Supporting Model-to-Model Transformations: The VMT Approach[1]

Shane Sendall[†], Gilles Perrouin[❖], Nicolas Guelfi[❖], Olivier Biberstein[+]

[†]Swiss Federal Institute of Technology in Lausanne
Software Engineering Laboratory
CH-1015 Lausanne EPFL, Switzerland[2]
sendall@acm.org

[❖]Luxembourg University of Applied Sciences
SE2C-Software Engineering Competence Center
6, Rue Coudenhove-Kalergi
L-1359 Luxembourg-Kirchberg, Luxembourg
{gilles.perrouin, nicolas.guelfi}@ist.lu

[+]Berner Fachhochschule - University of Applied Sciences
Biel School of Engineering and Architecture
Quellgasse 21, CH-2501 Biel, Switzerland
Olivier.Biberstein@hta-bi.bfh.ch

Abstract

The model-driven architecture approach (MDA) promotes software development as driven by a thorough modeling phase where application code is automatically generated from a platform specific UML model (PSM). The idea is that the PSM is itself derived from a platform independent UML model (PIM). Such code generation and model derivation are examples of model-to-model transformations.

This paper presents the Visual Model Transformation (VMT) approach, which includes a transformation language and a tool to support UML model transformations. The transformation language is a visual declarative language that supports the specification, composition and reuse of model transformation rules. These rules make use of the OCL language and a visual notation to indicate the selection, creation, modification and removal of model elements. An abstract denotational semantics based on graph transformation is sketched for the VMT language. We also present the MEDAL tool, which is a prototype build on top of IBM/Rational XDE development environment, and is a first step towards tool support for the VMT approach.

1 Introduction

More so than ever, software is hard to construct and maintain. One of the main factors that make it difficult is the size and complexity of the problem to be addressed. Abstraction and Separation of Concerns offer some of the best tools to combat problem size and complexity in software design. In particular, the use of software models has become a popular way to harness the principles of abstraction and separation of concerns.

In this context, the Unified Modeling Language (UML) [Omg03a], which became an Object Management Group (OMG) standard in 1997, is used by the majority of software modeling techniques and approaches. UML can support many different kinds of abstractions and separation of concerns. UML is a rich language that can be used to develop a set of inter-related models. The number and complexity of such models can vary depending on the abstraction level and kind of view taken, where the overall model is defined as the composition of these models.

There are a number of model engineering techniques that can be applied during software development, which include: refinement and derivation of models toward a software realization,

[1] This work is supported by the Luxembourg Ministry of Higher Education and Research under the Title I project n°MEN/IST/01/001

[2] From August 2003, Shane's address will be: Software Modeling and Verification Lab., University of Geneva, CH-1211 Geneva 4, Switzerland.

reverse engineering models to a higher level of abstraction, generation of models that act as views of existing models, and the synchronization of models.

These techniques are examples of *Model Transformations*. A model transformation involves taking one or more models as input and producing one or more models as output according to a set of rules specific to the purpose in hand. Performing these model transformations by hand can be quite a cumbersome and error-prone task. Ideally, such tasks should be automated in order to improve developer productivity and reduce human error.

Our work on model transformation is performed in the context of the FIDJI project [FIDJI] of the Luxembourg University of Applied Sciences in collaboration with the Swiss Federal Institute of Technology in Lausanne and the University of Applied Sciences in Biel, Switzerland.

The FIDJI team has experimented with concrete model transformations while developing architectural frameworks [GS02, GR03]. Theoretical issues addressed by the FIDJI project [BG00, DiM99, Gue01] and more practical needs of our architectural frameworks led the team to define a systematic approach to perform generic model transformations. Tool support for this approach was prototyped in a tool called MEDAL (uMl gEneric moDel trAnsformer tooL), which is an add-in to IBM/Rational's XDE UML modeling tool [XDE].

Owing to our experiences with MEDAL and our wider vision for model-based software development, we are working on an approach that is capable of transforming any set of UML models to any set of UML models. Our goal is to provide an approach that offers a means to specify and execute any UML-to-UML model transformation that could be useful during the model engineering activities of software development.

Our proposal for achieving this objective is called the Visual Model Transformation (VMT) approach. It offers a visual and declarative language for specifying UML model transformations.

This paper is organized the following way: Section 2 describes the first prototype developed and highlights the lessons learned; it sets the stage for the description of the VMT approach, which is given in Section 3. A formalization of the VMT approach using Graph Transformation Theory is proposed in Section 4. Section 5 presents related work in the field of model transformations. Finally, Section 6 concludes and presents future work.

2 MEDAL – A First Cut at Tool Support for Model Transformation

2.1 Context

Our interest in model transformation began with our work developing architectural frameworks using IBM/Rational's XDE UML modeling tool. The framework, which we call JAFAR [GR03], makes use of transformations to refine high-level models into J2EE artifacts. These transformations are defined in XDE and executed by its pattern engine. In XDE, a transformation is treated as a UML collaboration (also called "Model Template"), which has a set of template parameters. Using a scriptlet language, it is possible to perform actions on these parameters. It is also possible to define pre and post conditions to model template applications. For more sophisticated transformations, one can attach a Java program that realizes the actual transformation (referred to as a "callout").

