Supporting Model Evolution through Demonstration-based Model Transformation

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

Abstract

Model evolution is often supported by writing model transformation rules in specialized languages. This presents challenges to those who are unfamiliar with model transformation languages or metamodel definitions. This poster describes research that simplifies the creation of model evolution transformations by recording and analyzing the operational behavior exhibited by end-users.

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 transformation, demonstration, MT-Scribe.

1. Background and Motivation

With the rapid development of Model-Driven Engineering (MDE), managing complex change evolution within the model representation is becoming increasingly important. Manual model evolution is often tedious and error-prone. Model transformation provides an efficient way to automate complex model evolution tasks, transforming a model from one state (i.e., configuration) to another [1].

The traditional and common approach toward implementing model transformations is to specify the transformation rules and automate the transformation process by using an executable model transformation language [1]. Although most of these languages are already powerful enough to implement large-scale and complex model transformation tasks, they may present some challenges to users, particularly to those who are unfamiliar with a specific transformation language. Firstly, even though declarative expressions are supported in most model transformation languages, they may not be at the proper level of abstraction for an end-user, and may result in a steep learning curve and high training cost. Moreover, the transformation rules are usually defined at the metamodel level, which requires a clear and deep understanding about the abstract syntax and semantic interrelationships between the source and target models. In some cases, domain concepts may be hidden in the metamodel and difficult to unveil [2] (e.g., some concepts are hidden in attributes or association ends, rather than being represented as first-class entities). These implicit concepts make writing transformation rules challenging. Thus, the difficulty of specifying transformation rules at the metamodel level and the associated learning curve may prevent some domain experts from building model transformations for which they have extensive domain experience.

The research described in this poster aims to simplify the implementation of model transformations, so that general users (e.g., domains experts and non-programmers) are enabled to realize model evolution tasks without knowing a specific model transformation language or the metamodel definition.

2. Limitations of Related Work

Model Transformation By Example (MTBE) [3] is an innovative approach to address the challenges inherent from using model transformation languages. Instead of writing transformation rules manually, users are asked to build a prototypical set of interrelated mappings between the source and target model instances, and then the metamodel-level transformation rules will be semi-automatically generated. Because users configure the mappings at the model instance level without knowing any details about the metamodel definition or the hidden concepts, combined with the generated rules, the simplicity of specifying model transformations can be improved using MTBE. Varró and Balogh first introduced the MTBE idea in [3] and later proposed an implementation framework by using inductive logic programming [4], [5]. The user-defined interrelated mappings between the source and target model instances are first transformed to logic rules and facts. Through an inductive logic programming engine, the metamodel-level mapping rules can be inferred. Similarly, Strommer and Wimmer implemented an Eclipse prototype to generate ATL transformation rules from the user-defined semantic mappings between domain models [2], [6], [7]. Rather than applying a logic programming engine, their inference and reasoning process was based on pattern matching.

The current status of MTBE suffers from several problems, preventing it from being a more accepted model transformation approach. Firstly, it only supports semi-automatic generation of rules, requiring further user refinement, which means that users cannot be fully isolated from the knowledge of the transformation languages and metamodel definition. Furthermore, the quality of the inferred transformation rules depends on the availability of suitable source and target model instances that can be used to perform the semantic mapping. Also, current MTBE approaches only focus on directly mapping the corresponding domain concepts between two different metamodels without handling complex attribute transformations (e.g., an attribute in the source model is transformed to another in the target model by some arithmetic operations). Ignoring this function will undermine the power of model transformations.

OOPSLA 2009, October 25-29, 2009, Orlando, FL, USA.
ACM 978-1-60558-768-4/09/10.

