Specifying Model Transformations at the Metamodel Level

Sheena R. Judson[1], Robert B. France[2] and Doris L. Carver[1]

[1]Computer Science Department, Louisiana State University
Baton Rouge, Louisiana USA
judson@bit.csc.lsu.edu, carver@bit.csc.lsu.edu

[2]Computer Science Department, Colorado State University
Fort Collins, Colorado USA
france@cs.colostate.edu

Abstract

The MOF 2.0 Query/Views/Transformations (QVT) goal is to provide a standard for expressing model transformations. Techniques and technologies that support the rigorous definition and application of model transformations are required to realize the Model Driven Architecture (MDA) vision. In this paper, we describe an approach to rigorous modeling of pattern-based transformations. We discuss how our transformation specification approach promotes controlled pattern-based model evolution.

1 Introduction

Since the emergence of Model Driven Architecture (MDA), numerous techniques have been proposed for transforming models (see e.g., [1, 2]). MDA supports the development of software-intensive systems through the transformation of models to executable components and applications. The main motivation behind MDA is to transfer the focus of work from coding to solution modeling by treating models as the primary artifacts of development. As stated in [3], "MDA provides a set of guidelines for structuring specifications expressed as models and the mappings between those models". The mappings transform the elements of a source model that conforms to a particular metamodel into elements of another model, the target model that conforms to a metamodel [3].

In response to the need for a standard approach to defining the functions that map between metamodels, the Object Management Group (OMG) issued the MOF (Meta Object Facility) 2.0 Query/View/Transformation (QVT) Request for Proposals [3]. As stated in [4], "the principle requirement of QVT is to provide a standard … for expressing transformations". QVT requires that model transformations be defined

precisely in terms of the relationship between a source MOF metamodel and a target MOF metamodel [3]. In some situations the source and target metamodels may be the same metamodel.

Well-defined transformations that support rigorous model evolution, refinement, and code generation are key elements of MDA and QVT. In this paper, we describe an approach to rigorous modeling of pattern-based transformations at the metamodel level. The transformation specification technique can be used as a base for developing tools that support systematic application of reusable transformations.

The structure of the paper is as follows. Section 2 gives an overview of model refactoring with respect to model transformations. In Section 3, we describe our approach to metamodeling transformations. Finally, Section 4 provides a summary of the advantages of our approach and points out ongoing and future work.

2 Pattern-Based Model Transformation

Model transformation is the process of converting one model to another model [5]. In [6], France and Bieman categorize model transformations along *vertical* and *horizontal* dimensions. *Vertical transformations* occur when a source model is transformed into a target model at a different level of abstraction. Refining a model and realizing a model in a target programming language are examples of vertical transformations. In the context of MDA, vertical transformations are useful when transforming a platform-independent model (PIM) to a platform-specific model (PSM). A *horizontal transformation* involves transforming a source model into a target model that is at the same level of abstraction as the source model. Horizontal transformations are carried out to support model evolution. Two examples of model evolution are adding new features to a design and restructuring a design to enhance existing features. As is the case for code evolution, three types of model evolution can be distinguished:

- *Perfective evolution* is concerned with modifying a design so that it can better meet objectives.
- *Corrective evolution* is concerned with correcting errors in the design.
- *Adaptive evolution* is concerned with modifying a design model to accommodate changes in requirements and design constraints.

Of the three types of evolution identified above, perfective evolution is the one more likely to yield well-defined, reusable transformations - work on patterns (see e.g., [7, 8, 9]) and refactoring (see e.g., [1, 2, 10]) have yielded examples of such transformations. Capturing transformations in a form that is reusable can lead to the development of tools that support controlled evolution of models (with respect to the reusable transformations).

Our approach to model transformation focuses on transformations that support perfective model evolution. This type of transformation is referred to as *pattern-based model refactoring*. Ad-hoc approaches to model refactoring can lead to convoluted designs that are difficult to evolve and analyze. Controlled model refactoring can be accomplished by developing metamodels for the transformations. The metamodels can be used to determine how models are refactored and can be used as points against which the model transformations are checked for conformance.

A design pattern describes a family of solutions for a class of recurring design problems [9]. In pattern-based refactoring, a pattern is incorporated into a source design model to obtain a target model that contains an instantiation of the pattern. Pattern-based model refactoring is carried out when it is determined that a pattern can help improve how a design accomplishes its objectives. Figure 1 shows an example of model refactoring. In the example, a model in which a *Display* class is associated with a specific implementation class (*ImagImpl1)* is transformed using the Bridge pattern [9] to a design model in which the *Display* class is associated with a class structure that allows the image implementation to be varied.

