From Requirements to Design: Model-driven Transformation or Mapping?

Hermann Kaindl and Jürgen Falb

Institute of Computer Technology
Vienna University of Technology
A-1040 Vienna, Austria
{kaindl,falb}@ict.tuwien.ac.at

Abstract. In a case-based approach to software reuse, a requirements model may be used for indexing and retrieving other models and ideally all relevant software artifacts. In this context, the exact relationship between a requirements and a design model is of interest. We investigate this relationship in terms of model-driven approaches and discuss more precisely the question whether the transition from requirements to software design can be a model-based transformation or just a mapping. We show that it cannot be a model-driven transformation in the sense of such transformations from higher- to lower-level design models.

1 Introduction

In the EU-funded project ReDSeeDS, we pursue an approach to software reuse that builds on case-based reasoning. From a given (part of) a requirements model, cases should be retrievable that would contain solutions to similar problems, in terms of models and other software artifacts. In this context, we think it is important to raise the question, how exactly requirements and design models relate.

In this project, we investigate model-driven (MD) technology as well. In terms of this technology, this question can be stated more precisely as follows: Can the transition from requirements to software design be a model-driven transformation or just a mapping? So, we discuss in this paper, whether model-driven transformations are applicable for moving from requirements to software design.

The remainder of this paper is organized in the following manner. First, we set the stage by a brief discussion of related work. Then we explain our view of the differences between MD transformation and mapping. Finally, we show how to map from requirements to design.

2 Related Work

As shown in earlier work by Kaindl [1], (OO) domain models cannot be simply become (OO) design models. Neither is it possible to transform domain models to design models. Otherwise, there would be the implicit assumption that each

and every object class in the domain model would finally end up (after a metamorphosis) as several object classes in the detailed design and consequently the implementation. Larmen also states in [2] that domain models represent real-world concepts and not software objects and thus cannot be transformed automatically to a software design, but having mappings between domain and design classes lowers the representational gap between our mental model and the software.

Taking this together with Smialek's [3] approach to come from the notions mentioned in the scenario descriptions, everything mentioned in any scenario would end up in the implementation as such.

Even though automatic transformation seems not possible without intelligent problem solvers, establishing traces like in [4] and mappings between a domain and design model is useful for validation and case-based reuse. The mappings from requirements to design may be viewed as special and elaborate forms of traceability links. Some MD transformation approaches as classified in [5] support generation of additional traceability links, to trace model elements without requiring the mapping rules. Especially for reuse of software cases, traces to the applied mapping rules are necessary, barely dealt with in current approaches.

3 Differences between MD Transformation and Mapping

The view of MD transformations would suggest that a problem formulated in a requirements model could be simply solved by a few automated transformation rules. It is important to understand that this is not the case. In our attempt to explain this, let us discuss the differences between MD transformation and mapping.

MD transformations can be viewed as mathematical functions that constructively and uniquely map some input in one or more models to some output in one or more models. As such they are useful for systematic elaborations from higher- to lower-level design models and even to source code in some programming language. Ideally, these functions would even be computable (and the computations tractable).

More traditionally, software artifacts and models are related with each other for facilitating traceability. Traceability links can be viewed as mathematical relations, which associate artifacts of various kinds that belong together somehow. Traceability can and should be installed especially between requirements and design artifacts.

In terms of MD transformations, the transition from requirements to software design is more intricate, however, since it leads from a problem space to a solution space. In principle, methods from artificial intelligence might automate this kind of problem solving. As it stands, however, for most practical problems it involves many design decisions and judgment that cannot be readily formulated as MD transformation rules at today´s state of the art.

Once a software design is available as a solution to a problem formulated as requirements, artifacts in the problem space can be related to corresponding

ones in the solution space by traceability links. Even a functional mapping can be established in hindsight, as shown below, while this function is not computable for most practical problems.

4 Mapping from Requirements to Design

We show that it is possible to use MD rules to establish a relational mapping instead of using just simple traceability links. This mapping can be used in the sense of checking model consistency, but not for a transformation from problem space to solution space. That is, we cannot enforce consistency in the sense that the models are modified by MD rules.

