JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2010**

Vol. 9, No. 2, March - April 2010

A Rule-Driven Approach for composing Viewpoint-oriented Models

Adil Anwar[1,3], Sophie Ebersold[1], Bernard Coulette[1]
Mahmoud Nassar[2] , Abdelaziz Kriouile[2]
[1] University of Toulouse, IRIT, UT2, 5 allées A. Machado, F-31058 Toulouse, France
[2] SI2M, ENSIAS, BP 713 Agdal, Rabat, Morocco
[3] UFR ACSYS, Faculty of Sciences, Rabat, Morocco

Abstract

Model composition is a crucial activity in Model Driven Engineering (MDE). It is particularly useful when adopting a multi-modeling approach to analyze and design software systems. In previous works, we defined a view-based UML profile called VUML. In this paper, we describe a composition process and a MDE-based framework, which contains a generic composition part, and a specific part dedicated to a given modeling domain. To illustrate our approach, we apply it to the composition (merging) of two UML class diagrams into one VUML class diagram. The composition operator is implemented as a ruled-based transformation in ATL.

1 INTRODUCTION

Several approaches adopted by the software engineering community rely on the principle of multi-modeling, which allows to separate concerns and to model a system as a set of less complex sub-models. This principle has been introduced in several programming approaches like subject-oriented programming [Ossher96] or aspect-oriented programming [Kiczales97]. At the model level, comparable approaches use concepts such as Views/Viewpoints [Finkelstein90], Subject-oriented development [Clarke 02], and Aspect-oriented Modeling [Baniassad04] [France04]. With all these approaches, the key issue is the composition of (sub-) models. Composition consists of combining one or several source models to create one or several target models. In Aspect Oriented Software Development, the composition is called weaving [Kiczales97] [France04]. In the field of Databases, the composition of views can be seen as an integration of different views of the same database or of heterogeneous and possibly distributed database schemas [Batini86]. In Requirement Engineering, viewpoints are used to describe system requirements as a collection of modules that are merged to get a global view on the system [Sabetzadeh05] [Finkelstein90].

Our work in this area [Nassar03] led to the definition of the VUML profile (View based Unified Modeling Language). VUML proposes a formalism and a methodology to support view-based modeling from analysis to coding. VUML enables to model a software system according to each actor's viewpoint. First, actors of the system are identified as in UML. Each actor is associated with a unique viewpoint. Then, for each viewpoint, we describe, in an iterative way, use cases and scenarios as well as related classes. The result is a set of class diagrams (called also viewpoint models) in the UML formalism. Finally, a VUML model is produced by composing the partial models.

Since the OMG's MDA initiative [Soley00], Model Driven Engineering (MDE) has been taking an increasing place in the software development process. It consists of centering activities on the paradigm of model considered as a first class entity [Bézivin06]. The main interest of this approach is to describe models at different abstraction levels in order to facilitate their reuse during the development process. To face the composition issue, MDE appears to be an elegant solution since one can consider some steps of the composition process as transformations. For these reasons, we have adopted the MDE approach and especially the transformation paradigm to partially automate the model composition in VUML (see [Anwar08a] [Anwar08b] for more details). More precisely, we define the composition of static UML models as a set of transformation rules classified as correspondence, merging and translation rules. These rules allow first to establish correspondences between input models, and second to merge viewpoint models into a global VUML model shared by all the actors.

The main contribution of this paper regarding our previous work (cf. [Anwar08b]) is the focus on the reusability of the composition process. For this purpose, we propose a two level composition approach: a generic level independent from any modeling language, and a specific level that depends on a given modeling domain. The generic level is defined as a generic composition framework. This framework is independent from any specific transformation language and provides means to express the key features necessary to compose models automatically. The framework comprises a generic relationship metamodel, a generic transformation rules metamodel and a transformation strategies metamodel. For a given modeling language (e.g. source models conform to UML and target model conform to VUML), we specialize the generic framework by (i) specializing correspondence relationships, (ii) defining transformation strategies (iii) and defining transformation rules so as to generate a set of executable composition-oriented transformations.