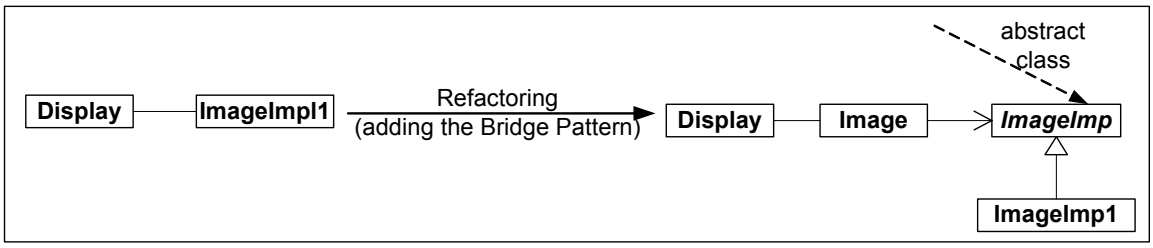

Figure 1. Model Refactoring.

The Bridge pattern transformation shown in Figure 1 is an example of a model-level transformation. A model-level transformation defines a relationship or mapping between a source model and a target model. In order to support controlled model-level transformation it is useful to have a specification of the transformation that can be used to determine valid and invalid forms of model-level pattern transformations. For example, a specification of the Bridge pattern transformation constrains how models can be transformed using the Bridge pattern – if a model-level transformation does not violate the specified constraints then one can be assured that the resulting target model has the properties specified in the Bridge pattern. The approach described in this paper uses metamodels as transformation specifications. A metamodel specification of a transformation is called a *transformation pattern* because it captures properties that are common to a family of model-level transformations.

The metamodeling approach described in this paper defines families of transformations in terms of classes of UML model elements that are created and deleted during transformations. The classes of model elements affected by specified transformations are treated as subclasses of UML metamodel classes. As an example,

consider the Bridge pattern shown in Figure 1. The source model can be characterized as a structure consisting of a client class (e.g., *Display*) associated with the implementation of a product class (e.g., *ImagImpl1*). This structure can be modeled in the UML metamodel by defining a subclass of *Class* whose instances are client classes, another subclass of *Class* whose instances are image implementation classes, and a subclass of *Association* whose instances represent associations between client and image implementation classes.

A specification of the transformation specifies the effect of characterized transformations in terms of a relationship between a source and target metamodel. The source metamodel specifies the UML (Unified Modeling Language [11]) models to which conforming transformations can be applied and the target metamodel specifies the UML models that are produced by conforming model-level transformations.

3 Metamodeling Transformations

The approach described in this paper allows one to define one or more specialized UML metamodels for pattern-based transformations, where each metamodel specialization characterizes a family of model-level transformations. This metamodeling approach defines families of transformations in terms of relationships between metamodels that specify source and target UML models.

An overview of the transformation approach is shown in Figure 2. The M2' level is an extension of the UML metamodel level (M2) that supports metamodeling of transformations. The M1' level is an extension of the UML model level (M1) that supports representation of model transformations. A model transformation, T1, at the M1' level takes a source UML model and transforms it to a target UML model. T1, which is denoted using a notation adopted from [12], is a member of the family of transformations characterized at the metamodel level by *Transformation Pattern*. A transformation pattern consists of characterizations of source and target models (*Source Pattern* and *Target Pattern*, respectively), and constraints on relationships between source and target elements. The *Source Pattern* is a metamodel that characterizes source UML models and the *Target Pattern* is a metamodel that characterizes target models for the family of transformations characterized by *Transformation Pattern*.

A transformation (e.g., T1) *conforms* to a transformation pattern if: (1) the source model (*Source Model*) is an instantiation of the source pattern (*Source Pattern*), (2) the target model (*Target Model*) is an instantiation of the target pattern (*Target Pattern*), and (3) the relationship between elements of source and target models satisfy the constraints specified by the transformation pattern. A UML model that conforms to a source or target pattern metamodel is said to be an *instance* of the source or target pattern. Similarly, a model transformation that conforms to a transformation pattern is said to be an instance of the transformation pattern.

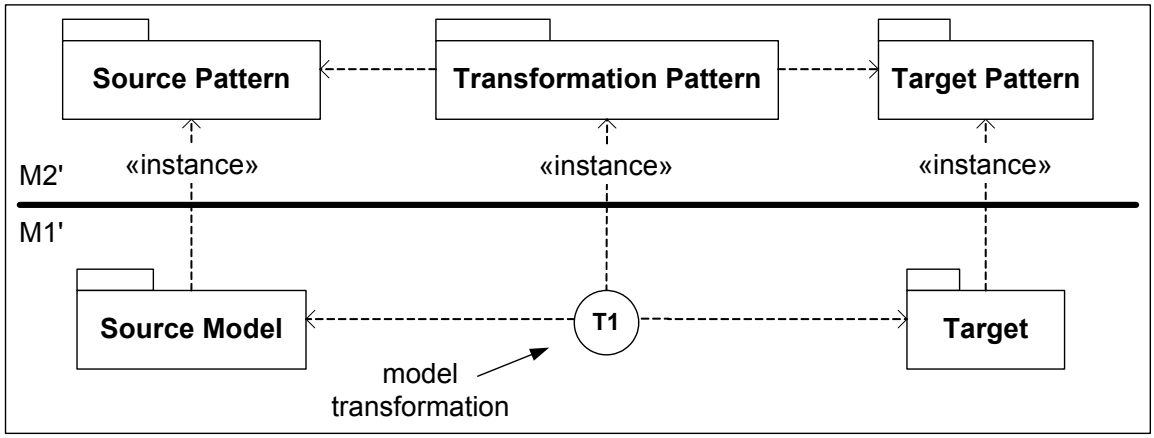

Figure 2. Transformation Overview.

