Generic Model Transformations:
Write Once, Reuse Everywhere

Jesús Sánchez Cuadrado\textsuperscript{1}, Esther Guerra\textsuperscript{2}, and Juan de Lara\textsuperscript{2}

\textsuperscript{1} Universidad de Murcia (Spain)
jesusc@um.es
\textsuperscript{2} Universidad Autónoma de Madrid (Spain)
\{Esther.Guerra,Juan.deLara\}@uam.es

Abstract. Model transformation is one of the core techniques in Model Driven Engineering. Many transformation languages exist nowadays, but few offer mechanisms directed to the reuse of whole transformations or transformation fragments in different contexts.

Taking inspiration from generic programming, in this paper we define model transformation \textit{templates}. These templates are defined over \textit{meta-model concepts} which later can be bound to specific meta-models. The binding mechanism is flexible as it permits mapping concepts and meta-models with certain kinds of structural heterogeneities. The approach is general and can be applied to any model transformation language. In this paper we report on its application to ATL.

1 Introduction

Model Driven Engineering (MDE) proposes the use of models as the key assets in the development, and hence all sorts of model modifications are needed. In this way, model transformations become one of the basic building blocks of MDE.

Even though MDE is being successfully applied in many scenarios, it still needs appropriate mechanisms to handle the development of complex, large-scale systems. One such mechanism is a facility to make transformations reusable, so that we can apply them to different contexts (i.e. with different meta-models). This would enable the creation of transformation patterns and idioms\textsuperscript{3}, as well as libraries of transformations addressing recurrent transformation problems. Some examples of manipulations commonly needed in different contexts are calculating the transitive closure of a relation, moving and merging nodes through a relation (like pulling up a method or an attribute), and cycle detection. Unfortunately, the definition of model transformations is normally a type-centric activity, in the sense that transformations are defined using types of specific meta-models, thus making their reuse for other meta-models difficult.

In this work, we bring into model transformation elements from generic programming in order to make model transformations reusable. In particular, we propose defining model transformation templates over concepts\textsuperscript{6,10,17}. In generic programming, a concept expresses the requirements for a type parameter of a template. In our case, a concept is a meta-model that defines the set...
of requirements that a specific meta-model must fulfill to be used in a transformation. Thus, when then concept is bound to a specific meta-model satisfying the concept requirements, the transformation becomes applicable to this meta-model.

In [6] we proposed concepts as a mechanism to add genericity to models, metamodels and in-place transformations. However, we only allowed a restricted kind of binding between the concepts and the meta-models consisting in an exact embedding of the former in the latter (i.e. no structural heterogeneity was allowed). In this paper we apply concepts to model-to-model transformations, and propose a more powerful notion of binding that permits replication of elements in the concept, as well as adaptations from the structure in the concept to the structure of the meta-model. Both types of variability induce modifications in the transformation template when instantiation takes place.

As a proof of concept, we report on an implementation on top of ATL [11] where the adaptation of the transformation templates is realized by a higher-order transformation (HOT). Nonetheless, our approach is general and therefore applicable to other transformation languages.

**Paper Organization.** Section 2 reviews the main elements of generic programming, and outlines our approach. Section 3 introduces transformation templates, concepts and bindings. Next, Section 4 adds flexibility to our approach by providing multiple cardinality for our concepts and adapters for the bindings. Section 5 outlines our exemplary implementation on top of ATL and Section 6 presents a case study. Section 7 compares with related work and Section 8 concludes.

# 2 Genericity in Model Transformation

Genericity is a programming paradigm found in many languages like C++, Haskell, Eiffel or Java [8]. Its goal is to express algorithms and data structures in a broadly adaptable, interoperable form that allows their direct reuse in software construction. It involves expressing algorithms with minimal assumptions about data abstractions, as well as generalizing concrete algorithms without losing efficiency. It promotes a paradigm shift from types to algorithms’ requirements, so that even unrelated types may fulfil those requirements, hence making algorithms more general and reusable [10] [17].

Genericity is realized through function or class templates in many programming languages, like C++ or Java. Templates declare a number of type parameters for a given code fragment, which later can be instantiated with concrete types. Templates can also define requirements on the type parameters, so that only those concrete types fulfilling the requirements are considered valid. A unit expressing a set of requirements is called a concept [10], and usually declares the signature of the operations a given type needs to support in order to be acceptable in a template. Hence, templates rely on concepts to declare the requirements of their type parameters.