Reflecting on Higher Order Transformations: Challenges and Opportunities

Olaf Muliawan and Dirk Janssens

University of Antwerp, Middelheimlaan 1
2020 Antwerp, Belgium

Abstract. The area of Model Driven Engineering focuses on the transformation of models into source code. However, large projects require complex transformation patterns which are difficult to implement and maintain. New language features could represent often used transformation patterns. However, extending a transformation language is not the preferred solution to keep the language concise. Therefore we introduce the notion of *Higher Order Transformations* that manipulate these new language features and transform them back in the original language. In this paper we will explain the challenges of using Higher Order Transformations and the opportunities these techniques provide.

1 Introduction

Model Driven Engineering (MDE) has matured over the years, and current projects involve complex model transformation patterns. The existing model transformation languages are quite low-level and basically support create, update and delete operations on model elements. Common but more complex operations, such as copying a subset of model elements or creating association links between UML classes, are a number of examples which can be expressed using low level constructs. However, if these patterns are numerous, their specification is cumbersome and time-consuming.

For the convenience of users, new language features representing these complex patterns can be introduced. However, to avoid extending the original transformation language, the features are transformed back into the original language. These actions are called *Higher Order Transformations* (HOTs). Sometimes there is no access to the original source code of the transformation tool and HOTs provide a means of introducing new language features with the possibility of considering the transformation tools black-box. More important, these transformations are done using existing transformation tools and do not incur extra development time to support HOTs. This is possible if the transformations can be considered as transformation models. Since model transformation tools can transform a model given its meta-model, a transformation model is treated like a normal transformation.

We believe HOTs are useful in complex and large transformations to simplify their specification and maintain transformations in a more efficient manner. However, challenges exist: a chain of HOTs creates the problem of dependencies and order of the transformations. Another challenge is assuring that the mapping is possible and that a HOT exists. Merely adding a new feature without providing automated support through transformation is not a desired outcome.

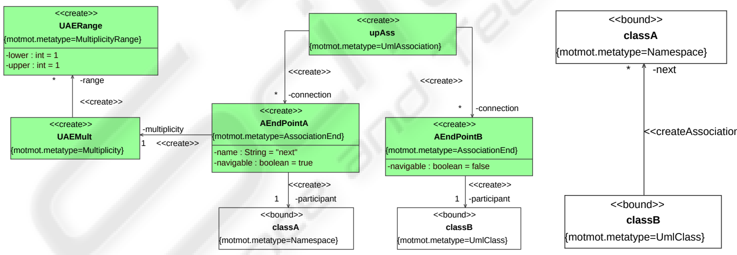
Muliawan O. and Janssens D. (2009).
Reflecting on Higher Order Transformations: Challenges and Opportunities.
In *Proceedings of the 1st International Workshop on Future Trends of Model-Driven Development*, pages 52-55
DOI: 10.5220/0002203000520055
Copyright © SciTePress

2 Simplifying Transformations: Case Studies

To illustrate the use of HOTs, we will focus on a few case studies that we already implemented and presented in earlier papers. The case studies present problems where the introduction of a new language feature improves the conciseness and readability of the transformation. The transformation language employed in this paper is called *Story Driven Modeling* (SDM) implemented in MoTMoT [1]. This is a language based on graph transformation techniques but with an explicit control flow determining which graph transformations are executed. The basic functionality is limited to create, delete and update operations on model elements although the control flow allows greater flexibility to determine when and if transformations are performed.

Case 1: Creating and Deleting Associations in UML Class Diagrams. UML class diagrams are often used within MDE to create software working on databases or web applications. However the creation and deletion of association links involves more model elements than just the association link itself. Observe figure 1(a): the association ends with matching attributes have to be explicitly enumerated, and even multiplicities are separate model elements.

A more concise manner which does not clutter the transformation diagram is drawing an association with a new introduced stereotype <createAssociation >>or <<deleteAssociation >>and putting the information about the association ends on the ends of the drawn association, as seen on figure 1(b). The attribute values regarding the association still have to be filled in (in this example we omitted other possible elements on the association such as tagged values and stereotypes which would further increase the number of model elements without the use of the new language feature), however the use of the language feature signals the transformation tool to create all necessary model elements to represent the association in a correct manner.



(a) Representation with primitive constructs (b) HOT construct

Fig. 1. Creating a new UML association using HOT constructs.

Transformations regarding class diagrams are more readable because one can see an explicit association between two classes instead of analyzing all model elements pertaining the association to get an accurate view of which association is represented.

Case 2: Declarative Language Features in SDM. In this case we use the JDT2MDR tool to produce an XMI-compliant model from Java source code [2] to implement three refactorings: move method, pull up method and encapsulate field. The *move method* refactoring requires one step where references to the moved method are updated. According to the specification of the refactoring there are four possibilities, though only one of them should be applied. Moreover, the order in which these possibilities are traversed is not specified. MoTMoT is imperative, so an order is imposed, even if it is not intentional. Each option which does not match leads to the following option being evaluated until one (or none) is matched.

