MT-Scribe: A Flexible Tool to Support Model Evolution

Yu Sun
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294
yusun@cis.uab.edu

Jeff Gray
Department of Computer Science
University of Alabama
Tuscaloosa, AL 35487
gray@cs.ua.edu

Jules White
Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, VA 24061
julesw@vt.edu

ABSTRACT
Model evolution is an essential activity in software system modeling, which is traditionally supported by manual editing or writing model transformation rules. However, this process presents challenges to those who are unfamiliar with a model transformation language or abstract syntax definitions. This demonstration presents an approach to ease the implementation of model evolution tasks by recording and analyzing demonstrated end-users behavior.

Categories and Subject Descriptors
D.2.2-2.6 [**Software Engineering**]: Design Tools and Techniques; Programming Environments; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms
Algorithms, Design, Languages.

Keywords
Model evolution, demonstration, MT-Scribe.

1. INTRODUCTION
As a high-level representation for software artifacts, models play an increasingly significant role in the whole software lifecycle. Instead of drawing models on paper and white boards, a number of formal and informal modeling tools have been developed to support various types of modeling activities, ranging from basic functions such as visual editing and syntax / semantic checking, to the more advanced feature of aiding code generation in the context of Model-Driven Engineering [1].

During the modeling process, models must inevitably change and evolve for different purposes. For instance, a system requirement model has to be modified occasionally to adapt the new requirements from the end-users; a system design model is often changed to evaluate the different design decisions or to optimize its internal structure (i.e., refactoring); if a software system is generated directly from models, all of its maintenance activities might involve changing its high-level models. Therefore, model evolution has become an indispensible activity when using models to support software development.

However, the tools to support model evolution are not well-developed as other general model tools. In most situations, manual evolution has to be done through the basic operations provided by the editing environment, which is tedious, error-prone and time consuming, particularly when a large amount of model elements need to be changed (e.g., scaling up a model by adding one thousand new elements). An alternative to efficiently evolve models is to use model transformation languages [3], because model evolution is actually a type of model transformation (i.e., transform a model from one configuration to another configuration, or from an old state to a new state). Using model transformation languages, a set of transformation rules can be defined to specify how a source model should be changed and evolved to the desired target model. Executing the rules leads to an automatic model evolution process.

Although model transformation languages are very powerful and expressive to handle various kinds of model evolution tasks, using transformation languages is not always the perfect solution, due to the steep learning curve of the languages and the need to deeply understand the abstract syntax or semantics of the models (e.g., the metamodel definitions). Because many potential model users (e.g., requirements engineers, domain experts) are not necessarily software engineers or programmers, learning transformation languages and understanding the formal syntax definitions may be beyond their capability. Such inflexibility may prevent some users from realizing model evolution tasks for which they have extensive domain experience.

To overcome such problems and to simplify the realization of model transformation to support model evolution, we have been investigating the idea of Model Transformation By Demonstration (MTBD) [6]. Tool support, such as presented in this demonstration, can assist general model users (e.g., domain experts and non-programmers) in realizing model evolution in a flexible way, without knowing a specific model transformation language or the abstract syntax definition.

2. RELATED WORK
To address the challenges inherent in using model transformation languages, Model Transformation By Example (MTBE) has been proposed [4][5]. Rather than writing transformation rules manually, MTBE enables users to setup interrelated mappings between the source and target model instances, and the transformation rules are semi-automatically generated.

However, the current state of MTBE research still has several limitations that prevent it from being a widely used modeling approach to support model evolution. The semi-automatic generation often leads to an iterative manual refinement of the

generated rules; therefore, the model transformation designers are not isolated completely from knowing the transformation languages and the metamodel definitions. In addition, the inference of transformation rules depends on the given sets of mapping examples (i.e., the model inference is only as good as the seeded examples). In order to obtain a complete and precise inference result, one or more representative examples must be available for users to setup the prototypical mappings, but seeding the process with such examples is not always an easy task in practice. Furthermore, current MTBE approaches focus on mapping the corresponding concepts between two different domains without handling complex attribute transformations. For instance, in practice, it is quite common to transform an attribute in the source model to another in the target model with some arithmetic or string operations, which is expressed by imperative transformation rules in a transformation language. Unfortunately, these imperative expressions can only be added manually to the generated rules using current MTBE approaches. The related work mentioned here primarily has been applied to exogenous model transformation (i.e., transformation of model instances from different metamodels), but they are not as beneficial for inferring the refinements that are typical of model evolution tasks where the source and target models are from the same domain.

Another work has been described by Brosch et al. in [7], which uses an example-based approach to address model transformation tasks. Because it supports model transformation within the same domain, it also has potential to be applied in model evolution scenarios. However, the user feedback step may not be at the proper level of abstraction in their approach, and complex attribute transformation is not provided.