Appendix A gives an annotated transformation pattern for a variant of the Abstract Factory pattern. A transformation pattern currently consists of three parts: *Source Pattern*, *Transformation Schema*, and *Transformation Constraint*.

The *source pattern* defines the set of source models to which transformations characterized by the transformation pattern can be applied. The pattern is expressed as a metamodel fragment that consists of classes characterizing model elements that are affected by the transformations. The source pattern consists of classes that are specializations (subclasses) of classes in the UML metamodel. The source pattern thus determines a specialized UML metamodel.

The *transformation schema* determines the structure of the target model. It shows the classes of model elements that are created by the transformations and the classes of source model elements that are removed by conforming transformations. The classes in a transformation schema are specializations of UML metamodel classes. The schema is expressed as a metamodel fragment in which the classes of new model elements are enclosed in dashed boxes and classes of deleted source model elements are marked with an X.

The *transformation constraint* determines the relationships that must hold between target and source model elements. It is expressed as object structures, where the objects are prototypical instances of classes in the source pattern and the transformation schema (i.e., the objects are prototypical representations of UML model elements). The object structures describe transformation constraints in terms of relationships that must hold between elements of the source and target models.

4 Conclusion

The approach to specifying model transformations involves specializing the UML metamodel to characterize source and target models. The transformation patterns can be used to constrain how transformations are defined at the model level and can act as points against which transformations are checked for conformance. Specifying the transformations as metamodels can lead to the development of tools that support controlled evolution of models. While other work focus on developing mechanism for carrying out transformations, this work focuses on how transformations can be specified at the metamodel level.

We have applied the approach to the Abstract Factory (AF) and to the Visitor design patterns. AF transformation patterns have been developed for UML class and interaction design models. We plan to apply our approach to other design patterns.

Future work includes the development of a technique for obtaining model level transformations from transformation patterns, and techniques for establishing conformance of transformations to transformation patterns. To support those techniques, we are developing a formal basis for the transformation specifications. We are also investigating how existing transformation mechanisms (e.g., the SMW (Software Modeling Workbench) [14] transformation mechanisms) can be incorporated into our approach.

5 References

[1] Akehurst, David and Stuart Kent. A Relational Approach to Defining Transformations in a Metamodel, in UML 2002 - The Unified Modeling Language: Model Engineering, Concepts, and Tools. Springer, October 2002.

[2] Song, E., R. B. France, D. K. Kim, and S. Ghosh. Using Roles for Pattern-based Model Refactoring, in Proceedings of the Workshop on Critical Systems Development with UML (CSDUML'02), 2002

[3] Object Management Group. Request for Proposal: MOF 2.0 Query / View / Transformations RFP. OMG 2002. http://www.omg.org/docs/ad/02-04-10.pdf.

[4] QVT Partners. QVT: The high level scope, QVT-Partners, 2003. http://qvtp.org/downloads/qvtscope.pdf.

[5] Object Management Group. MDA Guide Version 1.0.1. OMG, 2003. http://www.omg.org/docs/omg/03-06-01.pdf.

[6] France, Robert and James Bieman. Multi-view Software Evolution: A UML-based Framework for Evolving Object-Oriented Software. in Proceedings International Conference on Software Maintenance (ICSM 2001), November 2001.

[7] Eden, Ammon H., Joseph Gil, and Amiram Yehudai, "Precise Specification and Automatic Application of Design Patterns. in Proceedings of the 12th IEEE International Automated Software Engineering Conference (ASE 1997), November 1997.

[8] Cinnéide, Mel Ó. Automated Refactoring to Introduce Design Patterns. in Proceedings of the International Conference on Software Engineering, 2000.

[9] Gamma, E., Richard Helm, Ralph Johnson and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[10] Khriss, Ismaïl and Rudolf K. Keller. Transformations for Pattern-based Forward-Engineering. in Proceedings of the International Workshop on Software Transformation Systems (STS'99), May 1999.

[11] Object Management Group, UML 2.0 Superstructure Final Adopted Specification. OMG 2003. http://www.omg.org/cgi-bin/doc?ptc/2003-08-02.

[12] QVT Partners. Initial Submission for MOF 2.0 Query/View/Transformations RFP. QVT-Partners, 2003. http://qvtp.org/downloads/1.0/qvtpartners1.0.pdf.

[13] QVT Partners. The QVT Partners Manifesto. QVT Partners, 2003 http://www.qvtp.org/downloads/manifesto20030422.pdf.

[14] Porres, Ivan. A Toolkit for Manipulating UML Models, TUCS Technical Report No. 441, Turku Center for Computer Science, Åbo Akademi University, January 2002. http://www.tucs.fi/Research/Series/index.php.

Appendix A: An Annotated Abstract Factory Transformation Pattern