A Unified Template for Model Transformation Design Patterns

Huseyin Ergin[1], Eugene Syriani[2]

[1] University of Alabama, USA
[2] University of Montreal, Canada
hergin@crimson.ua.edu, syriani@iro.umontreal.ca

1 Introduction

Design patterns are of tremendous value to developers when faced with recurring problems [1]. Given their various applications and uses, model transformations would benefit tremendously from design patterns as well. Although several studies have proposed design patterns for model transformation[2,3,4,5,6], there is still no accepted common language to express them.

In this paper, we propose a unified template to describe model transformation design patterns based on the analysis of existing model transformation design pattern studies. We have also initiated a unified template candidate that is suitable for the purpose, along with an actual model transformation design pattern adapted to the unified template.

2 Unified Template

2.1 Existing templates

Fig. 1 depicts the correspondences between existing proposals for model transformation design pattern templates and their equivalence with the template used in GoF [1]. Initial studies on model transformation design patterns proposed useful idioms that are specific to model transformation languages: ATL [4], VMTS [5], GReAT [2], and QVT-R [3] (corresponding to columns 2 to 5 in Fig. 1). These shall therefore be considered as implementations of design patterns in a specific language, rather than design pattern descriptions.

More recently, Lano et al. [6] provided the most comprehensive model transformation design patterns study (column 6 in Fig. 1). The template used by the authors leads to a thorough description of the patterns. In particular, the benefits and disadvantage fields are of great value to analyze and help to choose the appropriate pattern. The most critical is the solution field that explains the structure of the pattern in a language-neutral form. For this purpose, Lano et al. used an abstract transformation specification called TSPEC that manipulates an abstraction of the UML, called LMM. Although this provide a formalization of design patterns, the notation hinders the comprehension of design patterns, which does not guide developers in correctly implementing the pattern in a concrete transformation. At about the same time, we proposed a language, DelTa [7], that is independent from concrete transformation languages, understandable by developers, and enables the automatic instantiation and detection of the pattern in a concrete transformation.

Unified template	Bezivin	Levendovsky	Agrawal	Iacob	Lano	Ergin	GoF meaning
Summary	Motivation	Motivation	Motivation	Goal	Summary	Motivation	Intent
				Motivation			
Application condition		Applicability	Applicability	Applicability	Application conditions	Applicability	Applicability
Solution	Solution	Structure	Structure	Specification	Solution	Structure	Structure
Benefits	Consequences	Consequences	Benefits		Benefits		Consequences
Disadvantages			Limitations		Disadvantages		
Example		Known uses	Known uses	Example	Application and examples	Examples	Known uses
						Implementation	Sample code
Implementation						Variations	Implementation
Related patterns		Variations			Related patterns		Related patterns

Fig. 1. Comparison of fields for design pattern description

2.2 Proposed template

Existing works show that in model transformation area, there is not an agreement how to represent model transformation design patterns. Different studies have used different fields to represent a design pattern (i.e., applicability, benefits, structure). In addition, there is no common language of providing the structure of a model transformation design pattern, analogous to how UML is used in representing the structures of object-oriented design patterns. We therefore propose a unified template for expressing model transformation design patterns using the following fields:

- **Summary:** a short description of the design pattern that usually gives the outline of the other fields in a few sentences.
- **Application Conditions:** pre-conditions on the context of use of the pattern. The conditions can be either preconditions on the metamodel or constraints in the transformation overall. This is usually expressed in the same language as the solution field.
- **Solution:** generic solution to the problem the design pattern addresses. The structure of the solution is expressed DelTa [7].
- **Benefits:** advantages of applying the design pattern. The benefits can either be measurements with respect to some criteria or improvements on some features of the transformation.
- **Disadvantages:** pitfalls of applying the design patterns. The disadvantages are measured similarly to the benefits.
- **Examples:** example of applying the design pattern in a real context. The example is implemented in a specific model transformation language.
- **Implementation:** discussion providing guidelines and hints on how to implement the design pattern in various transformation languages.
- **Related patterns:** correlation of the pattern with other patterns. This relation may be specialization, generalization, sequence, grouping, alternatives, etc.