3. MTBD SOLUTION

The MTBD idea derives from MTBE. Instead of inferring the rules from a set of interrelated mappings between the source and target models, users are asked to demonstrate how the model transformation should be done by directly editing (e.g., add, delete, update) the source model to simulate the transformation process step-by-step and changing it into the desirable target model. A recording engine captures all of the user's operations during the demonstration, and then the inference engine infers the user's intention and generates a transformation pattern from the recorded operations. This generated pattern can be reused and executed by the engine in any model instance to carry out the model transformation.

As an implementation of the MTBD idea, MT-Scribe is an Eclipse plug-in for GEMS (Generic Eclipse Modeling System) [8]. It consists of the following main steps, as shown in Figure 1.

Step 1 – User Demonstration and Recording. Users first give a demonstration by directly editing a model instance (e.g., add a new model element or connection, modify the attribute of a model element, connect two model elements) to simulate an evolution task. During the demonstration, users are expected to perform operations not only on model elements and connections, but also on their attributes, so that the attribute evolution can be supported. An attribute refactoring editor has been developed to enable users to access all the attributes in the current model editor and specify the desired transformation (e.g., string and arithmetic computation). At the same time, an event listener is developed to monitor all the operations occurring in the model editor and collect the information for each operation in sequence.

Step 2 – Operation Optimization. The list of recorded operations indicates how a model evolution should be performed. However, not all operations in the list are meaningful. Users may perform useless or inefficient operations during the demonstration. For instance, without a careful design, it is possible that a user first adds a new element and modifies its attributes, and then deletes it in another operation later, with the result being that all the operations regarding this element actually did not take effect in the transformation process and therefore are meaningless. Thus, after the demonstration, the engine optimizes the recorded operations to eliminate any meaningless actions.

Step 3 – Pattern Inference. With an optimized list of recorded operations, the transformation can be inferred. Because the proposed approach does not rely on any model transformation languages, it is not necessary to generate specific transformation rules, although that is possible. Instead, we generate a transformation pattern, which summarizes the precondition of a transformation (i.e., *where* a transformation should be done) and the actions needed in a transformation (i.e., *how* a transformation should be done).

Step 4 – User Refinement. The initial pattern inferred is specific to the demonstration and is usually not generic and accurate enough, due to the limitation on the expressiveness of the user demonstration, so users are permitted to refine the inferred transformation by providing more feedback for the desired transformation scenario. For instance, users could give more restrictive preconditions on the desired evolution, such as replace element *A* only if *A* has no incoming or outgoing connections, add new element *B* in *C* only when the attribute value of *C* is greater than *200*. Users can also identify which operations should be generic (i.e., operations should be repeated as long as appropriate model elements are available, rather than being executed only once). All the user refinements are still performed at the model instance level without explicitly modifying the metamodel, after which a transformation pattern will be finalized and stored in the pattern repository for future use.

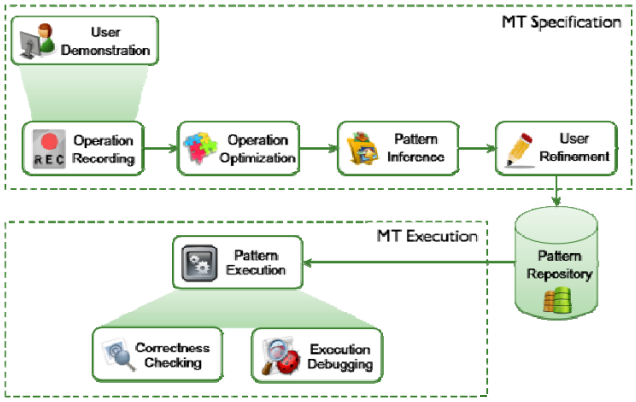


Figure 1. Overview of MTBD

Step 5 – Pattern Execution. The final generated patterns can be executed on any model instances. Because a pattern consists of the precondition and the transformation actions, the execution starts with matching the precondition in the new model instance and then carries out the transformation actions on the matched locations of the model.

Step 6 – Correctness Checking and Debugging. Although the location matching the precondition guarantees that all

transformation actions can be executed with necessary operands, it does not ensure that executing them will not violate the syntax, semantics definitions or external constraints. Therefore, the execution of each transformation action will be logged and the model instance correctness checking is performed after every execution. If a certain action violates the metamodel definition, all executed actions are undone and the whole transformation is cancelled. Finally, an execution debug has been proposed as part of MTBD to aid detecting errors in model transformations.