XDE 1.5, which will be released officially in July 2003, will add support for OCL and UML profiles. These two features are particularly relevant to model transformations. OCL is tailored to operate with UML and one of its purposes (together with the specification of model constraints) is the navigation through UML models and selection of subsets of them. UML profiles [Omg03b] are extension mechanisms that allow the specialization of the UML metamodel to fit particular business domains or platforms. Using an UML profile, one can provide to the modeler a set of predefined model elements via stereotypes and tagged-values. Together with a set of rules, UML profiles can be employed to define mappings into particular platforms (J2EE, CORBA…). So far, many profiles have been defined [Omg03b], which each target a specific platform or domain.

Kozaczynski and Thario showed in [KT02] how to make use of OCL and UML profiles in XDE to automatically transform RUP user-experience models into artifacts for the Struts framework [Struts03]. Starting from that experiment, we investigated how XDE 1.5 could help one to specify and perform more generic model transformations. The result of this initial work was a first prototype, which we named MEDAL [GS03] (uMl gEneric moDel trAnsformer tooL).

The objectives of this prototyping phase were to define a Domain Specific Visual Language (DSVL) dedicated to UML Model Transformations, which provides a means to define and compose model transformations in UML.

2.2 MEDAL Features

We designed the first release of the tool to act as a visual "wrapper" to the existing XDE's transformation mechanism. One can distinguish two kinds of tasks for specifying a transformation process: the definition of individual transformations and the definition of the sequencing between transformations.

To define a single transformation, we use a class diagram in which a model template is seen as a package and parameters are shown as stereotyped OCL notes. To enhance the expressiveness of notes we have defined a small extension to OCL called MxOCL (MEDAL's extended OCL) allowing user aliases to be entered inside OCL expressions. The same UML notes mechanism is employed to specify the source context of the transformation and the target location.

To compose transformations, we use UML activity diagrams. For this purpose, an activity in an activity diagram corresponds to a transformation and a transition between activities corresponds to a sequential control flow between transformations. In particular, we used a subset of the control flow constructs offered by UML activity diagrams (i.e., decisions, transitions, guard conditions) as a means to compose transformations. The combination of these diagrams forms the MEDAL transformation language. A sample overview of the different views of this language can be found in Figure 1. This figure shows a simple transformation that adds accessor and mutator methods to all classes that are contained in a given package. These methods are added for all the attributes on the classes, and the attributes are made private, if they are not already.

The meta-model of the MEDAL language as well as its well-formedness rules (expressed as OCL constraints) are described in an UML profile. This profile is evaluated by the tool to check diagrams and any errors are reported by the tool to the user. This language is out of the scope of this paper. Interested readers are referred to [GS03, SRP03].

2.3 Lessons Learned — Moving Towards VMT

Activity diagrams ("Transformation Sequencing Rules" see Figure 1) have revealed to be a good representation for composing transformations. However, notations for loops and other more complex control structures have to be studied further. Concerning the use of "Model Template Application Definition" diagrams (see Figure 1), the main issue is to cope with a huge number of parameters or accesses to elements. As the number of MxOCL notes and parameters can be high, using notes to specify a great number of parameters can become cumbersome. We must also consider the point that in case of obfuscated access to some model elements, the MxOCL expression included in a note can be very hard to read.

