A Directive-Based Transformation Approach for UML Class Diagrams

Devon M. Simmonds
Department of Computer Science
University of North Carolina, Wilmington
Wilmington, North Carolina, 28403
simmondsd@uncw.edu

Abstract

In a model driven engineering (MDE) environment, developers create and evolve applications by creating models and transforming abstract models to more concrete models. Model transformation languages are needed to realize the benefits of MDE. In this paper we describe the Directive-Based Transformation Approach (DBTA), an imperative transformation approach for lightweight transformations on class models. DBTA utilizes abstractions that are specific to class diagram transformations and helps remove some of the accidental complexities associated with more general purpose transformation languages such as QVT. DBTA uses schemas consisting of transformation directives that can be processed by an interpreter. We present the overall design of DBTA and illustrate the new approach using Fowler’s extract super class refactoring transformation.

Keywords: Class diagrams, model transformations, transformation schemas, transformation directives, middleware, CORBA.

1. Introduction

A model transformation is a process that takes as input one or more source models and produces one or more target models [1 - 9]. The MOF 2.0 Query View Transformation (QVT) Language [16] is an Object Management Group (OMG) standard for specifying model transformations. Our experience specifying transformations [18 - 21] using QVT suggests that transformation languages need to work at a high level of abstraction in order to reduce the accidental complexity associated with specifying non-trivial transformations. QVT includes an operational mappings language, a core language and a relations language. The relations language defines a transformation as a set of relations between source models and target models, where a relation consists of a source domain pattern that describes valid source models and a target domain pattern that describes valid target models. Each pattern is an object diagram consisting of instances of metamodel classes. Several problems arise when models are described in terms of metamodel class instances. First, these specifications can produce large descriptions. Second, expressing transformations at this level of granularity can be tedious for medium to large-sized models. For example, a QVT source pattern that describes class models consisting of two classes with one attribute each, and one association between the classes, will contain instances for the classes, the attributes, the attribute types, the association and the association ends, that is, at least 9 model elements. Third, it is not easy to differentiate between types of model elements if they are all represented as instances of metamodel classes.

We propose the Directive-Based Transformation Approach (DBTA) [19], a lightweight graphical domain specific transformation approach for UML [22] class diagrams. DBTA provides a transformations specification syntax based on the concrete syntax of UML class diagrams rather than on the abstract syntax (i.e., the class diagram metamodel). Class diagrams are one of the most commonly used diagram types in object-oriented modeling and thus, one can expect that class diagram transformations will have wide applicability.

DBTA is designed to: (1) make transformation specifications more understandable to individuals familiar with UML class diagrams, (2) support the development of mechanisms for representing model transformations at a more abstract level than object diagrams, and (3) explore the use of directives in expressing model transformations. As a lightweight specification, DBTA is not intended to be a replacement for QVT. Instead, the intent of the research is to explore approaches and techniques that can enhance the development of model transformation standards such as QVT.

In this paper we describe the design of DBTA and illustrate use of the new approach. We provide a description of the transformation process and transformation directives in Section 2 and illustrate the use of DBTA for specifying the extract super class transformation [15] in section 3. Discussion and conclusions are presented in section 4.
2. The Directive-Based Approach

Figure 1 shows a class diagram representation of MDE transformation concepts in DBTA. The **Source Model** represents the model that is being transformed while the **Target Model** represents the transformed class model. The source model conforms to the **Source Metamodel** while the target model conforms to the **Target Metamodel**. The notion of conformance used in this paper is that established by Kim et al. [11]. In general, a model A conforms to a model B, if A faithfully reflects the structural and behavioral constraints and properties defined in B.

DBTA defines transformations graphically using a class model transformation specification called a **Transformation Schema**. Transformation schemas contain imperative statements called **transformation directives** that stipulate how target model elements are formed from source elements. Transformation directives are processed by an **Interpreter** to perform the specified transformations on class models.

![Figure 1. Model Transformation Concepts.](image)

2.1. Overview of Transformation Schemas

To reflect the class diagram syntax, a transformation schema is divided into three compartments. The first compartment contains the name of the transformation, formal arguments or parameters to the transformation, and any preconditions on the transformation. Figure 2 (a) shows the structure of a transformation schema. In the figure, the first compartment has the name **CopyClass(ModelName)**, where **CopyClass** is the name of the transformation and **ModelName** is the formal argument to the transformation. Formal arguments to a transformation represent the UML class models on which the transformation will be performed.

The second compartment of a transformation schema contains constructs called element schemas. Element schemas contain directives for creating target class model elements from source models elements. There are different types of element schemas corresponding to model elements of UML class diagrams, for example, class schemas, interface schemas, operation schemas and association schemas. Class schemas may contain attribute schemas and operation schemas. To accommodate this, class and interface schemas are also divided into compartments. A class schema is divided into a **Name Directive** compartment, an **Attribute Directive** compartment, and an **Operation Directive** compartment. Attribute schemas are specified in Attribute Directive compartments and operation schemas are specified on Operation Directive compartments.

