation

The intrinsic benefit of this approach is large-scale adaptation across multiple source files that are driven by model properties. Such adaptation can be accomplished through minimal changes to the models. Such super-linearity is at the heart of the abstraction power provided by model-driven techniques [5].

Figure 3 visualizes the generalization of the process for supporting the MDPT approach. The gray squared areas indicate manual tasks that need to be performed to add new adaptation strategies within a model interpreter for a specific meta-model. The following three steps are involved:

- The system developers define and analyze the new requirements for the legacy system. If the current meta-model does not provide the proper concepts and notations to specify the new concern of interest, it has to be modified or extended to include the additional modeling concepts to support the new requirements. Developers then modify or extend the models based on the appropriate meta-model (a separate topic that also has received recent attention [10]). The resulting models reflect the evolving features of the system requirements.
- Meanwhile, the system transformers make modifications or extensions to the current MDPT model interpreter according to the new requirements specified by the developers. The updated interpreter is empowered to generate the corresponding DMS transformation rules for the new system concerns.
- As the system evolution is performed, the modified models are processed by the new MDPT interpreter. Finally, legacy source is transformed by DMS through the generated transformation rules.

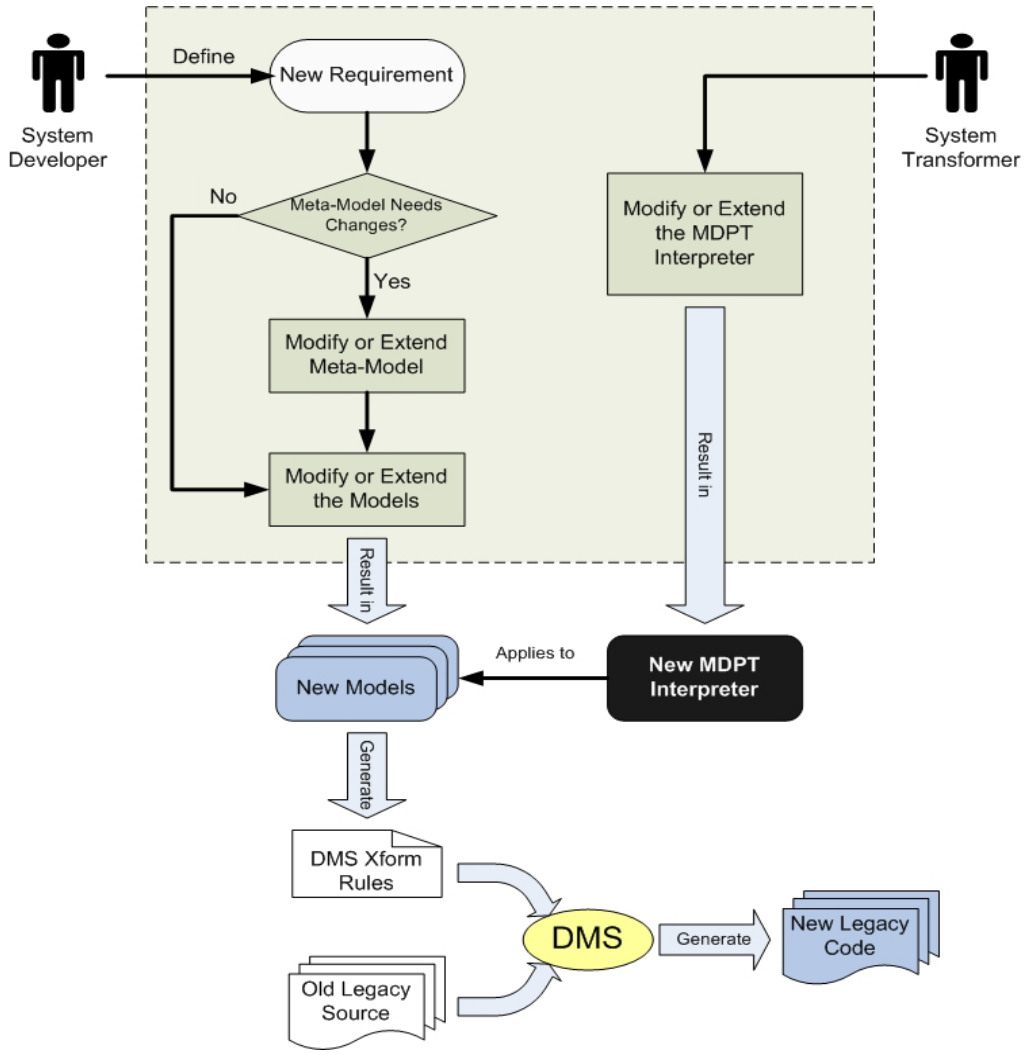


Figure 3. Generalization of the control flow for the MDPT process

4. Current Status and Future Work

An initial experiment was conducted to evaluate the feasibility of the MDPT approach. A model interpreter was developed to generate the program transformation rules needed to perform widespread source transformation of a large legacy avionics system consisting of over several millions lines of C++ source code. The transformation process provides adaptation based on Quality of Service (QoS) policies specified in the models, such as concurrency control patterns and state management. We selected a subset of this system and applied adaptations across hundreds of files that were successfully parsed and transformed in accordance with changes made in the representative models. For more technical details, please see [7]. Our initial investigation and associated prototype, however, is tailored to a specific domain with limited opportunity for extension. The future work will focus on the generalization of the process for supporting legacy system evolution through MDPT. Experimental studies will be designed to evaluate the results of this research using well-defined metrics to compare manual efforts with the proposed approaches.

