A Comprehensive Model Transformation Approach to Automated Model Construction and Evolution

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

ABSTRACT
As models are elevated to first-class artifacts within the software development lifecycle, the task of constructing and evolving large-scale system models becomes a manually intensive effort that can be very time consuming and error prone. To address these problems, this research poster presents a comprehensive approach to model transformation. Within this approach, a high-level aspectual model transformation language is designed to specify tasks of model construction and evolution, and then a model transformation engine is used to execute transformation specifications in an automated manner. Testing and debugging tools at the modeling level are provided to assist in detecting errors in the model transformation.

Categories & Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Object-oriented design methods,* D.2.5 [**Software Engineering**]: Testing and Debugging, and D.2.6 [**Software Engineering**]: Programming Environments – *graphical environments*.

General Terms
Design, Languages, Verification.

Keywords
Modeling, Model Transformation, Testing, Debugging.

1. INTRODUCTION
As models are elevated to first-class artifacts within the software development lifecycle (e.g., to represent designs, generate tests and synthesize code), the fundamental task of model construction and evolution can become manually intensive. For example, a distributed real-time and embedded (DRE) system can have multiple thousands of coarse grained component models [3]. A manual process for evolving such complex systems can be error prone and require much time to make changes across a large model hierarchy. Tools to automate the construction and evolution of models may help to mitigate such problems. Many commercial and research toolsuites provide APIs to manipulate models. However, an API approach requires model developers to learn and use low-level tools to program their tasks of transforming high-level models. To improve the level of abstraction in constructing and evolving models, this poster describes a high-level model transformation language and associated tools for automated model construction and evolution.

In our model transformation approach, tasks of model construction and evolution are specified in a transformation language (called the transformation specification) and then a model transformation engine is used to execute transformation specifications in an automated manner. However, a transformation specification is written by humans and susceptible to errors. Additionally, a transformation specification may be reusable across similar domains. Therefore, it is essential to ensure the correctness of the transformation specification (i.e., the consistency and completeness, as validated against model transformation requirements) before it is applied to a collection of models. An objective of this research is to apply testing and debugging techniques to model transformations to assist in improving the accuracy of transformation results. The transformation testing defined in this research involves executing a transformation specification with the intent of revealing errors by comparing the actual output model with an expected model. After determining that there are errors in a model transformation specification, a debugging tool at the modeling level assists in identifying the errors by stepwise execution of a transformation specification. Currently, a few transformation tools provide limited debugging facilities (e.g., GReAT [1]). However, there are no reports in the literature regarding efforts that provide the facilities for transformation specification testing.

The primary contribution of this research is an investigation into automated model construction and evolution through model transformation that considers additional issues of testing and debugging to assist in determining the correctness of model transformation. Other related contributions include addressing the critical issue of model comparison algorithms and visualization of model differences, which also assists in version control of models.

2. RESEARCH OVERVIEW
This research is conducted within a meta-modeling tool known as the Generic Modeling Environment (GME) [6]. Our core model transformation engine is the Constraint-Specification Aspect Weaver (C-SAW), which has been constructed as a GME plug-in and applied to various large system models. Initial progress on transformation testing and debugging is described in [5].

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

2.1 Model Transformation Engine: C-SAW

Originally, C-SAW was designed to address crosscutting modeling concerns, but has evolved into a general model transformation engine. When performing a transformation with C-SAW, one input is a source model, and another input to C-SAW is the transformation specifications written in the Embedded Constraint Language (ECL). These specifications are executed by C-SAW to weave changes into source models and generate the target models.

The ECL supports an imperative transformation style, which provides features such as collection and model navigation, as well as a rich set of operators to support model aggregations, connections and transformations. An ECL transformation specification usually consists of two kinds of modular units: aspect and strategy. An aspect is a modular construct that specifies a crosscutting concern across a model hierarchy. 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. The following code segment is an aspect example:

```
aspect ImplModels ( )
{  rootFolder().findFolder("Components").
    models()→select(m|m.name().endWith("Impl"))
                    →AddConcurrency();
}
```

This aspect selects all the models whose names end with "Impl" from a folder called "Components." A strategy called AddConcurrency (not shown here) is then applied to the selected models. Using the aspect and the strategy constructs in ECL, a task of model construction or evolution can be specified as a model transformation process.

