Transformations to Automate Model Change Evolution

Yuehua Lin
Department of Computer and Information Sciences
University of Alabama at Birmingham, Birmingham, AL 35294, USA
1-205-934-5841
liny @ cis.uab.edu

Abstract

As models are elevated to first-class artifacts within the software development lifecycle, new approaches are needed to address the accidental complexities associated with current modeling practice (e.g., manually evolving the deep hierarchical structures of large system models can be error prone and labor intensive). This research poster presents a model transformation approach to automate model evolution and testing tools to improve the quality of model transformation.

Categories and Subject Descriptors D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Object-oriented design methods*, D.2.5 [**Software Engineering**]: Testing and Debugging.

General Terms Design, Languages, Verification.

Keywords Model Transformation, Model Change Evolution, Testing

1. Research Problem

As a standard modeling language proposed by OMG, the Unified Modeling Language (UML) is a general purpose language for any domain. Distinguished from UML, Domain-Specific Modeling Languages aim to specify the solution directly using rules and concepts familiar to a particular domain of end-users. Large domain-specific system models often have repetitive and nested hierarchical structures and may contain large quantities of objects of the same type. For example, a model of a distributed real-time and embedded (DRE) system can have multiple thousands of coarse grained components. Because of such structural complexity, the fundamental task of model construction and maintenance can become manually intensive and error prone. Meanwhile, a powerful justification for the use of models concerns the flexibility of system analysis that can be performed while exploring various design alternatives. This requires an ability to rapidly evolve models in a reliable manner. To mitigate these problems, one possible solution is to provide an ability to automate model evolution.

To support automation of model evolution, there are several approaches in current modeling practice. Many commercial and research toolsuites provide APIs to manipulate models. However, these APIs are usually at a low-level involving many accidental complexities. Another approach uses domain-specific model translators to specify changes in models. To improve the level of abstraction and provide domain independency, this research advocates a high-level and domain-independent transformation language to define and execute tasks of model change evolution.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

Although various model transformation approaches have been developed [2], the role of model transformation in evolving models has not been explored fully. Specifically, this research poster describes the benefits that model transformation offers in terms of dealing with the difficulties of model scalability.

Another important issue of model transformation is to ensure its correctness. There are a variety of formal methods proposed for validation and verification for models and associated transformations (e.g., model checking [5]). However, the applicability of formal methods is limited due to the complexity of formal techniques and the lack of training of many software engineers in applying them. Execution-based testing is a feasible approach to finding transformation faults without the need to translate models and transformations to formal specifications. This research investigates testing techniques to model transformation to improve the accuracy of transformation results. A model transformation testing engine provides support to execute test cases with the intent of revealing errors in the transformation specification. Distinguished from classical software testing tools, to determine whether a model transformation test passes or fails requires comparison of the actual output model with the expected model, which requires model differencing algorithms and visualization.

2. Model Transformation Research

This research is conducted within the Generic Modeling Environment (GME), which is a metamodeling environment that can be configured and adapted from meta-level specifications that describe the domain [7]. As a preliminary result of this research, the core model transformation engine (C-SAW) has been constructed as a GME plug-in [8]. The Embedded Constraint Language (ECL) is the model transformation language of C-SAW.

2.1 The C-SAW Transformation Engine

C-SAW evolved from an aspect weaver originally designed to address crosscutting modeling concerns [3], which was constructed two years prior to the initialization of OMG's Query View Transformation (QVT) request for proposal. My initial work extended C-SAW to support additional modeling types and provide new operations for model transformation. To perform a model transformation, C-SAW takes source models and ECL transformation specifications as input, and generates the target models as output by weaving changes into source models.

Syntactically, ECL is an extension of the Object Constraint Language (OCL), which is the de facto constraint language for modeling. Although OCL does not allow altering the state of models, the ECL supports an imperative transformation style. It provides operations for model navigation and selection and also transformation operations such as dynamic creation and deletion of model elements. *Aspect* and *strategy* are two kinds of modular constructs in ECL. An aspect is used to specify a crosscutting concern across a model hierarchy (e.g., multiple locations in a model). A strategy is used to specify elements of computation (e.g., transformation behaviors) that will be bound to specific model nodes defined by an aspect.

