Model Transformation Design Patterns

Hüseyin Ergin
hergin@crimson.ua.edu

Department of Computer Science, University of Alabama, U.S.A.

Abstract. In this document, a brief overview of my doctoral research is presented. In model-driven engineering (MDE), most problems are solved using model transformation. An efficient process to solving these problems is to apply reusable patterns while solving them. Finding reusable design patterns to specific subsets of problems helps to decrease the time and cost needed to solve them. My doctoral research is based on finding these design patterns to be applied on model transformation problems and evaluating them in terms of quality criteria. My next step is to generate these design pattern instances automatically by using higher-order transformation.

1 Introduction

Model-driven engineering (MDE) [1] is considered a well-established software development approach that uses abstraction to bridge the gap between the problem and the software implementation. These abstractions are defined as models. Models are primary artifacts in MDE and are used to describe complex systems at multiple levels of abstraction, while capturing some of their essential properties. These levels of abstraction let domain experts describe and solve problems without depending on a specific platform or programming language.

Models are instances of a meta-model which defines the syntax of a modeling language. In MDE, the core development process consists of a series of transformations over models, called model transformation. Model transformations take models as input and output according to the specifications defined by the meta-model.

In this document, I summarize the subject that I want to work on in my dissertation. My dissertation will involve creating a formalism for new possible model transformation design patterns. With the help of the formalism, existing model transformation design patterns can be represent in a generic way. I will also identify more design patterns and evaluate them. In parallel my design pattern formalism will evolve as new ones are identified.

Rest of the document is organized as follows. Section 2 describes the problem. In Section 3, I give related work in this area. Section 4 introduces the solution I plan. In Section 5, I present current status and necessary steps to complete. Finally, Section 6 concludes the document.

2 Problem

MDE problems are solved using a series of model transformations. Model transformations have their own languages and they consist of two main components: *rules* and

scheduling. *Rules* are the smallest units of a model transformation and are used to transform one or more elements from source language to their intended equivalents in target language. *Scheduling* describes the order in which rules are executed. To develop a model transformation, developers design different rules and specify the scheduling of them. However, this design phase lacks reusability, which hampers the quality of model transformations. Therefore, there is a need for reusable, proven, and qualified structures in this phase. A design pattern encapsulates a proven solution to a recurring design problem [2]. As in the object-oriented world, design patterns help with the assessment of high quality model transformations. The first purpose of this doctoral research is to find design patterns that help developers to solve model transformation problems. In object-oriented design patterns, the community has agreed to provide design patterns in UML class diagrams. Due to the few works in the literature, there is no common language or standard for the model transformation field. Another purpose is to find the best formalism to express existing and newly identified model transformation design patterns.

3 Related Work

I have identified two studies in literature that introduce reusable structures in model transformation.

Agrawal *et al.* [3] used GReAT language to define three model transformation design patterns. This is the first structured design pattern study in the model transformation field. Each design pattern has *motivation*, *applicability*, *structure*, *known uses*, *limitation*, and *benefits* fields. They introduced the following design patterns: *the leaf collector*, which has a visitor pattern [2] like structure and aims to collect or process all leaf nodes in a hierarchy; *the transitive closure*, which can be used to compute the transitive closure of a graph; and *the proxy generator idiom*, which can be used in distributed systems where remote interactions to the system need to be abstracted and optimized.

Iacob *et al.* [4] used QVT Relations language to define five model transformation design patterns. Their design pattern structure has *name*, *goal*, *motivation*, *specification*, *example*, and *applicability* fields. They introduced the following design patterns: *the mapping pattern*, which establishes one-to-one relations between elements from the source model and elements from the target model and can be used to translate a model from one syntax to another; *the refinement pattern*, which obtains a more detailed target model by refining an edge or a node to multiple edges or nodes; *the node abstraction pattern*, which abstracts information from source nodes while keeping their relations and can be used to remove elements from models that hold certain criteria; *the duality pattern*, which generates a semantic dual of an instance model; and *the flattening pattern*, which removes the hierarchy from the source model.

These studies are excellent resources, but need to be analyzed, extended and improved. First, the design patterns are not analyzed in terms of quality. Agrawal *et al.* and Iacob *et al.* introduce design patterns as the core of their studies, but often do not mention how they affect quality in model transformation problems. Secondly, they often do

not provide a generic way of representing design patterns in terms of a design pattern formalism.

