Transformations of UML 2 Models Using Concrete Syntax Patterns*

Markus Schmidt

Real-Time Systems Lab
Darmstadt University of Technology
D-64283 Darmstadt, Germany
markus.schmidt@es.tu-darmstadt.de

Abstract. Model transformations are an important part of the MDA approach. The process of converting a PIM into a PSM should be done by certain model transformations. While there are many transformation languages for UML available they all share the discrepancy between the syntax of the transformation specification language and the visual UML syntax. Today model transformations are defined either in a textual manner or in a language that uses constructs from the underlying metamodel. This paper presents a novel approach to specify model transformations as patterns in the concrete syntax of UML 2. These patterns are easier to read than usual transformation specifications and use only standard UML 2 constructs. This is achieved using the built-in extension mechanism of UML 2 - the Profiles. Besides the specification, these profiles offer the application of patterns within any UML 2 compliant modeling tool. As such, these patterns can be seen as a front-end for model transformation.

Keywords: Model Transformations, Patterns, UML 2 Profiles.

1 Introduction

Patterns and model transformations are closely related in model-driven software engineering. While patterns specify source and target model the execution of model transformations creates the target model. Transformations between models is a key activity within the MDA [1] framework. The importance of model transformation is stressed by Sendall and Kozaczynski [2] as they describe it as "The heart and soul of model-driven software development".

Such transformations can be defined to cross different modeling languages or to change models within the same language. Graph based transformations is the most popular way to express model transformations. This seems to be a natural choice since most modeling language are also graph based. An overview of graph transformation in the context of model-driven engineering is given by Grunske et.al [3]. They propose some requirements a good transformation language should

* Work supported in part by the European Community's Human Potential Programme under contract HPRN-CT-2002-00275, SegraVis.

N. Guelfi and D. Buchs (Eds.): RISE 2006, LNCS 4401, pp. 130–143, 2007.

hold. One of these requirements states that "transformation rules should be easy to understand". While it is necessary to use transformation rules that are understandable by themselves, it is also important that there is a certain kind of similarity between the transformation rules and the models to transform.

One common way to describe transformation rules for a modeling language (e.g. UML) is to use the abstract syntax of this language (e.g. metamodel of UML). One major drawback of this method is the discrepancy between abstract and concrete syntax. Transformation rules in abstract syntax can easily become complex and hard to read. For the case of UML the user must also know the metamodel. As UML is a very extensive language this is a very challenging task for most users. As an example of a transformation rule based on a metamodel we will present a simple transformation within a UML statemachine. Figure 1 shows the insertion of a transition **beta** between the state **A** and state **C**.

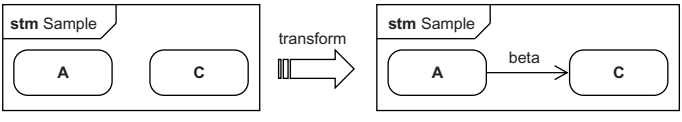

Fig. 1. informal description of a transformation that adds a transition

We use QVT [4] an an example of a transformation language that is based on a metamodel. The QVT rule for the transformation in figure 1 is shown in figure 2. The transformation rule itself is understandable but there is little relation between the statemachine and the transformation rule. Only a single transition must be inserted but the rule contains many more elements.

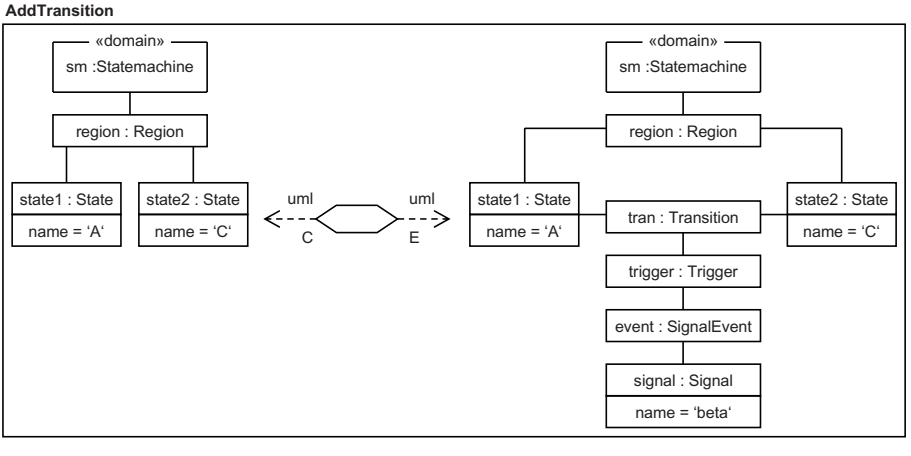

Fig. 2. QVT transformation to add a transition between two states