Within C-SAW, there is a parser and an interpreter for ECL. The parser is responsible for generating an abstract syntax tree (AST) of the ECL specification. The interpreter then traverses this generated AST from top to bottom, and interprets it to perform a transformation by using modeling APIs provided by GME. Thus, the accidental complexities of using the low-level details of the GME API are abstracted in the ECL to provide a more intuitive representation for specifying model transformations.

2.2 Transformation Testing and Debugging

Transformation specification testing is needed to assist in finding the errors in transformation specifications. The transformation specification test engine has three components: the executor, the comparator and the test analyzer. The executor is responsible for executing the to-be-tested specification on the input model to generate the output model. The comparator computes the mappings and differences between the output model and the expected model. A test analyzer visualizes the model differences to assist in comprehending the comparison and provides a capability to navigate among any differences. 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 specification. If there are differences between the output and expected models, the errors in the transformation specification need to be isolated and removed.

As an initial solution to model comparison, our algorithm will only determine whether the two models are syntactically equivalent by comparing all the elements and their properties within these models. A possible solution to visualization of model differences is to use graphical symbols and colors to indicate all possible kinds of model differences (e.g., a missing element, or an element that has different values for some properties [2]).

After determining that an error exists in a model transformation, a debugging tool can offer support for isolating the cause of a transformation error. A model transformation debugger must understand the model representation, as well as possess the ability to step through individual lines of the transformation specification to display the model data intuitively within the host modeling environment.

3. RESULTS AND EVALUATION

This work is being experimentally validated on a mission computing avionics application provided by Boeing, which contains over three million lines of C++ code. C-SAW is helping to evolve this code base through model transformations [4]. Recently, C-SAW has been used in addressing the important issue of model scalability [3]. Furthermore, we plan to apply C-SAW to several other research projects for rapid model construction and evolution. The feedback of these experiments will assist in evaluating C-SAW's ability to automate model construction and evolution in various domains for specific types of transformations (e.g. reduced time, increased accuracy and usability). More details about C-SAW can be found at the following project website: http://www.cis.uab.edu/gray/Research/C-SAW

ACKNOWLEDGEMENTS

This project was previously funded by the DARPA Program Composition for Embedded Systems (PCES) program, and currently supported by the National Science Foundation under CSR-SMA-0509342.

REFERENCES

[1] Agrawal, A., Karsai, G., and Lédeczi, A., "An End-to-End Domain-Driven Software Development Framework," *18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Anaheim, CA, October 2003, pp. 8-15.

[2] Alanen, M. and Porres, I., "Difference and Union of Models," *UML Conference*, Springer-Verlag LNCS 2863, San Francisco, CA, October 2003, pp. 2-17.

[3] Gray, J., Lin, Y., Zhang, J., Nordstrom, S., Gokhale, A., and Neema, S., "Replicators: Transformations to Address Model Scalability," *Model Driven Engineering Languages and Systems (MoDELS)*, Montego Bay, Jamaica, October 2005.

[4] Gray, J., Zhang, J., Lin, Y., Wu, H., Roychoudhury, S., Sudarsan, R., Neema, S., Shi, F., and Bapty, T., "Model-Driven Program Transformation of a Large Avionics Application," *Generative Programming and Component Engineering (GPCE 2004)*, Springer-Verlag LNCS 3286, Vancouver, BC, October 2004, pp. 361-378.

[5] Lin, Y., Zhang, J., and Gray, J., "A Framework for Testing Model Transformations," *Model-driven Software Development,* Springer, Chapter 10, pp. 219-236, 2005.

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