3. Solution Approach: MTBD

To further simplify the realization of model transformations, this poster introduces a new approach – Model Transformation By Demonstration (MTBD), which derives from the idea of 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. The source model will be changed into the desirable target model after the demonstration. During the demonstration process, a recording and inference engine captures all the user operations and infers the user's intention in a model transformation task. Finally, a transformation pattern is generated from the inference, specifying 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). This generated pattern can be executed by the engine in any model instance to carry out the model transformation. It can also serve as an intermediate representation that could be used to generate other transformation rules and code in different model transformation languages.

The prototype of MTBD, a tool called MT-Scribe, has been implemented as a plug-in for GEMS (Generic Eclipse Modeling System) [8], which is a modeling tool hosted within Eclipse. The MTBD process consists of five main steps. In the first step, users edit a model instance to demonstrate a transformation task. A recording engine has been developed to completely capture all user operations and related context. The recorded operations are optimized to eliminate those meaningless operations (e.g., users first add one element and then delete it later. In this case, both *add* and *delete* do not change the model and therefore are meaningless). In the third step, by analyzing the recorded operations, the inference engine will generate the transformation pattern, which specifies the weakest precondition and the transformation actions. The fourth step is to reuse the generated transformation pattern when needed. We have designed an algorithm and realized an execution engine to automatically match and execute the generated transformation pattern in any model instances. Finally, a correctness checking and undo mechanism will assist in confirming the correctness of executing transformation patterns. If the execution of a transformation pattern leads to a violation of the metamodel definition, the execution will be terminated and the model instance will be restored to its original state.

4. Results and Contribution

In our approach, users are only involved in editing a model instance to demonstrate the transformation process. All of the other procedures (i.e., optimization, inference, generation, execution, and correctness checking) are automated without any manual refinement. No model transformation languages are used and the generated transformation patterns are invisible to users. Therefore, users are completely isolated from knowing a model transformation language and the metmodel definition. In addition, complex attribute operations can be inferred as well, making our approach more transparent to end-users when compared to current MTBE approaches.

We have applied our approach to successfully implement several practical model evolution tasks in different domains without writing any transformation rules or codes, showing improvement in the efficiency and simplicity of specifying model transformations. One example is the MazeGame domain, which describes the structure and organization of a maze game. Various evolution transformations like *Balance Power Items*, *Simplify a Game* can be realized easily by demonstration. Additionally, we have used our approach to infer typical UML refactoring transformations, such as *Extract Subclass*, *Push Down Method*. More detailed description about the examples and associated video demos are available at the project's web site: http://www.cis.uab.edu/softcom/mtbd.

In addition to facilitating model refactoring, the current MTBD approach can also contribute to other areas related with model evolution. For instance, in Aspect-Oriented Modeling (AOM), MTBD can demonstrate the process of weaving a crosscutting concern to the base model to simplify aspect weaving; model scalability can be simplified as well through demonstrating the scaling process and generating the scaling pattern. As future work, implementing MTBD for exogenous model transformations (i.e., transformations between two different domains or metamodels) will simplify more applications of model engineering.

In the current version of MT-Scribe, one limitation is that only the basic or the weakest precondition can be inferred. Additionally, the attribute operations currently supported are only basic arithmetic operations and string concatenation, without more powerful operations and functions (e.g., max() and min()). Hence, to make MTBD more practical, these additional attribute operations should be supported in the demonstration process, and represent areas of future work.

Acknowledgement

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

References

[1] 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, Sep./Oct. 2003.

[2] Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In Proceedings of the 40th Hawaii International Conference on Systems Science, Big Island, HI, January 2007, pp. 285.

[3] Varró, D.: Model transformation by example. In Proceedings of Model Driven Engineering Languages and Systems, Genova, Italy, October 2006, pp. 410–424.

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

[5] Varró, D., Balogh, Z.: Automating model transformation by example using inductive logic programming. In Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea, March 2007, pp. 978–984.

[6] 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.

[7] Strommer, M., Murzek, M., Wimmer, M.: Applying model transformation by-example on business process modeling languages. In Proceedings of Third International Workshop on Foundations and Practices of UML, Auckland, New Zealand, November 2007, pp. 116–125.

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