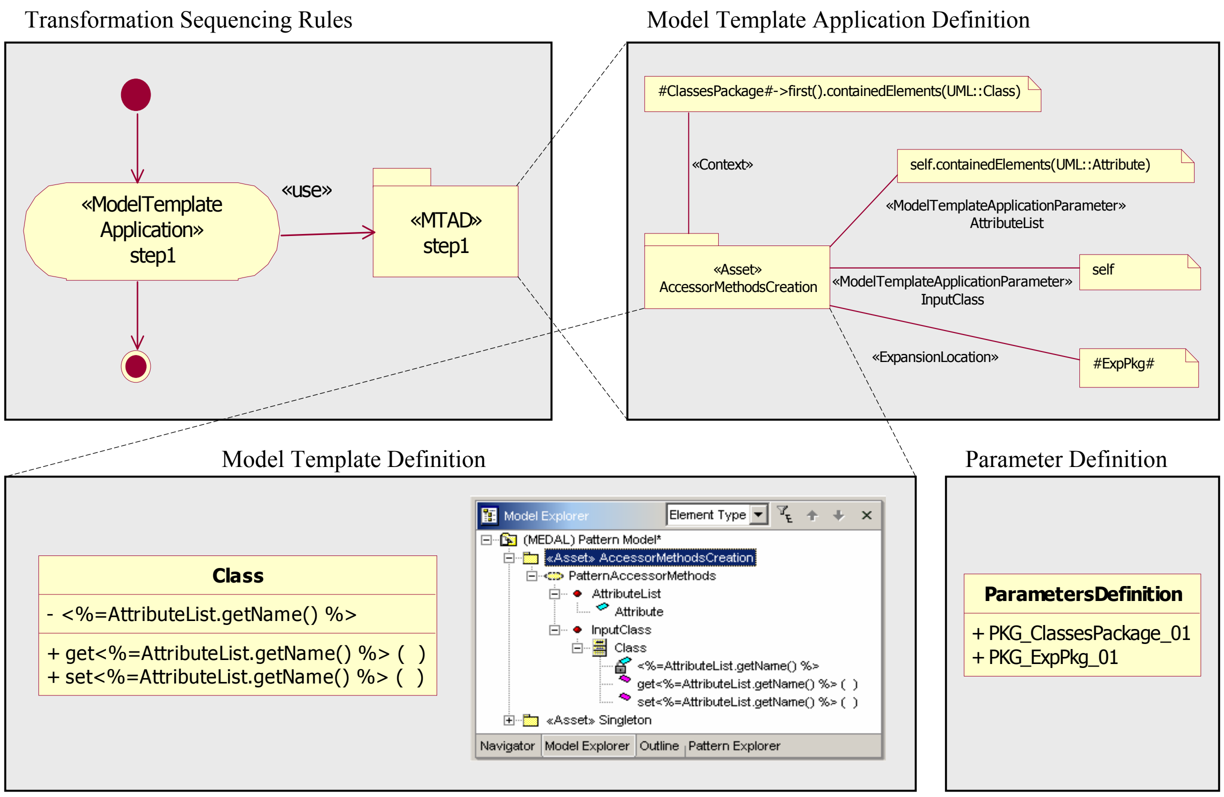

Transformation Sequencing Rules
«ModelTemplate Application» step1
«use»
«MTAD» step1
Model Template Application Definition
#ClassesPackage#->first().containedElements(UML::Class)
«Context»
self.containedElements(UML::Attribute)
«ModelTemplateApplicationParameter» AttributeList
«Asset» AccessorMethodsCreation
«ModelTemplateApplicationParameter» InputClass
self
«ExpansionLocation»
#ExpPkg#
Model Template Definition
Class
- <%=AttributeList.getName() %>
+ get<%=AttributeList.getName() %> ( )
+ set<%=AttributeList.getName() %> ( )
Model Explorer
Element Type
(MEDAL) Pattern Model*
«Asset» AccessorMethodsCreation
PatternAccessorMethods
AttributeList
Attribute
InputClass
Class
<%=AttributeList.getName() %>
get<%=AttributeList.getName() %> ( )
set<%=AttributeList.getName() %> ( )
«Asset» Singleton
Navigator
Model Explorer
Outline
Pattern Explorer
Parameter Definition
ParametersDefinition
+ PKG_ClassesPackage_01
+ PKG_ExpPkg_01

Figure 1- MEDAL Language Overview

3 The VMT Approach

In this section, we present the VMT approach, which includes our proposal for a UML model transformation language. We give an overview of VMT and describe the various components of the transformation language.

3.1 Approach Overview and Background

The VMT approach proposes a principally visual language for describing transformations between models specified with UML. In the language, a transformation is defined in terms of a set of *transformation rules*. Each transformation rule defines the way that one or more target UML diagram elements are created, changed, and/or deleted as a function of zero or more source diagram elements. Intuitively, this process can be seen as a mapping from source to target diagram elements, where target elements may be created in the process. A transformation rule is described by a *rule specification*. A rule specification consists of two parts: a *matching schema* and a *result schema*. The matching and result schemas are inspired from the work on graph transformation approaches (see Section 4).

The matching schema of a rule specification defines the condition under which the rule has permission to fire, the input arguments for the transformation, and those input arguments that will be deleted with the execution of the rule. A matching schema is represented as a graph, which has two roles: 1) it is used to define the condition that must be fulfilled for the rule to be permitted to fire, and 2) it is used to define the binding relation between source model elements and input arguments of the transformation rule. Intuitively, nodes in the matching graph can be seen as placeholders for elements in the source model. It is also possible to have a node that is a placeholder for a set of elements. Equally, it is possible to define a prohibited element, which is represented also as a node. This concept is used to strengthen the firing condition for the rule. In particular, if a match is found for a prohibited node, then the rule no longer has permission to fire.

A result schema defines the target diagram(s) as a function of the elements bound by the matching schema. A result schema is also represented as a graph. Intuitively, a node in the result graph represents an element of a target diagram. In particular, a node can represent a newly created element, a modified element from the source model, or an unmodified element from the source model. Like for the matching schema, it is also possible to have a node that represents a set of target model elements.

To illustrate the role of the matching and result schemas in a rule specification, suppose that we would like to transform an association class with certain multiplicities, and between two classes, into a class with binary associations (with the corresponding multiplicities). Figure 2 illustrates the transformation. It represents the transformation in terms of the before (left) and after (right) state, in the same vein that we would describe the matching schema and result schema, respectively, of the rule specification for this transformation.

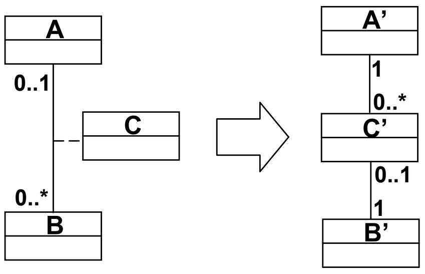


Figure 2 – Graphical Depiction of an Example Transformation

A model transformation can modify existing models or it can create new models (or a combination of both); we refer to these two categories of transformation as modification and creation transformations, respectively. A modification transformation involves changing an already existing model, or set of models, through the addition, modification and removal of elements. For example, a modification transformation could define the application of a design idiom to a UML class diagram. In this case, the target diagram, i.e., the output class diagram, is a slightly modified version of the source diagram. The example cited in Figure 2 is also an example of a modification transformation. A creation transformation involves creating only new elements as a result of the transformation. For example, a creation transformation could define the creation of a (randomly generated) end-to-end scenario, represented by a UML sequence diagram, from a UML state diagram. In this case, the target diagram, i.e., the resulting sequence diagram, only includes the new elements, which can be traced back to elements in the state diagram.