The rest of this paper is organized as follows: Section 2 presents the context and the motivation of our work by introducing the VUML approach and a case study. Section 3 describes the composition process according to a rule driven approach. Section 4 is devoted to the description of the generic composition framework. In section 5 we show how the generic framework can be specialized for a given modeling language, and in Section 6, we apply this specialization process to compose UML models (class diagrams) to VUML profile. Section 7 describes the main related works. In section 8, we discuss some issues raised by our work and we conclude this paper in Section 9.

2 CONTEXT AND MOTIVATION

In this section, we first give a brief overview of the VUML profile, and describe our view-based modeling approach in the context of MDE. To highlight and motivate our approach, we present a case study and show how UML class diagrams can be composed into one VUML class diagram.

VUML profile

The VUML profile was developed to meet the needs of complex systems analysis and design according to various viewpoints. In VUML, a viewpoint represents the perspective from which a given actor interacts with the system. In other words, a viewpoint expresses an actor's requirements and rights. A view is the result of the application of a viewpoint on a given entity of the system. The main new concept added by VUML to UML is the *multiview class* which is composed of a *base* class (shared by all viewpoints), and a set of *view* classes (extensions of the base class), each view class being specific for a given viewpoint. VUML's semantics is described by a metamodel, a set of well-formed rules expressed in OCL [OMG03b], and a set of textual descriptions in natural language. On the methodological level, a process allows to analyze and design software systems with respect to viewpoints. A multi-target code generator was developed to produce object code from VUML class diagrams and was tested with Java as target language [Nassar09].

MDA-based design process

In spite of the large amount of research works published [Bézivin06], there is no consensus related to the composition of models. To deal with model composition issues, MDE has appeared as a promising solution since one can consider some steps of the composition as special cases of model transformations. For this reason, we decided to use MDE techniques and particularly the transformation concept to formalize and implement the composition of several viewpoints models into one VUML model. More precisely, our composition can be regarded as an exogenous transformation of the same level of abstraction (horizontal transformation) (PIM UML to PIM VUML) because it takes in the input a set of PIM models expressed in UML and generates in the output another PIM model which is conform to the VUML profile.

VUML aims to reduce the design complexity of software systems trough decomposition according to the needs and access rights of the system actors. This horizontal separation of concerns completes the vertical approach of MDA by proposing a methodology that permits to develop models at each level of abstraction. However, to fully integrate a model driven approach, it is important to define and automate the transformations between the involved models. Figure 1 illustrates the VUML process with respect to MDA.

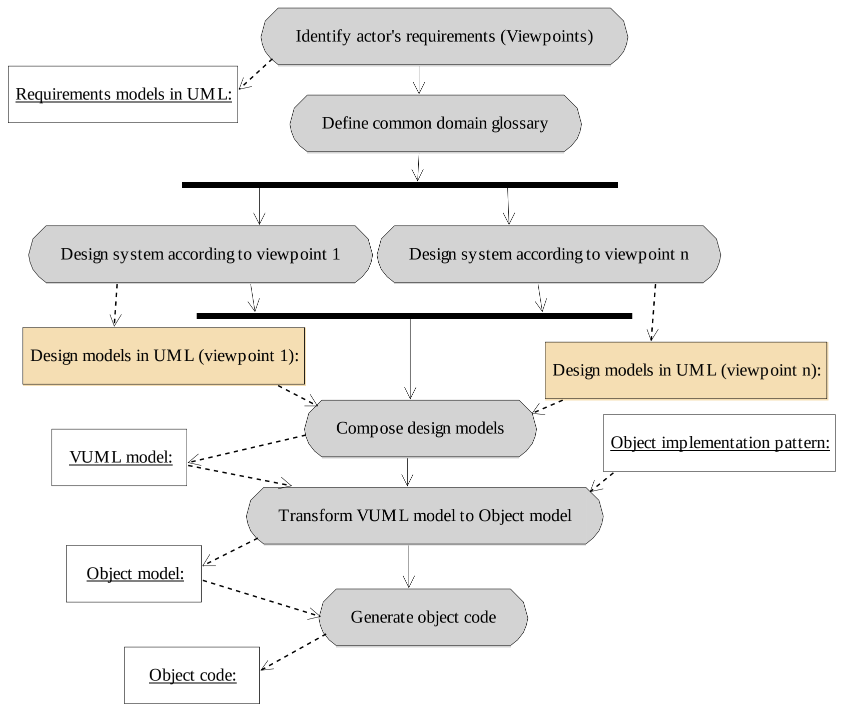
Identify actor's requirements (Viewpoints)
Requirements models in UML:
Define common domain glossary
Design system according to viewpoint 1
Design system according to viewpoint n
Design models in UML (viewpoint 1):
Design models in UML (viewpoint n):
Compose design models
VUML model:
Object implementation pattern:
Transform VUML model to Object model
Object model:
Generate object code
Object code:

Figure 1. MDA-based general VUML process

The first phase of the design process with VUML is the identification of actors' needs. The main goal is to create a requirements model (UML use case diagram). In addition, to reduce the possible inconsistencies of models carried out during the design phase, the requirements model is enriched with a glossary that specifies the basic concepts of the domain and serves as a reference for the designers of the system.

The second phase of the process, decentralized, consists of developing separate PIM models, each one representing a viewpoint. The result of this phase is a set of UML models (class diagrams, state machines, sequence diagrams, etc.). These models are produced according to an iterative process like RUP [Kruchten99]. In this paper, we focus our study on structural models (class diagrams).

The third phase of the process is composed of three steps. The first is a pre-composition step that reveals and fixes the different conflicts on these models (names, structural, etc.). So far, this step is done manually by a designer (cf. discussion in Section 8). The second step, automatic, aims at composing PIM models. This composition is an exogenous transformation because it takes as input n PIM models defined in UML and generates as output one model conform to the VUML profile. Once the model is generated, the third step consists of refining it. This refinement operation is represented (Figure 1) by a reflexive relation on the VUML model. During this step, possible dependencies between the view classes of a given multiview class must be identified and described in order to ensure the consistency of the system model. These dependencies are modeled in VUML by dependency relationships which are stereotyped by “viewDependency”, and annotated by constraints expressed in OCL language.

The last phase of the design process considers an execution platform. It operates to transform the VUML model into an implementation model according to the platform. This phase is carried out by applying an object code generation pattern as described in [Nassar09]. The technique applied combines the use of model transformation and design patterns (as described in [Jouault05]), and gives place the development of a model-to-model transformer in ATL (VUML2JAVA).

Case study

To illustrate our approach, we consider a Shared Medical File Management System (SMFMS). To simplify, we limit our study to the following actors and activities:

- *Patients* follow the treatments prescribed by doctors and undergo analysis in laboratories. They can also consult their medical files.
- *Doctors* carry out diagnoses and consultations, write prescriptions, prescribe drugs, and consult medical reports.

a) Requirements modeling of SMFMS

The analysis phase of the system consists of capturing its functional requirements. This phase, centered on the actors, gives place to one use case diagram per viewpoint. Figure 2 below illustrates a subset of the use cases identified for the doctor’s viewpoint.

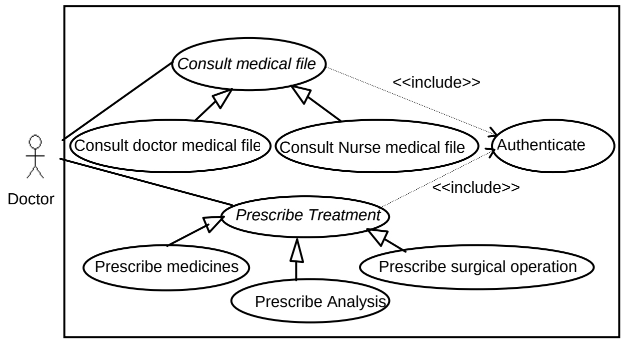

Figure 2. Excerpt from the Use case Diagram (Doctor’s viewpoint)