Compared to many existing model transformation languages, which focus on transformations between different domains, ECL is a small language that supports transformations within one domain to allow an in-place update where the source model becomes the target model. From our experience, many model evolution tasks can be defined concisely in ECL.

2.2 Model Scalability with C-SAW

One practical need for exploring design alternatives relates to scalability issues of the modeled system. A typical approach to address scalability is to create a base model that captures the key elements and their relationships. A collection of base models can be adorned with necessary information to characterize a specific scalability concern as it relates to how the base modeling elements are replicated and connected together. In current modeling practice, replication is usually accomplished by scaling the base model manually. This is a time-consuming process that represents a source of error, especially when there are deep interactions between model components. As an alternative to the manual process, we are investigating the idea of automated model replication through a model transformation process that scales a base model to a larger model. Recently, C-SAW has been used to perform several model scalability tasks on numerous experimental platforms [4].

3. Model Transformation Testing

To improve the quality of C-SAW transformations, testing is applied to detect errors in transformation specifications. A transformation testing engine supports execution of a finite set of test cases against a specific model and associated transformations. The basic functionality includes execution of the transformations, comparison of the actual output model and the expected model, and visualization of the test results. If there are no differences between the actual output and expected models, it can be inferred that the model transformation is correct with respect to the given test specification. If there are differences between the output and expected models, the errors in the transformation specification need to be isolated and removed.

To construct such a testing engine, there are two issues that need to be explored deeply: 1) model comparison for discovering differences between the expected model and the target output model; and 2) visualization of model differences to assist in comprehending the comparison results. Our model comparison algorithm determines whether the two models are syntactically equivalent by comparing their elements and properties. In general, the comparison starts from the top-level of the two containment models and then continues to the child sub-models. Signature (e.g., type and identifier) and structural similarity are combined to detect the mappings and differences between two models. The discovered model differences are displayed in a structural view with graphical symbols and colors to indicate the possible kinds of model differences (e.g., a missing element, or an element that has different values for some properties [1]). Further details about critical issues of model transformation testing are presented in [6].

4. Contribution and Evaluation

This research contributes to the long-term research goal of alleviating the increasing complexity of modeling large-scale, complex applications by assisting users in making changes into models correctly and rapidly. This work is distinguished from other model transformation works by the following contributions: 1) Investigating the new application of model transformation to address model scalability concerns; 2) Applying a testing process to model transformations, which assists in improving the quality of a transformation; and 3) Developing algorithms to visualize the differences among domain-specific models.

To evaluate the benefits of this model transformation research, experimental validation is being performed using several modeling languages from different domains. The results of the evaluation will help us to determine the effectiveness of C-SAW and its testing engine toward improving the capabilities to evolve large system models in a reliable manner. The assessment metrics include productivity (i.e., the ability to reduce human efforts) and accuracy (i.e., the ability to reduce errors). More details, including software and video demonstrations, can be found at the C-SAW web site [8].

Acknowledgements

This project is supported by the National Science Foundation under CSR-SMA-0509342.

References

[1] Alanen, M. and Porres, I., "Difference and Union of Models," *Proceedings of the UML Conference*, San Francisco, CA, October 2003, pp. 2-17.

[2] Czarnecki, K. and Helsen, S., "Feature-model-based characterization and survey of model transformation approaches," accepted to be published in *IBM Systems Journal*.

[3] Gray, J., Bapty, T., Neema, S., and Tuck, J., "Handling Crosscutting Constraints in Domain-Specific Modeling," *Communications of the ACM*, October 2001, pp. 87-93.

[4] Gray, J., Lin Y., Zhang J., "Automating Change Evolution in Model-Driven Engineering," *IEEE Computer (Special Issue on Model-Driven Engineering)*, February 2006, pp. 51-58.

[5] Holzmann, G. J., "The Model Checker SPIN," IEEE *Transactions on Software Engineering*, vol. 23 (no. 5), May 1997, pp. 279-295.

[6] Lin, Y., Zhang, J., and Gray, J., "A Framework for Testing Model Transformations," in *Model-driven Software Development*, (Sami Beydeda, Matthias Book, and Volker Gruhn, eds.), Springer, ISBN: 3-540-25613-X, 2005, Chapter 10, pp. 219-236, 2005.

[7] http://www.isis.vanderbilt.edu/Projects/gme/

[8] http://www.cis.uab.edu/gray/Research/C-SAW