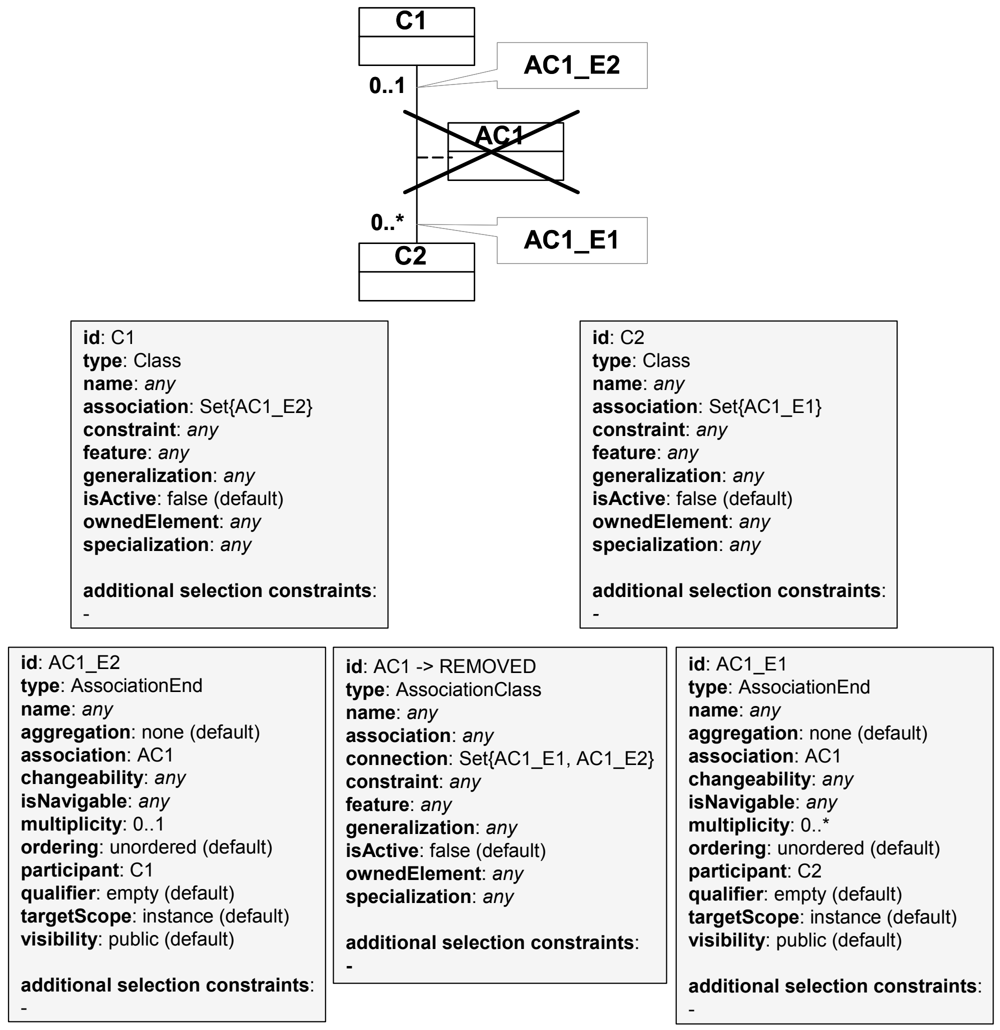
When defining a transformation, it can sometimes be easier to formulate the description in terms of a set of simpler transformations instead of a single more complex transformation. In our approach, this technique is made possible by defining each of the simpler transformations as a distinct rule and defining the required ordering of rule application using a separate description, which we call a *rule ordering schema*. We find it useful to have these two levels of description as we have observed that it promotes scalability and reuse, and it promotes a simpler rule specification language, due to the separation of concerns. In addition to rule sequencing, a rule ordering schema allows one to express rule iteration and conditional branching of rules. Furthermore, it allows one to map outputs of a rule to inputs of another rule. A rule ordering schema may also refer to other transformation definitions. Thus it is possible to have a composition hierarchy. This feature promotes transformation reuse and scalability.

Summarizing the execution process of a transformation description: upon the firing of a transformation, one or more transformation rules are applied to the one or more source models, according to the control flow defined by the rule ordering schema.

Even though the graphical language that our approach proposes offers a widely applicable and useful set of abstractions for specifying transformations, we believe that some of the complex algorithms required for model transformations in general are easier defined in a procedural language, such as, Java, C#, etc. As such, we propose a means to integrate our language with general-purpose programming languages. In particular, we chose the Java object-oriented programming language [SGB00] as the target language. The idea is that our graphical language would be used to describe the transformation rules— matching and result schemas—and the orderings of the transformation rules—rule ordering schema, and the Java programming language would be used to define those parts of the transformation that are easier expressed with it, e.g., complex transformation algorithms. We propose integration at the programming language level by generating code for the chosen programming language from the language of rule specifications and rule ordering schemas. In this way, we would allow users the full expressive power of a general-purpose programming language, yet a set of abstractions that are fine-tuned to transformations. We believe that this compromise offers quite some potential because it leaves the door open for the user to work in a way that is most comfortable to him/her. Due to space considerations, we do not go into any further details of this aspect of the approach.