In this work, I extend these studies with a generic formalism, evaluation of proposed design patterns and identication of new design patterns.

4 Solution

In this dissertation, I focus on the solution to the reusability problem by working on design patterns. The initial step in this direction is to identify a design pattern formalism. The formalism is important in terms of providing the standardization of future design patterns and automically applying them to the transformations. The formalism will also provide an abstract representation of the solution independent from different model transformation languages. In this section, I provide some details about the solution in this direction.

4.1 Preliminary Work

In [5], I have identified a model transformation design pattern, called Fixed-point Iteration. This design pattern is applicable when the problem can be solved stepwise, and a single answer or a subset of the input model is returned as a solution. I have applied the design pattern to find the lowest common ancestor, to compute the equivalent resistance and to find the shortest path by using Dijkstra's shortest path algorithm. In this design pattern, I have used MoTif [6] as my model transformation language. MoTif has the modern modeling concepts such as explicit scheduling and rule-based model transformation. Thus, the formalism I used to represent the design pattern was MoTif-like, which will be improved to be more general.

A new formalism is needed to represent other design patterns that will be identified. For that reason, I designed the formalism depicted in Fig. 1. Each design pattern represents a subset of a regular model transformation solution and can be counted as a transformation itself. For that reason, the main element in the formalism is a *transformation*. The *transformation* has a list of *components* and *meta-models*. The *components* are *rules*, the smallest unit of a transformation, and *blocks* to support nesting and hierarchy. Each component is connected with an *output* class to another *component* to represent the control flow of the transformation language. Each *rule* has some *sections e.g.,* left-hand side (LHS) as pre-section, right-hand side (RHS) as post-section, when/where clause, etc. and *expressions*. *Expressions* represent the *actions* to be executed after a rule application and the *constraints* to satisfy before/during transformation. Each rule also has access to *variables* of the involved modeling languages, which is depicted with abstract *type* class. The variables include both the model elements and their attributes. This formalism assumes the meta-model of each modeling language has a *rootType* for the types they provide.

4.2 Expected Contributions

My contributions will be a list of design patterns for model transformation problems. This may lead to a bigger output than I expected, so I have filtered some of the categories

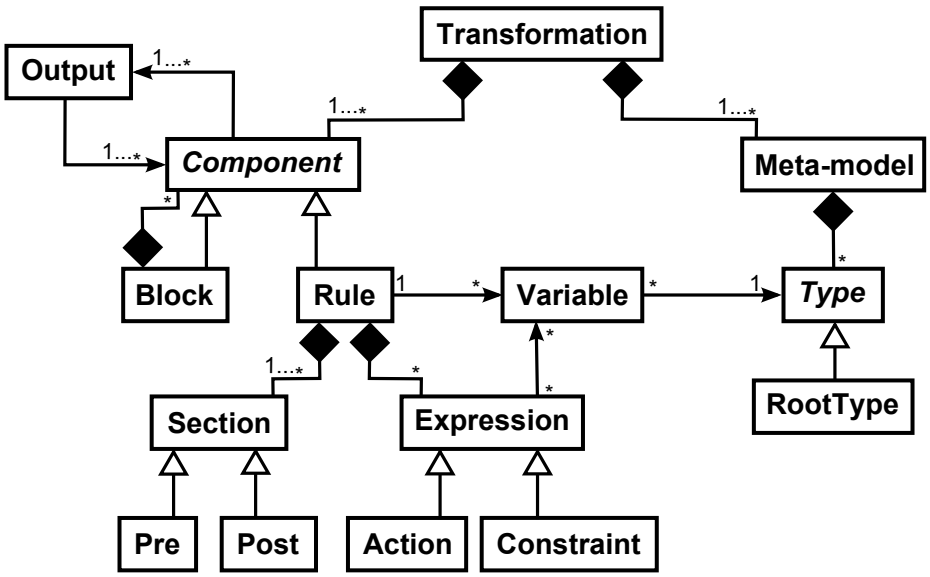

Fig. 1. Model transformation design pattern formalism

of design patterns by the model transformation intents. A model transformation intent is a description of the goal behind the model transformation and the reason for using it [7]. Classifying model transformation by intents is of paramount importance when working on design patterns. It helps target specific patterns for specific use cases and ensures they are useful in practice. Some of these intents are: manipulation, restrictive query, refinement, abstraction, translational semantics etc.

Each design pattern will have elements that are describing them. I have used *pattern name*, *problem*, *solution* and *structure* elements to describe a design pattern.