The software, publications, and several video demonstrations related to this research can be obtained at http://www.gray-area.org/Research/C-SAW.

5. Acknowledgement

This work is supported by the DARPA Information Exploitation Office (DARPA/IXO), under the Program Composition for Embedded Systems (PCES) program.

6. References

[1] U. Aßmann, *Invasive Software Composition*, Springer-Verlag, 2003.

[2] I. Baxter, "Design Maintenance Systems," *Communications of the ACM*, 1992, pp. 73-89.

[3] I. Baxter, C. Pidgeon, and M. Mehlich, "DMS: Program Transformation for Practical Scalable Software Evolution," *International Conference on Software Engineering (ICSE)*, Edinburgh, Scotland, May 2004, pp. 625-634.

[4] K. Czarnecki and U. Eiseneker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[5] J. Gray, J. Sztipanovits, D. Schmidt, T. Bapty, S. Neema, and A. Gokhale, "Two-level Aspect Weaving to Support Evolution of Model-Driven Synthesis," *Aspect-Oriented Software Development*, Addison-Wesley, 2004, Chapter 30.

[6] J. Gray, J. Tolvanen, and M. Rossi, guest editors, "Special Issue: Domain-Specific Modeling with Visual Languages," *Journal of Visual Languages and Computing*, Volume 15, Issues 3-4, June/August 2004, pp. 207-209.

[7] J. Gray, J. Zhang, Y. Lin, S. Roychoudhury, H. Wu, R. Sudarsan, A. Gokhale, S. Neema, F. Shi, and T. Bapty, "Model-Driven Program Transformation of a Large Avionics Framework," *Generative Programming and Component Engineering (GPCE 2004),* Springer-Verlag LNCS, Vancouver, BC, October 2004.

[8] G. Karsai, M. Maroti, Á. Lédeczi, J. Gray, and J. Sztipanovits, "Composition and Cloning in Modeling and Meta-Modeling," *IEEE Transactions on Control System Technology* (special issue on Computer Automated Multi-Paradigm Modeling), March 2004, pp. 263-278.

[9] Á. Lédeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing Domain-Specific Design Environments," IEEE Computer, November 2001, pp. 44-51.

[10] J. Sprinkle and G. Karsai, "A Domain-Specific Visual Language for Domain Model Evolution," *Journal of Visual Languages and Computing*, vol. 15, no. 3-4, June-August, 2004, (Edited by J. Gray, M. Rossi, J.-P. Tolvanen), pp. 291-307.

[11] J. Sztipanovits, "Generative Programming for Embedded Systems," *Keynote Address: Generative Programming and Component Engineering (GPCE),* LNCS 2487, Pittsburgh, Pennsylvania, October 2002, pp. 32-49.

[12] W. Ulrich, *Legacy Systems: Transformation Strategies*, Prentice-Hall, 2002.