Meyers et al. introduced a new construct which is called *ND-state* (non deterministic state) [3]. Transformation actions in this state are randomly chosen and in our case, if one of the actions matches, the rest of the control flow is carried out. If it does not match, the transformation tool keeps looking for other options until all are exhausted. This language feature does not impose an order on the traversed possibilities which is exactly the semantic meaning of this particular step in the *move method* refactoring. The language feature is quite powerful and incorporates such notions as having exactly n number of options matched during evaluation (can also be none or all options), layered evaluation (begin evaluating the first layer of options, then the second, ...).

Case 3: Support for ”Downcasting” in Model Transformations. The following example is related to the transformation of visual process models (UML activity diagrams) into low-level algebraic (CSP) programs that support formal verification [4]. In this case a new language feature is introduced which mimics downcasting: a model element is first recognized as of a general type. Only in a later stage this model element is established as of a more specific type.

Nonetheless, SDM assumes strict static typing and the first type declared to be the final and only possible type for the model element. The implementation in MoTMoT does allow for downcasting, although it is not immediately supported through the SDM language. The new language feature makes this functionality available for model transformation specifications.

3 Challenges

Optimizing and analyzing HOTs are some of the challenges we will have to face. More specifically the mapping problem for HOTs is more poignant because the model output serves as input for the transformation tool. In this regard tools are necessary which check the strict consistency of the output. The use of several new language features introduces the problem of order of execution (preferably confluence is possible in which case the order of execution is irrelevant).

4 Related Work

The idea of HOTs has been proposed by Varró et al [5]. In addition, Bézivin et al. have also suggested treating transformations as models that are subject to further transformations which reinforces our approach in this direction [6, 7]. Miguel et al [8] and Agrawal [9] have both investigated complex model transformations.

5 Conclusions

We presented a few case studies wherein we encountered the need for new language features. Each feature was implemented using Higher Order Transformations. We have also illustrated that using our transformation tool, so no additional tool support is necessary to achieve these goals. The introduction of language features increasing the transparency, readability and understandability for model transformations is done without adding bloat to the transformation tool. Using HOTs a given model transformation can use selected new language features which are transformed back in the original language.

We laid out some challenges that require more research. However, we believe the approach of Higher Order Transformations is promising to improve efficiency when setting up the model transformations and providing new language features is done in a non-intrusive manner without explicit extensions to the transformation language.

Acknowledgements

This work has been sponsored by the Belgian National Fund for Scientific Research (FWO) under project G.0422.05 on the 'Formal Support for the Transformation of Software Models' and the Belgian Science Policy (Belspo) under the project on 'Modeling, Verification and Evolution of Software' (MoVES) as part of the IAP-Phase VI Interuniversity Attraction Poles Programme.

References

1. Schippers, H., Gorp, P.V., Janssens, D.: Leveraging uml profiles to generate plugins from visual model transformations. Electr. Notes Theor. Comput. Sci. 127 (2005) 5–16
2. Muliawan, O., Bois, B.D., Janssens, D.: Refactoring using jdt2mdr, an industrial based solution. In: 4th International Workshop on Graph-Based Tools (GraBaTs) on the 4th International Conference on Graph Transformation (ICGT). (2008)
3. Meyers, B., Gorp, P.V.: Towards a hybrid transformation language: Implicit and explicit rule scheduling in story diagrams. In: Sixth International Fujaba Days (FD08). (2008)
4. Muliawan, O., Van Gorp, P., Keller, A., Janssens, D.: Executing a standard compliant transformation model on a non-standard platform. Software Testing Verification and Validation Workshop, 2008. ICSTW '08. IEEE International Conference on (2008) 151–160
5. Varró, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. In Baar, T., Strohmeier, A., Moreira, A., Mellor, S., eds.: Proc. UML 2004: 7th International Conference on the Unified Modeling Language. Volume 3273 of LNCS., Lisbon, Portugal, Springer (2004) 290–304
6. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model transformations? transformation models! In: MoDELS2006. Volume 4199 of LNCS., Springer (2006) 440–453
7. Jouault, F.: Loosely coupled traceability for atl. In: Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, Nuremberg, Germany (2005)
8. de Miguel, M.A., Exertier, D., Salicki, S.: Specification of model transformations based on meta templates. In Bezivin, J., France, R., eds.: Workshop in Software Model Engineering. (2002)
9. Agrawal, A., Simon, G., Karsai, G.: Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. In: Electronic Notes in Theoretical Computer Science. Number 109, Barcelona, Spain (2004) 43–56