Composing Transformation Operations Based on Complex Source Pattern Definitions

Arda Goknil[1], N. Yasemin Topaloglu[2]

Department of Computer Engineering, Ege University, Izmir, Turkey
[1]arda.goknil@ege.edu.tr, [2]yasemin.topaloglu@ege.edu.tr

Abstract. Rule composition and decomposition is a hot research topic within the context of model transformation. Mostly, transformation rules are considered as atomic parts of the transformation and rule composition has been the focus of recent research in the model transformation area. In our approach, we consider the transformation operations such as add, delete and update operations as the atomic parts of the transformation and the synthesis of these operations constitutes a single transformation rule. Defining complex and hierarchical source pattern definitions requires approaches and techniques about the composition and decomposition of these operations. In this paper, we discuss the problem statement and present an example case in which operation composition is required.

1 Introduction

Rule-based model transformation languages are the core technologies for operating the transformations between models on different abstraction levels in current *Model Driven Engineering (MDE)* approaches, such as *Model Driven Architecture (MDA)* [4] and *Model Integrated Computed (MIC)* [6]. In these languages, transformation rules are considered as the atomic elements of the transformation process.

Implementing large and complex transformations require complex and hierarchical pattern definitions to query models. In this context, complex pattern definitions mean that the pattern elements are tightly coupled and the relations between them are derived from the domain, not from the meta associations of the source metamodels. The coupling between the model elements is defined in the problem domain instead of metamodels. For instance, the relation between the *UML Class* and *UML Attribute* model elements in a proposed *UML2JAVA* transformation is derived from the UML metamodel. Complex source patterns include variation points and coupled elements hierarchically. Transformation rules which query these models should contain multiple operations and multiple pattern elements. In our approach, we consider the transformation operations such as add, delete and update operations as the atomic parts of the transformation and the collaboration between them constitutes a transformation rule.

In this paper, we highlight the need of transformation languages that support operation composition for complex pattern definitions. The paper is organized as follows. In Section 2, we discuss the transformations with operation definitions. In

Section 3, we present a sample transformation for composing transformation operations. Section 4 includes the conclusions.

2 Transformations with Operation Definitions

In transformation between two different metamodels, rules have simple definitions for transforming one model element in the source model into one or more model elements in the target model. This approach is called *one-many mapping*. In [7], the term *mapping* is defined as a synonym for correspondence between the elements of two metamodels, while the *mapping specification* precedes the *transformation definition*. Especially, transformation platforms which combine weaving and transformation, execute the transformations with one-many mapping. In transformations generated by mapping two different metamodels, the model elements are loosely coupled and the relations between them are derived from the source metamodel. The implicit rule calls defined in [1] solve the problem about integration and execution order of mapping rules. Composition approaches are mainly concerned about rule composition.

Since entities and the relations between these entities in the pattern definition are derived from the problem domain, they constitute the *hierarchical complex source patterns*. The rule structure requires complex pattern mapping within a single rule instead of one-many pattern element mapping within multiple rules. For instance, the relations between the pattern elements are defined by the problem domain in a proposed multiple inheritance-single inheritance (MI2SI) transformation [2] as a part of UML2JAVA transformation. In such a transformation operated by one-many mapping, the mechanism needs helper rules to define relations between the elements of the pattern. These helper rules make the problem definition more complex and incomprehensible. Transforming by pattern mapping takes the source pattern and transforms it into the target pattern by using less number of rules than one-many model element mapping uses. However, in this case, rules need multiple operations to transform one pattern into another one. Multiple operations in a transformation rule are given by Figure-1.

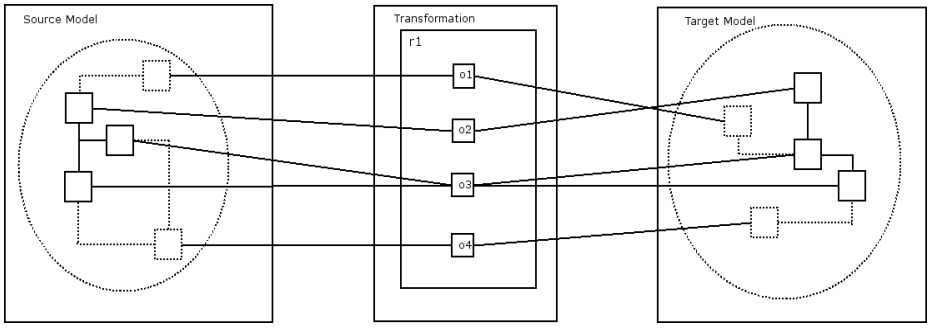

Figure-1. Multiple Operations in a Single Transformation Rule.