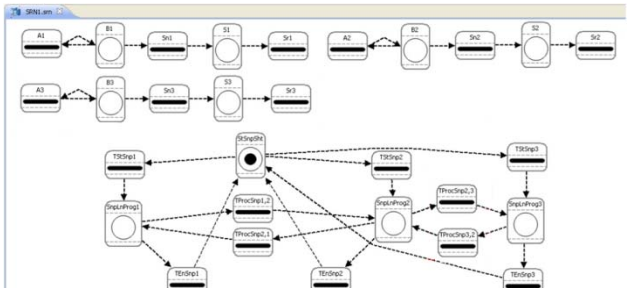
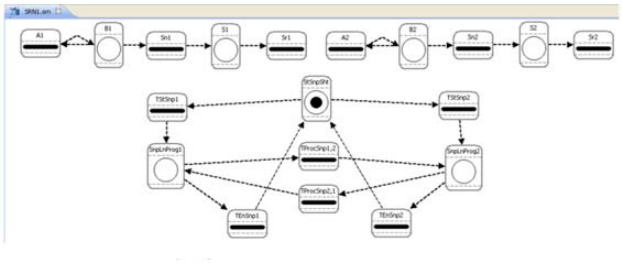


Figure 2a. Demonstrate a model evolution task



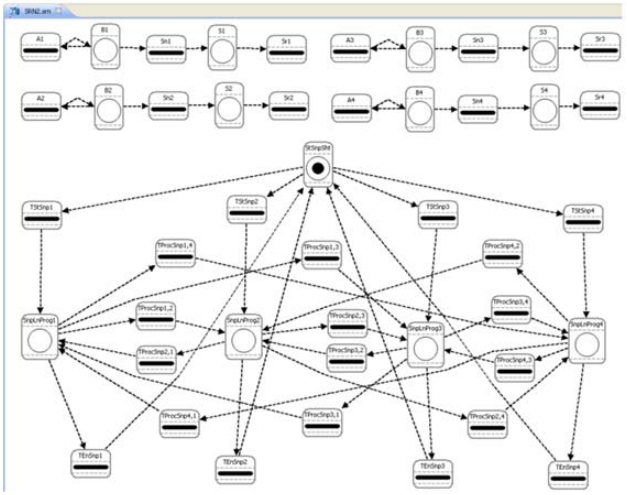


Figure 2b. Apply the generated pattern to evolve a model instance from one state (upper) to a new state (lower)

4. DEMONSTRATION OVERVIEW

The demonstration will be structured to contain the following parts through both a PowerPoint presentation and a live example of several case studies demonstrating use of MT-Scribe.

Part 1 - Motivation for MT-Scribe. The background information about model evolution and the traditional approaches to support model evolution will be given, followed by an introduction to the problems associated with manual evolution and using model transformation languages. We will also briefly introduce MTBE and its limitations to highlight the motivation of MT-Scribe. A model evolution sample problem in a simple domain will be used as a motivating example.

Part 2 - Overview of MT-Scribe. We will give an overview of MT-Scribe, including the basic MTBD idea and its main steps. Then, we will demonstrate how to use MT-Scribe to solve the motivating example. The demonstration will explain the main implementation details including the GEMS modeling platform, the architecture of MT-Scribe, and algorithms used.

Part 3 - More Demos of MT-Scribe. The presentation will proceed by demonstrating several representative model evolution tasks in different domains. Each example will be done from the demonstration, to applying the generated pattern on a new instance to support the desired evolution task as shown in Figure 2. We will provide MT-Scribe solutions for common model evolution tasks in model refactoring, model scalability [2], and model layout management. The modeling domains range from the general-purpose modeling area such as UML, to domain-specific modeling scenarios such as embedded systems and cloud computing.

Part 4 - Summary of MT-Scribe. The presentation will conclude by summarizing the tool, and identifying its advantages and disadvantages. The future work and research direction will be mentioned as well.

5. ACKNOWLEDGEMENT

This work is supported by an NSF CAREER award (CCF-0643725).

6. REFERENCES

[1] Schmidt, D.: Model-driven engineering. IEEE Computer, vol. 39, no. 2, pp. 25-32, 2006

[2] Gray, J., Lin, Y., Zhang, J., Nordstrom, S., Gokhale, A., Neema, S., Gokhale, S.: Replicators: Transformations to address model scalability. In Proceedings of the International Conference on Model-Driven Engineering Languages and Systems, Montego Bay, Jamaica, pp. 295-308, 2005.

[3] Sendall, S., Kozaczynski, W.: Model transformation - The heart and soul of model-driven software development. IEEE Software, Special Issue on Model Driven Software Development, vol. 20, no. 5, pp. 42-45, 2003.

[4] Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. Software and Systems Modeling, vol. 8, no. 3, pp. 347-364, 2009.

[5] Strommer, M., Wimmer, M.: A framework for model transformation by-example: Concepts and tool support. In Proceedings of the 46th International Conference on Technology of Object-Oriented Languages and Systems, Zurich, Switzerland, July 2008, pp. 372–391.

[6] Sun, Y., White, J., Gray, J.: Model transformation by demonstration. In Proceedings of International Conference on Model Driven Engineering Languages and Systems, Denver, CO, pp. 712-726, 2009.

[7] Brosch, P., Langer, P., Seidl, M., Wieland, K., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W.: An Example is Worth a Thousand Words: Composite Operation Modeling By-Example. In Proceedings of International Conference on Model Driven Engineering Languages and Systems, Denver, CO, 2009.

[8] Generic Eclipse Modeling System (GEMS). http://www.eclipse.org/gmt/gems/