3.2 Example

To illustrate a transformation described using the VMT approach, we revisit the example introduced in Section 3.1. This transformation involves taking an association class and replacing it by a class with two associations to the previously connected classes. Since this is a simple example, we will define the transformation using only a single rule specification, and we will omit the rule ordering schema (since there is only a single rule).



id: C1
type: Class
name: *any*
association: Set{AC1_E2}
constraint: *any*
feature: *any*
generalization: *any*
isActive: false (default)
ownedElement: *any*
specialization: *any*

additional selection constraints:
-

id: C2
type: Class
name: *any*
association: Set{AC1_E1}
constraint: *any*
feature: *any*
generalization: *any*
isActive: false (default)
ownedElement: *any*
specialization: *any*

additional selection constraints:
-

id: AC1_E2
type: AssociationEnd
name: *any*
aggregation: none (default)
association: AC1
changeability: *any*
isNavigable: *any*
multiplicity: 0..1
ordering: unordered (default)
participant: C1
qualifier: empty (default)
targetScope: instance (default)
visibility: public (default)

additional selection constraints:
-

id: AC1 -> REMOVED
type: AssociationClass
name: *any*
association: *any*
connection: Set{AC1_E1, AC1_E2}
constraint: *any*
feature: *any*
generalization: *any*
isActive: false (default)
ownedElement: *any*
specialization: *any*

additional selection constraints:
-

id: AC1_E1
type: AssociationEnd
name: *any*
aggregation: none (default)
association: AC1
changeability: *any*
isNavigable: *any*
multiplicity: 0..*
ordering: unordered (default)
participant: C2
qualifier: empty (default)
targetScope: instance (default)
visibility: public (default)

additional selection constraints:
-

Figure 3 – Graphical Depiction of the Matching Schema for the Association Class Transformation

Figure 3 shows the matching schema for the transformation. The UML-like class diagram in Figure 3 depicts the matching graph in model-level view mode. We also offer another visual form for the graph (not shown here): metamodel-level view mode, in which case the graph is shown in a notation neutral manner—all the different kinds of UML model elements are shown as objects in a UML-like object diagram. Each of the 5 boxes shown in Figure 3, which we refer to as a properties-constraint box, provides a set of conditions that must be observed in the selection of a source model element. The correspondence between the boxes and the diagram is made by the id property value of the box and the label of the element in the diagram. Note that we envisage that the tool would not show all boxes at the same time. Instead, the tool would provide a window pane for the diagram and one for a single properties-constraint box. The user could then select the appropriate properties-constraint box by selecting the corresponding element in the diagram.

Figure 3 shows two classes, labeled C1 and C2, and an association class, labeled AC1, which connects these two classes. Two association ends, labeled AC1_E1 and AC1_E2 are labeled using "callout" boxes. This notation is used because association ends do not have a corresponding graphical representation in UML. Also, the association class is graphically crossed out, which means that the element will be removed with the execution of the transformation.

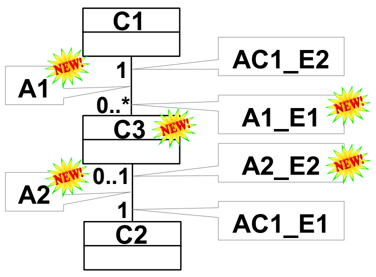
Looking closer at each properties-constraint box, the id property provides a name that can be used to refer to the bound element. The other properties shown below id are properties of the metaclass of the element; as such, this list is specific for each different kind of metaclass of the element. The any condition simply indicates that the corresponding property is unrestricted, i.e., a possible candidate element may have any value for this property. There is also a space for additional selection constraints. This clause provides an additional constraint, written in OCL, that must be observed for a valid match.

The meaning of the matching schema depicted in Figure 3 is the following: The transformation will try to bind all elements of the matching graph to elements in the source model. This will only occur if the source model has two classes with an association class connecting them, which has the multiplicities 0..1 and 0..* at either end. Each properties-constraint box makes some additional restrictions on binding, e.g., elements that bind to either C1 or C2 must have the isActive property equal to false, elements that bind to either AC1_E1 or AC1_E2 must be unordered and not an aggregation, etc. If all constraints of the diagram and properties-constraint boxes are satisfied then a binding is made and the transformation rule will fire.

Figure 4 shows the result schema for the transformation. The three boxes that surround the UML-like class diagram depict the properties-definition boxes for three of the nine element definitions implied by the diagram. Note that the other boxes have been omitted for space considerations. Figure 4 shows three classes, labeled C1, C2 and C3, two associations, labeled A1 and A2, and four association ends, labeled AC1_E1, AC1_E2, A1_E1 and A2_E2. In addition, five "new" labels are shown. These labels signify that the corresponding element definition will result in a new element (as opposed to a modified or unmodified existing one).

Looking closer at each properties-definition box, the element definition with id C1 states that the element bound to C1 (a class) by the matching schema will be left unchanged. The element definition with id AC1_E1 states that the bound association end will be modified in two ways: its multiplicity will be set to 1 and its related association property will be set to the association denoted by A2. Finally, the element definition with id C3 states that a new class will be created that has mostly the same properties of the association class that was bound to id AC1, with one exception: it will have the association ends of A1_E1 and A2_E2.