There are some issues to be considered in transforming by pattern mapping. Like rule composition, operation composition is required to manage and to organize the transformation operations defined between the complex pattern definitions. In Figure-1, the *r1* rule has four operations named *o1*, *o2*, *o3* and *o4*. The organization of the operations in the rule can be considered within the parallel of rule organization. The same problems about rule integration and organization occur in the operation organization and integration. There must be implicit operation calls in the rule to collaborate operations according to the relations between the pattern elements.

3 A Sample Transformation for Composing Transformation Operations

In this section, we explore the cases derived from the problem statements depicted in Section 2 over a sample pattern definition of multiple inheritance for a proposed multiple inheritance (MI) to single inheritance (SI) transformation. Although some programming languages include only single inheritance, defining a class by inheriting from more than one class is needed in a system design frequently. We consider UML models as the platform independent models which support multiple inheritance and Java programming language as our platform specific model which supports only single inheritance. The *MI2SI* transformation is a good example for complex pattern definitions because both source and target patterns contain multiple hierarchies and the transformation has multiple add, delete and update operations between MI and SI.

We consider the two alternatives as the representative cases of multiple inheritance (MI) since they can constitute the basis to generate other MI cases. In the first case, the inheritance hierarchy is composed of only one level and it is the simplest case of MI. The second case adds one more level to the inheritance hierarchy and called "diamond inheritance" [5].

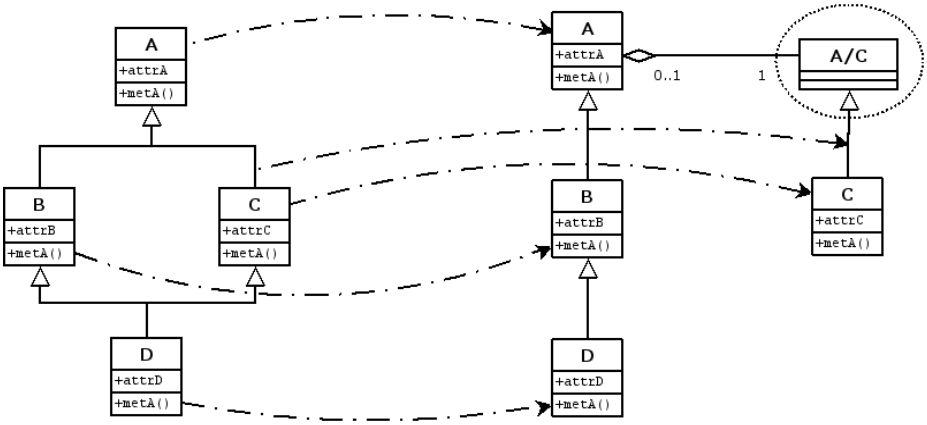

Figure-2. Transforming Multiple Inheritance to Single Inheritance with Role Aggregation.

In the hierarchy of multiple inheritance, we named the classes as *GrandParent*, *Parent* and *Child* classes. The *GrandParent* class is at the top of the hierarchy and the *Child* class is at the bottom. The *Parent* classes are the middle level classes which are the subclasses of the *GrandParent* class and super classes of the *Child* class. Figure-2 depicts the *MI2SI* transformation with role aggregation given in [2]. In Figure-2, one of the inheritance links is replaced by an aggregation link. The newly added abstract class named *A/C* is called *AbstractDiscriminatedClass* and the *Parent* classes in the *MI* which are the subclasses of the *A/C* class are now called the *ConcreteDiscriminatedClass* instead of *Parent* class.

The problem here is how to identify the rules between these two complex pattern definitions in the transformation definition. Decomposition satisfies the requirements for this identification. Kurtev [3] defined some rule decomposition approaches according to the target and source pattern. In source-driven approach [3], rules for every model element in the source pattern are defined. The rules for the *MI2SI* transformation are shown below:

GrandParentRule (**Source**[GrandParentClass],
Constraint[GrandParentClass, ParentClass, ChildClass],
Target[GrandParentJClass, AbstractJClass])

ParentRule (**Source**[ParentClass],
Constraint[ParentClass, GrandParentClass, ChildClass],
Target[ParentJClass, ConcreteDiscriminatedJClass])

ChildRule (**Source**[ChildClass],
Constraint[ChildClass, GrandParentClass, ParentClass],
Target[ChildJClass])

This transformation contains three rules. Each rule defines a source model element in its source part but each rule has the full definition of constraints to query the whole source pattern in the model. For instance, the *GrandParentClass* in the source part of *GrandParentRule* needs the full constraint definition of the source pattern to match in the model because the constraint part requires constraints of other source pattern elements related to the *GrandParentClass* to bind the appropriate model element. The helper rules are required in the constraint part to define the relationships between the pattern elements. In the *GrandParentRule*, we need to call two helper rules for the relation between the *GrandParentClass*, *ParentClass* and the *ChildClass*. The same helper rules and constraint repetitions are required for other rules named the *ParentRule* and the *ChildRule*. This kind of rule decomposition makes the definition more complex. We chose the composition of rules according to the source and target pattern mapping instead of rule decomposition.