b) Viewpoint Modeling of SMFMS

During this phase the system is modeled according to given viewpoints. Let us consider the doctor’s viewpoint and, for instance, the scenario “Record a treatment” of the use case "Prescribe treatment”. The purpose of this scenario, started by the doctor after a consultation, is to create a “ConsultationForm” which is used to record all the clinical acts concerning a given patient and the decisions taken by the doctor during the consultation. While proceeding in an incremental way with all use cases, the identified objects, as well as the methods associated appearing in the sequences diagrams, allow to build a class diagram related to the doctor’s viewpoint (Figure 3). We can notice that the doctor has access to information on the analysis carried out in laboratories, as well as to the reports created by other health professionals (doctors, nurses, etc).

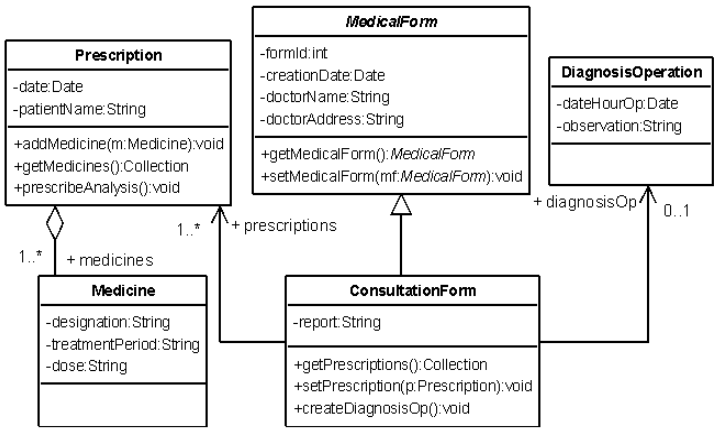

Figure 3. Doctor's viewpoint class diagram

For the patient's viewpoint, a similar process is performed and produces the class diagram of Figure 4. It shows that the patient has access to doctors' prescriptions that concern him, and to the list of payment forms associated to his treatments, but he has direct access neither to the reports of analysis laboratories, nor to the reports written by nurses (that are reserved to doctors).

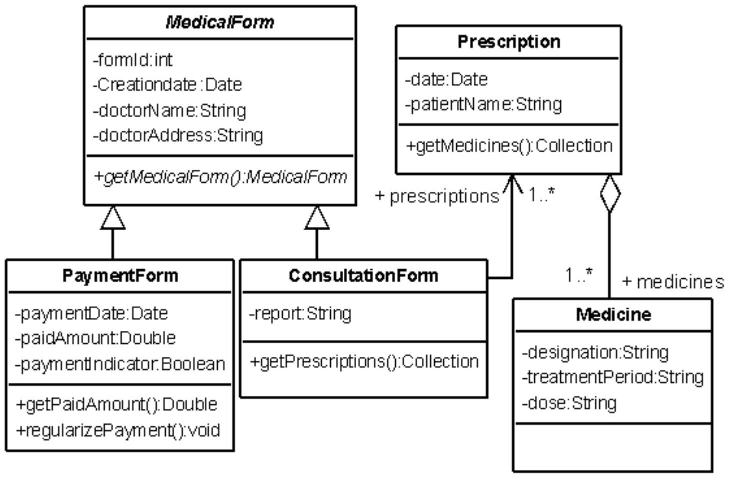

Figure 4. Patient's viewpoint Model

c) VUML Modeling of SMFMS

In this phase, viewpoint models are composed to produce a VUML model. Figure 5 depicts the VUML model resulting from the composition of the two viewpoint models shown above (Figure 3 and Figure 4). Classes appearing in two viewpoint models, with the same name and with different properties (attributes, operations, associations, etc), are merged as a multiview class. Figure 5 shows two multiview classes: *MedicalForm* and *Prescription*. Properties of the class *MedicalForm* that are shared by the two considered viewpoints have been put into the class stereotyped by “base”; properties that are specific of one viewpoint have been put into classes stereotyped by “view”. For reasons of readability, certain multiview classes are displayed in the iconified mode (stereotype “multiViewsClass”). To name the classes stereotyped by “view”, we have adopted the notation recommended by VUML (actor’s name + base class name). This strategy makes it possible to ensure traceability between elements of the viewpoint models and the VUML model. The class *Medecine* is not a multiview class because it is exactly the same (name and content) in the two viewpoint models.