For example, Figure 2 (b) shows an interface schema with the name **Resource**, containing the **NEW prepare()**:Vote and the **NEW commit()** operation schemas in its Operation Directive compartment and the **NEW resourceID:Integer** attribute schema in its Attribute Directive compartment. In these examples, the keyword **NEW** is an example of a transformation directive. The **NEW** directive is used to create new class model elements. We describe directives in section 2.3.

The third compartment of a transformation schema contains a list of transformations to be performed after the transformation defined in the first two compartments. In Figure 2, the **distributeClass** transformation will be executed after the **CopyClass** transformation has been executed. The use of transformation schema compartments allow the modularization of transformations and the specification of a transformation ‘program’ by listing modularized transformations in the second and third compartments of a transformation schema.

![Figure 2. Transformation Schema Elements](image)

2.2. Transformation Process

DBTA can be used to support the transformation process shown in Figure 3 [18, 20, 21]. The figure has two activity partitions. The **Development of Transformation** partition shows behavior associated with creating transformations and the **Application of Transformation** partition shows behavior associated with the use of transformations.

A complete DBTA class model transformation specification consists of a **Source Pattern** that describes valid source models and a transformation schema. The Source Pattern and transformation schema are created during the Develop Model Transformations activity. We describe model patterns using RBML templates, a variant of the Role Based...
Meta-modeling Language (RBML) [11 - 13]. RBML is a UML based language that supports rigorous specification of pattern solutions, where a pattern solution characterizes a family of solutions for a recurring design problem. RBML class diagram templates have template model elements that are explicitly marked using the “|” symbol (see Figure 4). Each template model element represents a model element in the source model. The template model elements in the class diagram template must be replaced by the source model elements they represent before a diagram template is used. In essence, we use RBML model patterns to specify the metamodel of source models.

Constraints on models may also be specified using a when statement. In the examples presented in this paper, transformation directives in the figures, are written using uppercase letters and the first and last compartments of the transformation schemas are not shown.

The source Directive: The source directive is used to select source model elements for inclusion in the target model. When the selected model element is to be modified, additional transformation directives are required. When the source directive is the only directive associated with a model element, the model element is copied to the target model without modification.

The use of the source directive is illustrated in Figure 4. In the figure, the transformation schema has the single source directive: source.|TransactionManager, where |TransactionManager is a source pattern model element that represents a source model class that manages a transaction (e.g., the 2-phase commit protocol). The meaning of this directive is that the class in the source model that |TransactionManager represents should be copied to the target model. This source model class is shown in Figure 4 (c). Before the transformation can be effected, the source model class is identified using the binding specification shown in Figure 4 (e). From this binding specification, |TransactionManager is bound to the AccountManager (the actual transaction manager class). This source directive results in the AccountManager class and its operations being copied to the target model. When a model element is copied, all its properties and any constraints associated with the model element are also copied. The source directive has the following forms.


Parent is a reference to a composite source pattern model element and SubElement is a reference to an operation,
operation template, attribute or attribute tem-plate, defined in the model element bound to Parent. RenameDirective is an optional rename directive, property is a keyword, and MetaAttribute is a meta-attribute of Parent or SubElement.

The rename Directive: Consider the scenario in which the AccountManager class from the previous example is being copied to an environment in which the name ServiceManager is used for transaction managers. The name of the copied class must be changed to reflect this environmental requirement. The rename directive is used to effect these kinds of transformations. The rename directive is used to provide a context-specific name for a model element. The rename directive has the form:

```
ModelElement {name = modelElementName}
```

where ModelElement is a reference to a model element in the source pattern and modelElementName is the context-specific name to be given to the source model element bound to ModelElement.

Figure 5. Modifying a class using the rename directive.

The rename directive is illustrated in Figure 5. The figure shows a class schema with the rename directive: \{name=AccountManager\}, attached to the source directive. For the source model shown in Figure 4 (c), the source.|TransactionManager directive results in the AccountManager class being copied to the target model, and the rename directive \{name=AccountManager\}, results in the name of the copied AccountManager class being changed to ServiceManager.

The exclude and new Directives: The exclude directive is used to exclude source model elements from inclusion in the target model. For example, if the AccountManager input class must be copied to the target model but its canCommit operation must be excluded, we can specify an exclude directive as shown in Figure 6 (a). The effect of the directive is to eliminate the canCommit operation from the target model as shown in 6 (d). The exclude directive may be applied to any model element. The exclude directive has two forms:

1. exclude ModelElement,
2. exclude

where the directive is associated with a transformation schema association or other transformation schema relationship (e.g., dependency and generalization) that is to be excluded from the target model.