MI2SIRule (**Source**[GrandParentClass, ParentClass, ChildClass, GPFeature],
Constraint[GrandParent, Parent, Child, GPFeature],
Target[GrandParentJClass, ParentJClass, ChildJClass,
ConcreteDiscriminatedJClass, AbstractDiscriminatedJClass,
GPJFeature, CDJFeature])

MI2SIRule maps the multiple inheritance source pattern and the single inheritance target pattern. The source and target parts of this rule include all pattern elements and the constraint part of the rules defines all relation and cardinality constraints of these pattern elements at once. In this kind of rule structure, we need to define the transformation operations which are the atomic parts of the transformation definition. There is a need of operation composition to organize and manage the appropriate operations within a single rule. Transformation languages should also support explicit definition of operation structures in rules. Every single operation should be able to map a number of source model elements to a number of target model elements.

MI2SIRule (**Source**[GrandParentClass, ParentClass, ChildClass],
Constraint[GrandParent, Parent, Child],
Target[GrandParentJClass, ParentJClass, ChildJClass,
ConcreteDiscriminatedJClass,AbstractDiscriminatedJClass]
Operation1[Type: Add, GPP_Generalization],
Operation2[Type: Delete, PC_Generalization],
Operation3[Type: Add, AbstractDiscriminatedJClass],
Operation4[Type: Add, GPA_Aggregation],
Operation5[Type: Add, AC_Generalization])

Another issue in the operation composition is the reuse of transformation definitions. *OMI2SIRule* depicts the transformation from one level multiple inheritance to single inheritance. Source pattern elements of the *MI2SIRule* except the *GrandParentClass* constitute the pattern elements of *OMI2SIRule*.

OMI2SIRule (**Source**[ParentClass, ChildClass],
Constraint[Parent, Child],
Target[ParentJClass, ChildJClass]
Operation1[Type: Delete, PC_Generalization],
Operation2[Type: Add, ParentFeaturetoChild])

We must decompose the constraint part of the rule structure to reuse the pattern elements and constraints of the *MI2SIRule* in the *OMI2SIRule*. The composed constraint structure of the *MI2SIRule* prevents the reuse of constraints for the *ParentClass* and *ChildClass* pattern elements. *MI2SIRule2* is the rule whose constraints are decomposed for every pattern element in the source part of *MI2SIRule*.

MI2SIRule2(**Source**[GrandParentClass, ParentClass, ChildClass],
ConstraintGP[GrandParent], **ConstraintP**[Parent],
ConstraintC[Child],
Target[GrandParentJClass, ParentJClass, ChildJClass,
ConcreteDiscriminatedJClass,AbstractDiscriminatedJClass]
Operation1[Type: Delete, GPP_Generalization],
Operation2[Type: Delete, PC_Generalization],
Operation3[Type: Add, AbstractDiscriminatedJClass],
Operation4[Type: Add, GPA_Aggregation],
Operation5[Type: Add, AC_Generalization])

As shown briefly in the above example, decomposing a transformation into operations and composing operations according to the complex pattern mapping make the transformation definitions more expressive. In addition, reusing the operations to constitute new rules is trivial.

4 Conclusion

In this paper, we discuss the need of operation composition and the general operation structure in transformation rules that transformation languages should support. Transformation languages should have additional features in their rule structures that provide operation composition. Composing operations within transformation rules will enable us to query and transform complex pattern definitions in a more expressive way.

References

1. Czarnecki, K., Helsen, S. Classification of Model Transformation Approaches. OOPSLA2003 Workshop on Generative Techniques in the Context of MDA, USA, 2003
2. Dao, M., Huchard, M., Libourel, T., Pons, A., Villerd, J.: Proposals for Multiple to Single Inheritance Transformation. In Proceedings of the 3rd International Workshop on Mechanisms for Specialization, Generalization and Inheritance MASPEGHI 2004 (Workshop ECOOP 2004), Oslo Norway, June 2004
3. Kurtev, I.: Adaptability of Model Transformations, PhD Thesis, University of Twente, 240p, ISBN 90-365-2184-X
4. OMG: MDA Guide Version 1.0.1. The Object Management Group, Document Number: omg/2003-06-01 (2003)
5. Sebesta, R.: Concepts of Programming Languages. Addison-Wesley Publishing, 2002.
6. Sztipanovits, J., Karsai, G.: Model-Integrated Computing. Computer, Apr. 1997,pp. 110-112
7. Lopes, D., Hammoudi, S., Bezivin, J., Jouault, F.: Mapping Specification in MDA: From Theory to Practice. INTEROP-ESA'2005