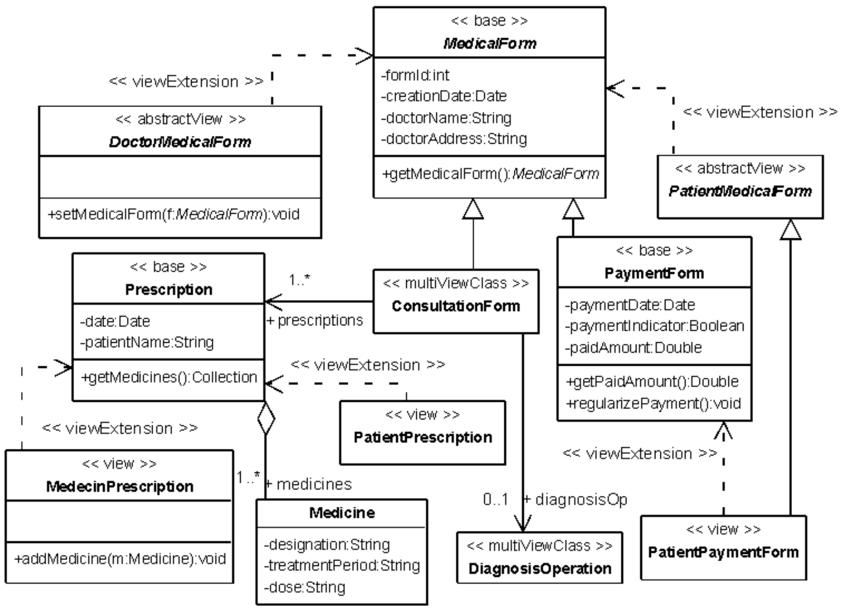

Figure 5. VUML model of SMFMS

3 MODEL COMPOSITION PROCESS

In this section we detail the methodological aspect of our approach trough a model composition process structured into three phases: a pre-composition phase, a composition phase, and a post-composition phase which is semi-automatic (guided by the user). The composition process is represented by an activity flow depicted by Figure 6.

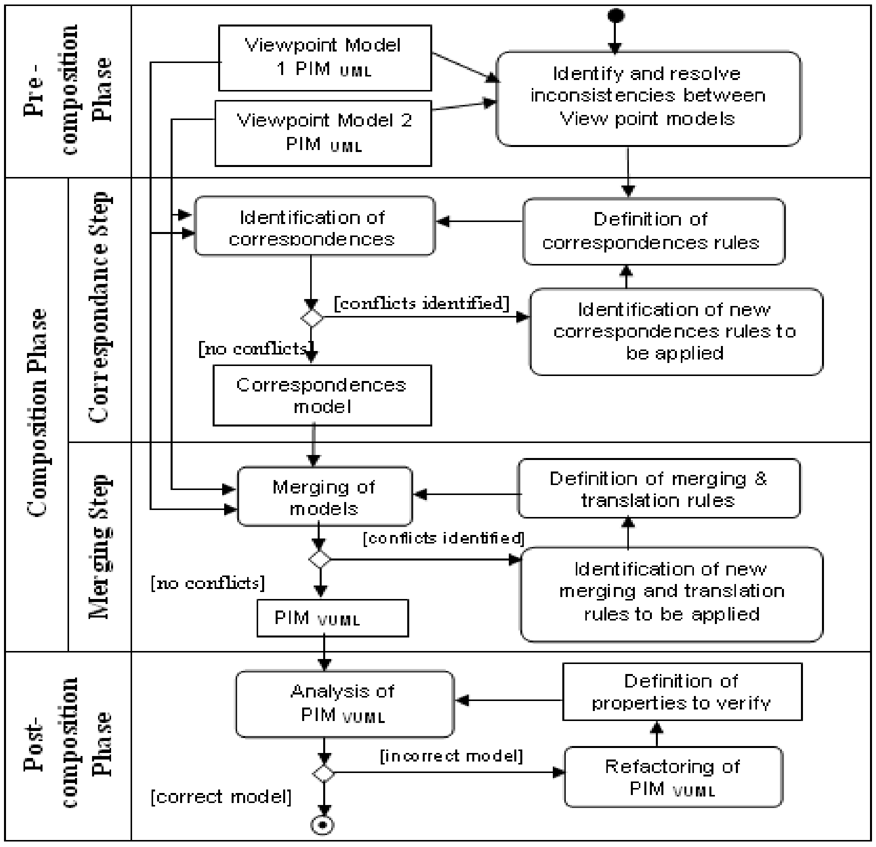
Pre - composition Phase
Composition Phase
Correspondance Step
Merging Step
Post- composition Phase
Viewpoint Model 1 PIM UML
Viewpoint Model 2 PIM UML
Identify and resolve inconsistencies between View point models
Identification of correspondences
Definition of correspondences rules
[conflicts identified]
Identification of new correspondences rules to be applied
[no conflicts]
Correspondences model
Merging of models
Definition of merging & translation rules
[conflicts identified]
Identification of new merging and translation rules to be applied
[no conflicts]
PIM VUML
Analysis of PIM VUML
Definition of properties to verify
[incorrect model]
Refactoring of PIM VUML
[correct model]