Also I am extending the features of our model transformation tool, AToMPM [8]. AToMPM allows one to model and execute model transformations. It provides a graphical user interface to define the meta-models of the intended formalisms, to describe rules graphically as well as control structure for model transformations, and to execute step-by-step transformations on given models.

4.3 Plan for Evaluation and Validation

I have analyzed the design pattern introduced in [5] in terms of some quality metrics. The analysis consists of the following metrics: size of rules, number of rule applications, and number of auxiliary elements. These metrics will be extended to support a more reliable and comprehensive evaluation and used in the following quality framework. In this doctoral research, I initially extended the analysis and created the quality framework in Fig. 2 by adapting Mohagheghi and Dehlen [9]'s study to MDE. The framework has the following steps:

1. Identify quality criteria, such as maintainability and reusability.

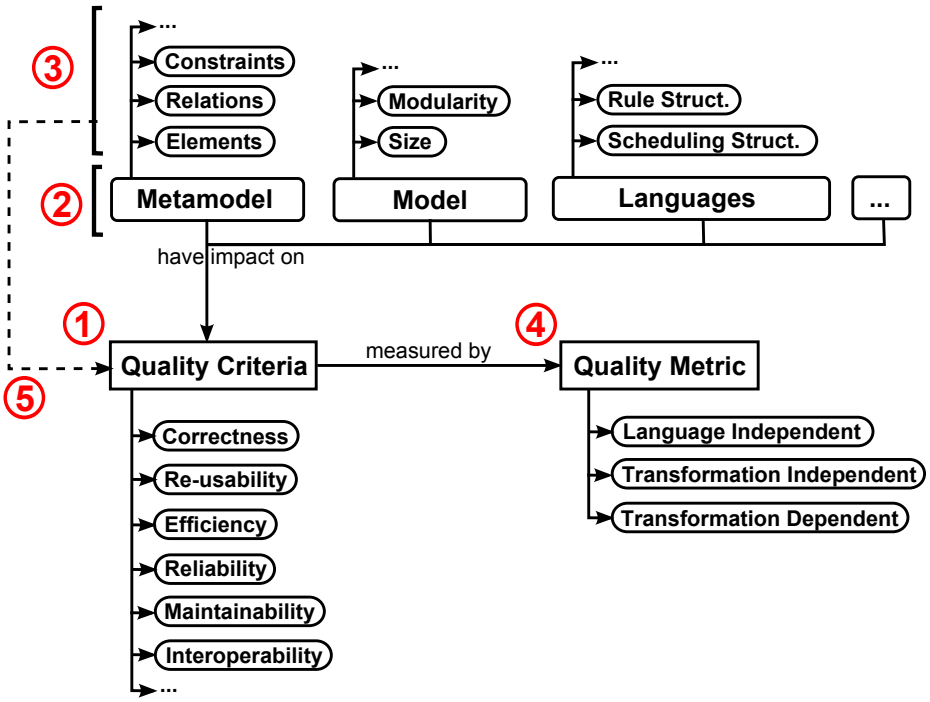

Fig. 2. Quality framework

2. Identify target objects that have an impact on quality criteria. These objects are meta-models, models, languages, transformations; *i.e.,* all concepts related to model transformation.
3. Identify the properties of target objects that have an impact on quality criteria. Following the example objects in the previous step, these properties can be elements, relations, and constraints for meta-models, size, and modularity for models, scheduling structure, and rule structure of languages.
4. Specify how to evaluate the quality properties. This includes the metrics to be measured quantitatively or subjective evaluation of the transformation. Other approaches may be empirical evaluation by interviewing the users or inspections using checklists.
5. Specify traceability links between quality properties and quality criteria and initiate the implementation of quality properties and evaluation of metrics. Evaluation of metrics together with the quality criteria in mind lets the designer which design pattern is good or bad for a specific purpose and intent.

Interviewing the users is an effective way to get feedback and identify the effect of some different technologies in empirical software engineering. This consists of preparing predefined experiments with scenarios and applying these experiments on the targeted users. In my case, I am planning to apply some design experiments, which will consist of finding solutions to some modeling problems, on different set of developers

in modeling community. The experiments may be supported with a questionnares and surveys at the beginning or at the end. The results of the experiments will lead to more improvements in the formalism and the design patterns I will identify.