The meaning of the result schema depicted in Figure 4 is the following: if the rule has permission to fire (according to matching schema), it will create one class (C3), two associations (A1 and A2) and two association ends (A1_E1 and A2_E2), and modify two association ends (AC1_E1 and AC1_E2), according to the statements given in their corresponding properties-definition boxes. And, the transformation will leave untouched two classes (C1 and C2).



id: AC1_E1
type: AssociationEnd
result: modified
name: -
aggregation: -
association: A2
changeability: -
isNavigable: -
multiplicity: 1
ordering: -
participant: -
qualifier: -
targetScope: -
visibility: -

additional statements:
-

id: C1
type: Class
result: unmodified
name: -
isActive: -
feature: -
association: -
constraint: -
generalization: -
specialization: -
ownedElement: -

additional statements:
-

id: C3
type: Class
result: new
name: AC1.name
isActive: AC1.isActive
feature: AC1.feature
association: -
constraint: AC1.constraint
generalization: AC1.generalization
specialization: AC1.specialization
ownedElement: AC1.ownedElement

additional statements:
```
// start association
Set assocEndSet = new Set ();
assocEndSet.add(A1_E1);
assocEndSet.add(A2_E2);
this.association = assocEndSet;
// end association
```

Figure 4– Graphical Depiction of the Result Schema for the Association Class Transformation

4 Formal Aspects of Model Transformation

As mentioned in the above section, our approach considers a model, roughly speaking, as a graph. Thus, the theoretical foundation of model transformation in VMT can be easily expressed in terms of graph transformations that have been studied in [Roz97]. This section presents briefly such aspects adapted to the VMT approach. First of all, we give some basic definitions related to the notion of graph transformation as the application of elementary transformation rules and then we discuss how such transformation rules are used in VMT. Finally, we present the correspondence between the notion of model considered in VMT and the notion of graph presented below.

4.1 Graph Transformation

A ***labeled graph*** $G=(N,E,l)$ consists of a finite set of nodes N, a finite set of edges E such that the elements of E are 2-elements subsets of V denoted by $E\subseteq[V]^2$, and a mapping l assigning a *labeling* symbol to each edge. A graph K is a ***subgraph*** of G, denoted by $K \subseteq G$, if the node and edge sets of K are subset of the respective sets of G, and the label mappings of K coincide with the one of G restricted to K. We say that K has an ***occurrence*** in G, if there is a mapping o, which maps the nodes and edges of K to the nodes and edges of G, respectively, and preserves labelings.

A *graph transformation* consists of applying *transformation rules* to a graph iteratively. Each rule application transforms a graph into another graph by specifying a graph pattern and the elements that are removed and added (a modified element can be removed and added with new values). A ***transformation rule*** $r=(M,R,A,c)$ consists of a graph $M=(N,E,l)$ called the ***matching schema***, $R=(N_R,E_R,l_R)$ corresponds to the nodes, the edges, and the labelings that are removed ($N_R \subseteq N$, $E_R \subseteq E$), $A=(N_A,E_A,l_A)$ groups the components that are added, and c is a condition for the application of the rule.

An ***application*** of a rule $r=(M,R,A,c)$ to a given graph G yields a resulting graph H, provided that H can be obtained from G in the following four steps:

1. CHOOSE an occurrence of the matching schema M in G.
2. CHECK the condition c.
3. REMOVE the components of R from G, in addition all the edges incident to a removed node are removed.

4. ADD the components of A in G, in addition all the nodes related to an added edge are also added.

The condition c is in general the fact that the matching schema is isomorphic to the chosen occurrence, but other conditions can be added such that some global conditions on the graph to be transformed. The application of a rule r to a graph G yielding a graph H is called a ***direct derivation*** from G to H through r, denoted by $G \Rightarrow_r H$ or simply by $G \Rightarrow H$. Given a set of rules P, the successive direct derivations $G_0 \Rightarrow G_1 \Rightarrow G_2 \Rightarrow \dots \Rightarrow G_n$ is a ***derivation*** from G_0 to G_n by rules of P. Since there are possibly several occurrences of a given matching schema (that can even overlap) and because there are many transformation rules and the direct derivation order is relevant, all the derivations form a set of graphs, which can be infinite. A set of terminals symbols T together with a set of rules P and an initial graph S form a ***graph grammar.*** The set of all graphs labeled with terminal symbols of a set T that can be derived from an initial graph S and a set of rules P, is the ***language*** generated by the graph grammar made of S, T, and P. This language is denoted by $L(S,T,P)$.

4.2 Graph Transformation and the VMT Approach

We observe that at the meta-model level the notion of attributes is essential. Thus, we introduce the notion of ***attributed labeled graph*** [Roz97] that consists of a graph with attributed nodes and edges, i.e. an underlying graph structure equipped with (named) attributes for the edges and the nodes. Formally speaking, attributes are represented as two mappings that map, respectively, the edges and the nodes into the global domains of possible values for attributed edges and nodes. This notion of attributed labeled graph is a central notion of the abstract syntax of VMT. The notions of occurrence, transformation rule, and derivation remain very close to the definitions given in the previous section, the greatest change concerns the definition of the occurrence that, now, has to preserve not only the labelings, but also the mappings that associate attributes to the nodes and the edges.

The VMT approach provides a **rule ordering schema** that allows the user to formulate a graph transformation as a chain of simpler transformations. A rule ordering schema is nothing else than an attributed labeled graph in which the edges are directed. Naturally, we introduce directed graph by means of two additional mappings that associate a start and an end node to each edge in order to obtain an ***attributed directed labeled graph***. Thus, a rule ordering schema is formally represented as an attributed directed labeled graph in which each node corresponds to a single transformation rule. Since no choice is allowed for the application of a single transformation rule, the application of such a rule must terminate and the successive direct derivations of a single rule must be confluent, i.e. give a unique result. The chain of derivations generated by the rule ordering schema produces a unique graph that is the result of the graph transformation that belongs to the language generated by the set of transformation rules.

