Supporting Software Evolution through Model-Driven Program Transformation

Jing Zhang

Department of Computer and Information Sciences

The University of Alabama at Birmingham, Birmingham, AL, 35294, USA

1-205-934-5841

zhangj@cis.uab.edu

ABSTRACT

Model-Driven Software Development (MDSD) techniques are being adopted with more frequency in the engineering of computer based systems, especially in the area of distributed real-time embedded (DRE) systems. This brief summary presents two research objectives for supporting software evolution in MDSD. First, the concept of model-driven program transformation is introduced to support evolution of large legacy software systems. The second research objective that is presented involves the application of a mature program transformation engine to automate model interpreter evolution in the presence of meta-model schema changes.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques, D.2.6 [**Software Engineering**]: Programming Environments – *graphical environments* and F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems.

General Terms

Design, Experimentation, Languages.

Keywords

Model-Driven Software Development, Model-Driven Program Transformation, Software Evolution.

1. INTRODUCTION

Model-driven approaches to software development, when coupled with meta-modeling [7] and a domain-specific visual language [5], assist in capturing the essence of a large system in a notation that is familiar to a domain expert. Domain-specific modeling 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 [10]. 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.

In MDSD, an essential characteristic is the ability to generate, or synthesize, new software artifacts from domain models [8]. This is typically accomplished by associating a *model interpreter* with a particular modeling language. The purpose of a model interpreter 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 tools and techniques to support change management, which are standard in other stages of the development lifecycle, are not commonly available in existing MDSD technologies. The scientific foundations to support advanced processes and constructive methods involved in the development and evolution of MDSD are still in need. The research objectives outlined in this paper can assist in elevating models and model transformations to first-class development artifacts. The primary motivator is to facilitate the ability to evolve model interpreters, as well as a generalized process for transforming legacy software using model-driven techniques.

2. GOALS

The goals of the doctoral research outlined in this paper can be summarized by two key objectives:

- **Implementation of model-driven program transformation techniques for supporting legacy software evolution**

  MDSD has the potential for broad impact when applied to the evolution and maintenance tasks of legacy software systems. Transformation of legacy software has many well-known technical obstacles [11]. A key challenge for MDSD is maintaining the fidelity between the mapping of model properties and the legacy source code. With respect to model-driven evolution, the majority of MDSD tools are well-equipped to generate and synthesize new software artifacts. However, support for parsing and invasively transforming [1] legacy source code from higher-level models is not well-represented in the research literature. A key contribution of this doctoral research is to provide a generative approach [3] to perform widespread adaptations of legacy source from transformation rules that are generated from the domain models.

- **Evolution of model interpreters in the presence of meta-model schema changes**

  In the presence of changing stakeholder requirements, it is possible that a meta-model undergoes numerous evolutions during periods of instability. Regarding the notion of *model evolution* in terms of meta-model schema changes, an initial investigation is described in [9]. However, the more difficult problem of *model interpreter evolution* still needs to be addressed. Each evolution of the meta-model breaks the interpreter that was defined on the previous version because

OOPSLA'04, Oct. 24–28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

a model interpreter depends on the entities and relationships defined in the meta-model. Based upon the different types of operations that can be performed to change a meta-model, corresponding changes to the interpreter can be stated formally. To accomplish the goal of model interpreter evolution, our initial research has applied a mature program transformation engine to assist in automated evolution of interpreters in the presence of meta-schema changes.

3. TECHNICAL APPROACHES

A key to evolution in MDSD is the generation of program transformation rules from model specifications. The generated rules can serve as input to a program transformation system. In our research, we generate transformation rules for the Design Maintenance System (DMS) [2]. The source code for a legacy application, along with the generated transformation rules, is sent to DMS. As a result, the legacy source is transformed as the generated rules are applied by DMS. The domain expert makes changes to models using a higher-level modeling language. Those models are then interpreted to generate transformation rules that can invasively modify a large cross-section of an implementation. It should be noted that the modeler does not need to understand the transformation rule language. That process is transparent and is generated by the model interpreter. Initial experimentation has been conducted to evaluate the feasibility of the approach. More specific details are covered in [6]. The overall benefit 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 [4].

With respect to model interpreter evolution, the core capability for transforming the interpreter implementation is also provided by DMS. As the meta-model evolves, the domain expert uses a formal specification to represent each step of the model transformation. This specification language will consist of two parts: 1) a pattern description of the interpreter signature, and 2) the replacement rule to perform the transformation. The transformation specification will be used to generate the DMS rewriting rules to transform the original interpreter into a new version that matches the changes to the meta-model.

4. CURRENT STATUS AND FUTURE WORK

A domain-specific interpreter has been constructed to perform wide-scale source transformation of a large legacy avionics system from system properties described in high-level models [6]. The transformation process provides adaptation based on Quality of Service (QoS) policies specified in the models, such as concurrency control patterns and state management. Our initial investigation and associated prototype, however, is tailored to a specific domain. The focus of future research will investigate the *generalization* of a process for supporting evolution of legacy software through model-driven program transformation.

The development of an appropriate model transformation specification language is being investigated as the kernel task in order to build up a complete architecture for model interpreter evolution. There is an additional external factor that may force interpreter evolution: from our experience, it is possible that the API provided by the modeling tool itself may also change. Interpreters that are coded to a specific API must evolve in the presence of changes to the host modeling tool.

Experimental studies are being conducted to evaluate the results of this research using well-defined metrics to compare manual efforts with the proposed automated transformation approach. The software, publications, and video demonstrations related to this research project can be obtained from the following web site: http://www.gray-area.org/Research/C-SAW.

5. ACKNOWLEDGEMENT

This work is supported by the DARPA Information Exploitation Office (DARPA/IXO), under the PCES program.

6. REFERENCES

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

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

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

[4] Gray, J., Sztipanovits, J., Schmidt, D., Bapty, T., Neema, S., and Gokhale, A., "Two-level Aspect Weaving to Support Evolution of Model-Driven Synthesis," in *Aspect-Oriented Software Development*, (Robert Filman, Tzilla Elrad, Mehmet Aksit, and Siobhán Clarke, eds.), Addison-Wesley, 2004, Chapter 30.

[5] Gray, J., Tolvanen, J., and Rossi, M., 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.

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

[7] Karsai, G., Maroti, M., Lédeczi, Á., Gray, J., and Sztipanovits, J., "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.

[8] Neema, S., Bapty, T., Gray, J. and Gokhale, A., "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems," *First ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE '02)*, Springer-Verlag LNCS 2487, Pittsburgh, PA, October 2002, pp. 236-251.

[9] Sprinkle, J. and Karsai, G., "A Domain-specific Visual Language for Domain Model Evolution," *Journal of Visual Languages & Computing*, Volume 15, Issues 3-4, June/August 2004, pp. 291-307.

[10] Sztipanovits, J., and Karsai, G., "Model-Integrated Computing," *IEEE Computer*, April 1997, pp. 10-12.

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