5 Current Status & Plan

AToMPM is an online modeling and transformation tool [8]. Currently, it lets users design modeling languages, create instances of these languages as models and execute model transformations over these models. One of its unique features is letting users do all of these tasks graphically. The overall purpose of this dissertation is to extend AToMPM in a way that users will specify their model transformation solutions in terms of the design pattern formalism I provided in Section 4.1. The specification of the model transformation in the formalism enables users to be independent from any model transformation language.

The main steps are listed below:

- **The improvement of the design pattern formalism (Fall 2013):** Currently, the formalism is very similar to the transformation language of AToMPM: MoTif. The design pattern formalism will be refined to support other popular model transformation languages, beginning with ATL [10] and QVT [11]. Each model transformation language has its own unique characteristics, so the formalism should be as abstract as possible to satisfy them.
- **The identification of new design patterns (Fall 2013 - Spring 2014):** So far, we have identified one model transformation design pattern [5]. Design pattern identification requires finding the problems in the model transformation design process. Examples must be investigated carefully to find the recurring problems in existing model transformation problems. Another way to identify a design pattern is to focus on single problems and solve them efficiently by adopting different approaches, which we have applied to three different problems in [5]. While identifying new design patterns, categorization with respect to intents will help me to work on a subset.
- **Adapting the existing design patterns (Fall 2013):** The existing design patterns, which the authors call reusable patterns and idioms, will be adapted, analyzed and represented in the new formalism. This will help me to gather and utilize existing work.
- **Application and generation (Spring 2013 - Spring 2014):** AToMPM currently has MoTif and *T-Core* [6] as model transformation languages. In AToMPM, everything is modeled and can be manipulated using model transformations. Therefore, the solutions designed by users in the design pattern formalism can generate the results in any model transformation language. This can be realized by using the higher-order transformation (HOT) feature of AToMPM. By using HOT, users can not only generate their solutions in intended model transformation language, but also transform one language to another.

My dissertation touches a subject where the community does not have many studies. Therefore, an IDE with the ability of inserting design patterns automatically and generation of these pieces of transformations to a preferred model transformation language

to be a part of a larger solution is expected at the end. A list of newly identified design patterns will also be available as a part of this project.

6 Conclusion

In this document, I gave some details about the field I want to work on in my dissertation. I believe the design patterns will lead developers to increase their effectiveness and decrease the time and cost required to solve a model transformation problem. By using the design pattern catalog, related intents as categories and evaluation of each design pattern with respect to its purpose and metrics, developers will be able to reduce the amount of work they need. I also believe the formalism will help the model transformation community to represent the solutions independent from the specific model transformation languages. In this doctoral symposium, I hope to receive feedback on my approach and direction in order to improve the contents and my research focus.

References

1. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons (2006)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. 1 edn. Addison-Wesley Professional (November 1994)
3. Agrawal, A., Vizhanyo, A., Kalmar, Z., Shi, F., Narayanan, A., Karsai, G.: Reusable Idioms and Patterns in Graph Transformation Languages. In: International Workshop on Graph-Based Tools. Volume 127 of ENTCS., Rome, Elsevier (March 2005) 181–192
4. Iacob, M.E., Steen, M.W.A., Heerink, L.: Reusable Model Transformation Patterns. In: Proceedings of the Enterprise Distributed Object Computing Conference Workshops, Munich, IEEE Computer Society (September 2008) 1–10
5. Ergin, H., Syriani, E.: Identification and application of a model transformation design pattern. In: ACM Southeast Conference, Savannah, GA. (April 2013)
6. Syriani, E., Vangheluwe, H.: A Modular Timed Model Transformation Language. Journal on Software and Systems Modeling **11** (June 2011) 1–28
7. Amrani, M., Dingel, J., Lambers, L., Lucio, L., Salay, R., Selim, G., Syriani, E., Wimmer, M.: Towards a Model Transformation Intent Catalog. In: MoDELS workshop on Analysis of model Transformation, IEEE (2012)
8. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S.V., Ergin, H.: Atompm: A web-based modeling environment. In: MODELS'13 Demonstrations. (2013)
9. Mohagheghi, P., Dehlen, V.: Developing a Quality Framework for Model-Driven Engineering. In Giese, H., ed.: Models in Software Engineering. Volume 5002 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 275–286
10. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming **72**(1-2) (June 2008) 31–39
11. Object Management Group: Meta Object Facility 2.0 Query/View/Transformation Specification. (jan 2011)