Figure 6. Composition Process in VUML

Pre-Composition phase

The main objective of this phase is to harmonize the partial models in order to eliminate possible conflicts (naming, structural, etc) resulting from separate modeling. This is done amongst other things by the resolution of conflicts by determining the inconsistencies and the similarities between the elements of viewpoint design models. This phase copes with conflicts such as polysemy (same name and different meanings), synonymy (same meaning and different names), and structural inconsistencies (in particular generalization versus association relationships). In the latter case, it is necessary to apply heuristics that may be based on patterns, or to require the intervention of the designer who controls the composition process. Besides some of the conflicts identified here may be also due to the fact that the various actors of the system may have contradictory objectives. This particular problem, frequently encountered in requirement engineering, is out of the scope of this paper.

Composition phase

We agree with authors [Fleurey07] [Kleinner07] [Kolovos06] who argue that automating model composition includes two different tasks that should be carried out trough two distinguished operators: a correspondence operator and a merging operator. This separation facilitates the maintainability of the composition process since the correspondence operation is more stable than the merging operation that may obey to modifiable strategies. Therefore we propose a composition operation which is made of two steps: *Correspondence step* and *Merging step* (Figure 7).

The Correspondence step consists of identifying links between models to be composed (to make things easy we consider only two source models here). It is governed by correspondence rules that implement comparison strategies between model elements. Comparison of elements is based on internal properties defined at the metamodel level. For example, a subset of internal properties of an UML class may be represented by {name, isAbstract, ownedAttribute, ownedOperation} which are properties of the metaclass Class in the UML metamodel [OMG03a]. A correspondence rule, applied to two elements describing the same concept in different source models, creates a correspondence relationship between those elements. This relationship is then stored in a correspondence model.

The Merging step depends on the target metamodel. In our application context, this merging step aims to produce a VUML model whose elements are stereotyped according to the VUML profile. In fact, VUML elements are created by applying both merging and translation rules. The merging strategy mainly depends on link type. Merging rules are applied to elements that are related to each other through correspondence relationships. Elements which have no correspondent in the opposite model are simply translated (copied) into the VUML model with respect to translation rules.