3 Top-down Phased Construction

We apply the unified template to the *Top-down Phased Construction* design pattern as introduced in [6]. We only present an abbreviated version of the pattern due to page limitation.

– **Summary:** This pattern separates the transformation into phases depending on how the target model is composed. The complete summary is found in [6].
– **Application conditions:** As depicted in Fig. 2, when there is a composition hierarchy in the source metamodel and the sub-element is mandatory in that relation.

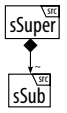

Fig. 2. Application condition

– **Solution:** As depicted in Fig. 3, the transformation is split into two phases. In the formerPhase rule, the target element corresponding to the super-element in the source is first created. In the latterPhase rule, target elements corresponding to the sub-elements in the source are then created.

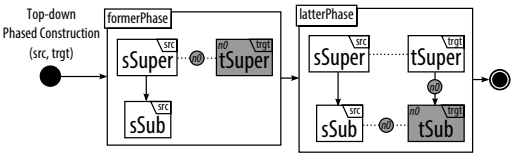

Fig. 3. Phased Construction - Structure in DelTa

– **Benefits:** This pattern increases the modularity of the rules, letting each rule create one layer of target elements. The complete list of benefits is found in [6].
– **Disadvantages:** Since the rules are broken into phases, that will increase the rule count. The complete list of disadvantages is found in [6].
– **Examples:** The UML class diagram to relational database diagrams can be considered a top-down phased construction, where we first use classes to create tables, and then use attributes to create columns [8].
– **Implementation:** The model transformation languages with explicit scheduling can create phases sequentially. The languages, that have implicit scheduling only, can refer to "simulating explicit rule scheduling" pattern in [6].
– **Related patterns:** The pattern resembles entity relationship mapping pattern [7], when the entities are considered the top layer and relations are the layer below. There are also variations of this pattern that starts the construction starting from bottom. Further related patterns are found in [6].

4 Conclusion

In this paper, we have summarized the structure of design patterns in terms of how they are represented in different studies. The need for a unified template approach is inevitable to make design pattern identification easier for future design pattern candidates. We believe using a UML-like graphical notation in at least application condition and structure fields of a design pattern makes the understanding and the application of the design pattern easier for the developer. The unified template should be discussed and improved to provide standardization in model transformation design patterns.

References

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Professional (nov 1994)
2. Agrawal, A.: Reusable Idioms and Patterns in Graph Transformation Languages. In: International Workshop on Graph-Based Tools. Volume 127 of ENTCS., Elsevier (2005) 181–192
3. Iacob, M.E., Steen, M.W.A., Heerink, L.: Reusable Model Transformation Patterns. In: EDOC Workshops, IEEE Computer Society (September 2008) 1–10
4. Bézivin, J., Jouault, F., Paliès, J.: Towards model transformation design patterns. In: Proceedings of the First European Workshop on Model Transformations (EWMT 2005). (2005)
5. Levendovszky, T., Lengyel, L., Meszaros, T.: Supporting Domain-specific Model Patterns with Metamodeling. Software & Systems Modeling **8**(4) (2009) 501–520
6. Lano, K., Kolahdouz Rahimi, S.: Model-Transformation Design Patterns. IEEE Transactions on Software Engineering **40**(12) (Dec 2014) 1224–1259
7. Ergin, H., Syriani, E.: Towards a Language for Graph-Based Model Transformation Design Patterns. In: Theory and Practice of Model Transformations. Volume 8568 of LNCS., York, Springer (jul 2014) 91–105
8. Jouault, F., Tisi, M.: Towards incremental execution of atl transformations. In Tratt, L., Gogolla, M., eds.: Theory and Practice of Model Transformations. Volume 6142 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 123–137