5 Related Work

In this section, we survey some of the approaches and technologies that are related to our approach and/or influenced its development in one way or another.

One approach to UML model transformation is to encode them directly in a procedural language using an API to the model repository offered by a UML tool. An advantage of this approach is that developers do not need any extra training to become operational. However, a disadvantage is that encoding transformations in a procedural language can be time-consuming and difficult to understand and maintain due to a lack of high-level abstractions for transformation specification. Also, tool model repositories usually restrict the kind of transformations that can be performed, simply because the API may not let one access or manipulate the required model information.

One proposal that promises to raise the level of abstraction of operations on UML models is UML's action language [Omg03a]. The language has been proposed as a way to procedurally define UML transformations [MB02, SPH+01] and is a special-purpose language for manipulating UML models. However, due to its "general-purpose" context, the UML action language still suffers, albeit less chronically, from a lack of high-level abstractions for dealing with model transformations.

Another technique is to treat UML models as graphs. Much work has been performed on graph grammars and graph transformation systems. Graph transformations are realized by the application of transformation rules, which are rewriting rules for graphs. A transformation rule consists of a graph to match, commonly referred to as LHS, and a replacement graph, commonly referred to as RHS. If a match is found for the LHS graph, then the rule is fired, which results in the matched sub-graph of the graph under transformation being replaced by the RHS graph. The PROgrammed GRaph REplacement System (PROGRES) approach [SWZ97] offers a mean to not only specify transformation rules but to also define the sequencing of these rules, described using imperative constructs. There are several other tools that are based on graph transformation theory; such as AGG, GenGed [BEW02], and one dedicated to UML and Java, Fujaba [FUJABA].

The use of logic programming languages have also used in the context of model transformations. In [Whi02], a framework dedicated to UML model transformations implemented in MAUDE programming language [CDE+01] is presented.

Milicev proposes a graphical language for specifying model transformations [Mil02]. The approach proposes an extended UML object diagram as notation for developing the mapping specification. The diagrams are extended with the concepts of conditional, repetitive, parameterized, and polymorphic model creation, using UML's stereotype extensibility mechanism. Execution support for the proposed language is offered by a mapping to C++ code. Also, the approach offers the ability to reuse fragments of mapping specification through the use of parameterized model creation. An important limitation of the approach is its underlying assumption that the selection of source model elements for the transformation can be easily expressed in a general-purpose programming language, i.e., C++. If one were faced with complex selection criteria, it would be very likely that these selection conditions would become complex and hard to maintain. In fact, it would be at least useful to offer a language that is tailored for such a purpose, such as, UML's Object Constraint Language (OCL) [Omg03a].

UML's Object Constraint Language [WK98] has also been proposed as a way to declaratively describe UML model transformations, e.g., [PVJ02, SPL+01]. Kleppe et al. [KWB03] define a transformation language that uses OCL to specify the conditions for the firing of a transformation and the elements that are input to and output of the transformation. One nice feature of their approach is the way that it facilitates the definition of bi-directional transformations, so that one can perform a transformation in either direction. The approach has many similarities to our approach. However, we chose a principally graphical notation, compared to their fully textual notation, which we believe has advantages in terms of usability and conciseness of description.

6 Conclusion

For model-driven software development approaches to become a reality in mainstream software development practice, software development tools need to be able to better automate the creation, evolution and maintenance of the models used throughout the software lifecycle. In this direction, one of the keys areas is the easy description and execution of UML model transformations.

In this paper, we presented the Visual Model Transformation (VMT) approach. The VMT approach offers a visual and declarative language to specify UML model transformations. Transformations can be defined in the proposed language by specifying transformations rules and defining the order in which these ones are to be executed.

Further work on the VMT approach will consist in continuing work on the design and implementation of the second version of MEDAL tool to support the proposed VMT approach, and developing the formal foundations of the language in order to be able to prove transformation properties based on the source and target model semantics.

7 Acknowledgements

We would like to thank Wojtek Kozaczynski and Jim Thario who gave us a lot of inputs on Model transformations and Rational XDE. We also would like to thank the other SE2C team members for their valuable comments.

8 References

[BEW02] R. Bardohl, C. Ermel and I. Weinhold, "AGG and GenGED: Graph Transformation-Based Specification and Analysis Techniques for Visual Languages", Electronic Notes in Theoretical Computer Science, volume 72 issue 2, Elsevier Science B.V, 2002

[BG00] D. Buchs and N. Guelfi; "A formal specification framework for object-oriented distributed systems". IEEE Transactions on Software Engineering, special issue on Formal Methods for Open Object Based Distributed Systems, July 2000, vol. 26, n°7, pp 635-652.

[CDE+01] M. Clavel, F. Durän, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesda; "Maude: Specification and Programming in Rewriting Logic". Theoretical Computer Science, 2001.

[DiM99] G. Di Marzo Serugendo; "Stepwise Refinement of Formal Specifications Based on Logical Formulae: from COOPN/2 Specifications to Java Programs". Ph.D. Thesis, no 1931, Ecole Polytechnique Fédérale de Lausanne, Département d'informatique, CH-1015, Lausanne, Suisse, 1999.