In Figure 1 we illustrate a simple mapping between domain classes and design classes by using the graphical notation of QVT (Query/View/Transformations Language) [6] specified by the OMG [1]. This mapping relates model elements from the `<<domain>>` Class on both sides. The correspondence between the classes is based on their names, in which the name of the design class is suffixed with '`-D`' (as proposed in [1]).

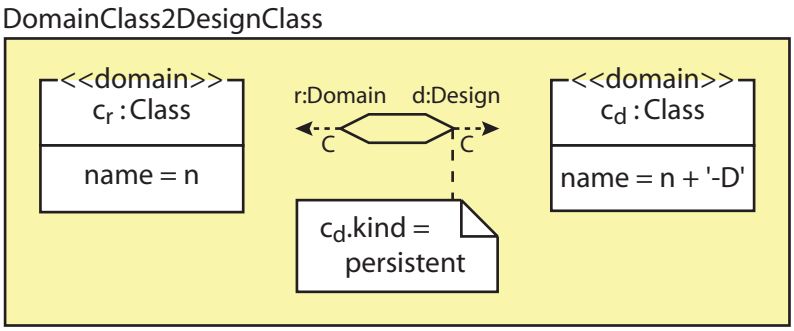

Fig. 1. Mapping of a domain class to a design class

The mapping relation specified by the hexagon in the middle of Figure 1 states that the checking (indicated by the '`C`'s) can be done in both directions, from the domain model to the design model and vice versa. In fact, these relations are even mathematical functions in both directions. However, they are only defined after the design work and can, in most practical cases, not be computed in advance.

This example further states, that the corresponding design class has to be persistent. The rationale is that only those domain classes lead to design classes, for which it is important to have a representation in the software, and these should be persistent. Note, however, that this is subject to design decisions based on judgment of the designer.

[1] OMG - Object Management Group, see `http://www.omg.org/`

The major problem of current mapping/transformation languages like QVT or ATL [7] is their use of universal quantification in the semantics of rules. Thus a check is only successful iff a rule evaluates true for all valid variable bindings. This reduces the reuse of more general mapping rules for the mapping of requirement models to design models, since only in rare cases a mapping of all requirements model elements matching a rule to design model elements is meaningful. Possible solutions for a higher reuse are the use of a large set of very specific mapping rules or the definition of a mapping language based on existential quantification.

5 Conclusion

In model-driven development, it is possible to establish a mapping between requirements and design, which can be employed for consistency checks. We have briefly sketched a related MD rule. Such rules should be used for checking consistency between requirements and design models, before they will be put into a case base for reuse. When a case will be retrieved later, this consistency should facilitate successful reuse. So, such a mapping as sketched in this paper is useful in the context of reusable software cases.

Whether the mapping resulted from the work of software designers with or without employing MD transformations, it would be useful in this sense. So, it may not even matter whether there is a transformation from requirements to software design or not. However, if such a transformation were available, the transition to design might be much easier and make reuse of software cases not cost effective. Anyway, this paper shows that the transition from requirements to software design cannot be a model-driven transformation in the sense of such transformations from higher- to lower-level design models.

References

1. Kaindl, H.: Difficulties in the transition from OO analysis to design. Software, IEEE **16**(5) (1999) 94–102
2. Larman, C.: Applying UML and Patterns. third edn. Prentice Hall PTR (2004)
3. Smialek, M., Bojarski, J., Nowakowski, W., Straszak, T.: Scenario Construction Tool Based on Extended UML Metamodel. In: Proc. of the 8th Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS'05). Volume 3713 of LNCS., Springer (2005) 414–429
4. Ebner, G., Kaindl, H.: Tracing all around in reengineering. Software, IEEE **19**(3) (2002) 70–77
5. Czarnecki, K., Helsen, S.: Classification of Model Transformation Approaches. In: Online Proc. of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA, Anaheim, USA (2003)
6. OMG: MOF QVT final adopted specification. http://www.omg.org/cgi-bin/doc?ptc/2005-11-01 (2005)
7. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Proc. of the Satellite Events at the MoDELS2005 Conference. Volume 3844 of LNCS., Springer (2006) 128–138