The new directive is used to specify a new model element or a value for a meta-attribute. For example in Figure 6(a), a new operation is added to the target model using the directive, new getState():Integer. The new directive has three forms:

1. [new] ModelElement,
2. [new]
3. [new] Property.metaAttribute = newValue

The redefine Directive: Consider the scenario in which the AccountManager input class in Figure 6 is being copied to a Jini [10] middleware environment where the following requirements typically hold:

1) The name abort is used for the rollback operation.
2) The abort operation has an integer parameter but does not return a value.
3) The name prepare is used for canCommit.
4) The prepare operation has an integer parameter.

These requirements may be realized using the redefine directive. Using the redefine directive, modifications are specified using rename, new and exclude directives. In this example, the redefine directive:
redefine rollback (name=abort) (new id:Integer, exclude <params>):exclude Boolean

transforms the rollback operation by:

a) Changing the name of the operation to abort using the {name=abort} rename directive.
b) Adding a new integer parameter using the new id:Integer directive.
c) Causing all other parameters of the source model operation to be deleted using the exclude <params> directive.

Similarly, the directive:

redefine |canCommit {name=prepare} (new id:Integer, exclude <params>):Boolean

transforms the canCommit operation.

2.4. The when Statement
Preconditions are specified for transformations and directives using a when statement. Preconditions on transformations are described in the first compartment of a transformation schema after the name of the transformation. Preconditions on directives are specified immediately after a directive. The when statement has the forms:

(1) transformation-name WHEN expression
(2) directive WHEN expression

In this statement, transformation-name is the name of a transformation, directive is a transformation directive and expression is a logical expression. The when statement is illustrated in Figure 7.

3. Refactoring Example: Extract Superclass

Model refactoring is an important class of transformations in model driven engineering. In model refactoring, the structure of a model is transformed without changing the behavioral properties of the model [15]. Models are typically refactored to improved one or more model properties. For example, a model may be refactored to enhance reusability of specific model components or to make a model amenable to distribution.

This section illustrates the representation of the extract superclass model refactoring transformation described by Fowler [15]. In the extract superclass transformation, class operations with the same (or similar) operation signature are extracted from source classes and used to form a super class.

A transformation schema that describes the extract superclass model transformation is illustrated in Figure 7. The precondition on the transformation is that an operation with the same signature must exist in two different classes. In such cases, a new super class is created and populated with matching operations as well as matching class attributes. Preconditions are specified for transformations and directives using a when statement. Preconditions on transformations are described in the first compartment of a transformation schema after the name of the transformation. Preconditions on directives are specified immediately after a directive.

A source model for the transformation is shown in Figure 8 (a). The source model has four classes, Employee, Manager, Company, and Customer. The execution of the transformation proceeds by binding one of the four classes to ClassA, and then binding the other three classes to ClassB, one at a time to determine if the precondition may be satisfied.
in common with the Employee class. However, when the Manager class is bound to |ClassB, then |ClassA.operation in the precondition match the promote and updateSalary operations in the Employee class and |ClassB.operation in the precondition match the promote and updateSalary operations in the Manager class. Since the precondition is satisfied, the transformation is executed and a new super class is created with copies of these two operations. In addition, the name:String attribute is common to both Employee and Manager, so this attribute is copied to the super class as well. The resulting target model is shown in Figure 8 (b).

The current DBTA specification allows for the addition of other directives should it become necessary. The primary limitation of the new approach is that the approach is class-model specific at this point, and does not support transformations involving other diagram types. We plan on extending DBTA to other UML diagrams (e.g., sequence diagrams and activity diagrams). Part of the strength of the QVT approach is that the level of detail at which transformations are specified allows for the precise specification of many small details. Further research is needed to determine the expressive limitations of using more abstract notations.

As part of our future work, we intend to build a tool to support the DBTA model transformation process. We would also like to explore the use of target patterns, where each target pattern describes the minimum set of properties expected of valid target models. In particular, we are interested in exploring the automated generation of target patterns from a source pattern and a transformation schema.

Acknowledgements

I would like to thank Dr. Sudipto Ghosh and the late Dr. Robert France who guided me through my PhD dissertation and with whom these ideas were first articulated.

References


[2] Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. 2013. A research roadmap towards achieving schemas that contain imperative statements called directives. The use of the concrete syntax of model elements is an important feature of domain-specific model specification and transformation. The abstract syntax provides for the precise characterization of minute details of a model at the expense of readability and comprehension. The concrete syntax provides for a more abstract representation and the primary expenses is details. Since the concrete syntax of any UML model element specified using the abstract syntax can be specified in DBTA, we believe the use of the concrete syntax provides a viable alternative to the use of abstract syntax in the specification of model transformations. This is akin to developing programs in assembly versus development in a high level language.
scalability in model driven engineering. In Proceedings of the Workshop on Scalability in Model Driven Engineering (BigMDE ’13), Davide Di Ruscio, Dimitris S. Kolovos, and Nicholas Matragkas (Eds.). ACM, New York, NY, USA, 10 pages. URL: http://0-doi.acm.org.libcat.uncw.edu/10.1145/2487766.2487768