[Eclipse03] Eclipse IDE website: http://www.eclipse.org

[FIDJI] N. Guelfi, G. Perrouin, B. Ries and P. Sterges; "FIDJI Project Annual Activities Report". Applied Computer Science Department technical report n° TR-DIA-03-01, Luxembourg University of Applied Sciences, Luxembourg-Kirchberg, Luxembourg, 2002.

[FUJABA] Fujaba project website : http://www.fujaba.de

[GP02] N. Guelfi and G. Perrouin; "Rigourous Engineering of Software Architectures: Integrating ADLs, UML and Development Methodologies". In Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA'02), November 4-6 2002 Cambridge, USA.

[GR03] N. Guelfi, B.Ries," JAFAR2:An Extensible J2EE Architectural Framework for Web Applications", submitted to PFE-5, Fifth International Workshop on Product Family Engineering, November 4-6, 2003, Sienna, Italy

[GS02] N. Guelfi and P. Sterges; "JAFAR: Detailed Design of a Pattern-based J2EE Framework". In Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA'02), November 4-6 2002 Cambridge, USA.

[GS03] N. Guelfi and P. Sterges, "MEDAL: A CASE Tool Extension for Model-driven Software Engineering", submitted to SwSTE'03, IEEE International Conference on Software - Science, Technology & Engineering, November 4-5 2003, Herzelia, Israel

[Gue01] N. Guelfi; "Flexible Consistency In Software Development Using Contracts and Refinements". 2nd International Workshop on Living With Inconsistency, part of the International Conference on Software Engineering - ICSE'01, May 2001, Toronto, Canada.

[KT02] W. Kozaczynski and J. Thario; "Transforming User Experience Models To Presentation Layer Implementations". OOPSLA 2002, Second Workshop on Domain-Specific Visual Languages available on http://www.cis.uab.edu/info/OOPSLA-DSVL2/Papers/

[Mil02] D. Milicev; "Domain Mapping Using Extended UML Object Diagrams". IEEE Software, pp. 90-97, March/April 2002.

[Omg01a] OMG CWM Partners; "Common Warehouse Metamodel (CWM) Specification", Feb. 2001. http://www.cwmforum.org/spec.htm

[Omg01b] OMG Architecture Board ORMSC; "Model Driven Architecture (MDA)". July 9, 2001 (draft). http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

[Omg02a] OMG TC; "MOF 2.0 Query/Views/Transformations RFP", 2002. http://cgi.omg.org/cgi-bin/doc?ad/02-04-10

[Omg03a] OMG Unified Modeling Language Revision Task Force; "OMG Unified Modeling Language Specification". Version 1.5, March 2003. http://www.omg.org/technology/documents/formal/uml.htm

[Omg03b] OMG UML profiles, 2003. http://www.omg.org/mda/specs.htm#Profiles

[PVJ02] D. Pollet, D. Vojtisek, J-M. Jézéquel; "OCL as a Core UML Transformation Language". WITUML: Workshop on Integration and Transformation of UML models (held at ECOOP 2002), Malaga, Spain, June 2002.

[Roz97] G. Rozenberg (ed.); "Handbook of graph grammars and computing by graph transformation: Volume I Foundations". World Scientific Publishing, 1997.

[SGB00] J. Gosling, B. Joy, G. Steele, and G. Bracha; "The Java Language Specification". Second Edition, Addison Wesley, June 2000

[SPH+01] G. Sunyé, F. Pennaneach, W-M. Ho, A. Le Guennec and J-M. Jézéquel; "Using UML Action Semantics for Executable Modeling and Beyond". Proceedings of 13th International Conference, CAiSE 2001, Switzerland. LNCS (Lecture Notes in Computer Science), no. 2068, pp. 433-447, Springer Verlag, 2001.

[SPL+01] G. Sunyé, D. Pollet, Y. Le Traon and J-M. Jézéquel; "Refactoring UML Models". UML 2001 — The Unified Modeling Language: Modeling Languages, Concepts, and Tools, 4th International Conference, Canada; Gogolla and Kobryn (Ed.), LNCS (Lecture Notes in Computer Science), no. 2185, pp. 134-148, Springer Verlag, 2001.

[SRP03] P. Sterges, B. Ries, G. Perrouin "UML-to-UML model generic Transformations: the MEDAL tool", Applied Computer Science Department technical report n° TR-DIA-03-02, Luxembourg University of Applied Sciences, Luxembourg-Kirchberg, Luxembourg, 2002.

[Struts03] Apache Struts framework website: http://jakarta.apache.org/struts/index.html

[SWZ97] A. Schürr, A. Winter and A.Zündorf; "The Progress Approach: Language and environment". In Chapter13 of G. Rozenberg (eds), *Handbook of graph grammars and computing by graph transformation: Volume II Applications, Languages and Tools*, World Scientific Publishing, 1997.

[Whi02] J. Whittle; "Transformations and Software Modeling Language: Automating Transformations in UML". Jézéquel, Hußmann & Cook (Eds.): Proceedings of UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany. Lecture Notes in Computer Science no. 2460, pp. 227-242, Springer, 2002.

[WK98] J. Warmer and A. Kleppe; "The Object Constraint Language: Precise Modeling With UML". Addison-Wesley 1998.

[XDE] Rational XDE Web Site; Rational Software Corporation, 2003. http://www.rational.com/products/xde/index.jsp