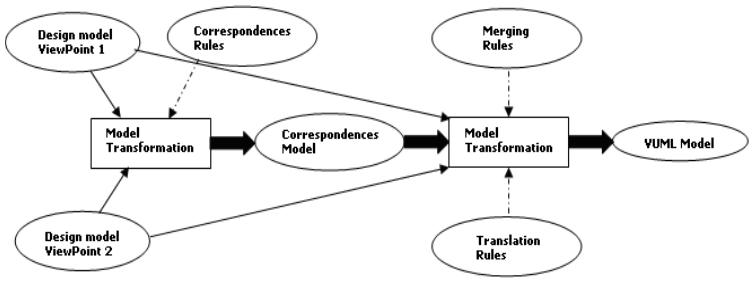

Figure 7. Transformation chain of VUML composition

Post-Composition phase

After the composition phase, an analysis step is performed (see Figure 6) in order to discover possible composition errors. The composed VUML model is checked against desired properties or by verifying its compliance with well-formed rules. When a rule is violated, an error is detected and a problem element is created and stored into a problem model which conforms to a problem metamodel [Bézivin05]. The problem model can be analyzed and then imported into a model refactoring tool dedicated to the resolution of such problems. Well-formed rules defined at the metamodel level to express the static semantics of VUML — in particular those relating to the constructions of the language — are used to develop a property-based proof technique as described in [Cousot90]. OMG recommends the use of OCL language [OMG03b] to express such rules. They enable to add structural properties that could not be captured during the definition of the metamodel. For example one can use the following OCL constraint to impose the fact that a direct descendant of a view class must be a concrete or an abstract view.

```
context view inv :
   self.specialization->forAll(g:Generalization|
   g.child.isStereotyped("view")    or
   g.child.isStereotyped("abstractView"))
```

Figure 8 describes the principle of the transformation used to check the composed VUML model. It produces a diagnosis model giving details on the identified errors.

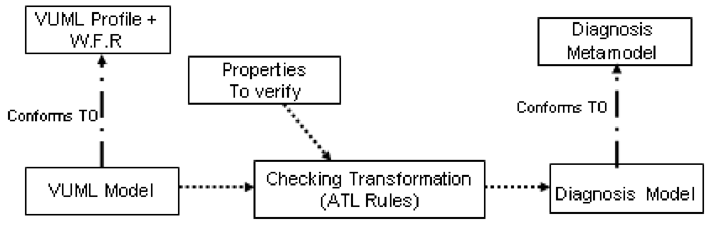

Figure 8. Check of a VUML model with an ATL transformation

This technique is based on an extension of OCL allowing to produce detailed information in an output model instead of a simple Boolean. It is based on ATL to implement the checking rules. An ATL rule is defined for each constraint to check. The context of the OCL constraint defines the type of the pattern source of the rule, while the guard condition is the negation of the boolean expression associated to the constraint. Finally, the type of the target pattern characterizes an error (Problem). This type, defined in the diagnosis metamodel [Bézivin05], gives precise details on the error (severity, localization, description, etc).

4 A GENERIC FRAMEWORK FOR MODEL COMPOSITION

A model composition framework should provide means to support common features for building a composition operator. The survey presented in [Bézivin06] summarizes a core set of minimal requirements for such a framework. We propose to define a generic composition operator through three components: relationships, transformations and strategies. The *relationships* component allows to define and to capture relationships between model elements. The *transformations* component provides means to carry out transformations between involved models. The *strategies* component provides means to define transformation strategies. Strategies specify the semantics of transformation rules. To describe these three types of component, we have defined three metamodels: the correspondence metamodel, the transformation rules metamodel and the transformation strategies metamodel.

Generic correspondence metamodel

The correspondence metamodel defines the different kinds of relationships between model elements independently from any given application domain (Figure 9). We have extended the core weaving metamodel proposed in [Del Fabro05] to support composition requirements and to handle new types of relationship. In what follows, we present the key elements of the correspondence metamodel.

- CorrespondenceRelationship: this metaclass defines the relationship between elements of the source models. The definition of a new relationship is made through a specialization of this metaclass; this allows the semantic definition of each relationship.(e.g. equality, equivalence, dependency, etc)
- CorrespondenceRelationshipEnd: this metaclass represents the extremity of a correspondence relationship.
- CorrespondingElementRef: this metaclass models the concepts of reference. It contains an attribute *name* that represents the name of the referenced element, and an attribute *ref* which acts as a persistent model-element identifier.
- ReferencePackage: this metaclass is a container for reference elements. An instance of this metaclass contains all references of linked elements.

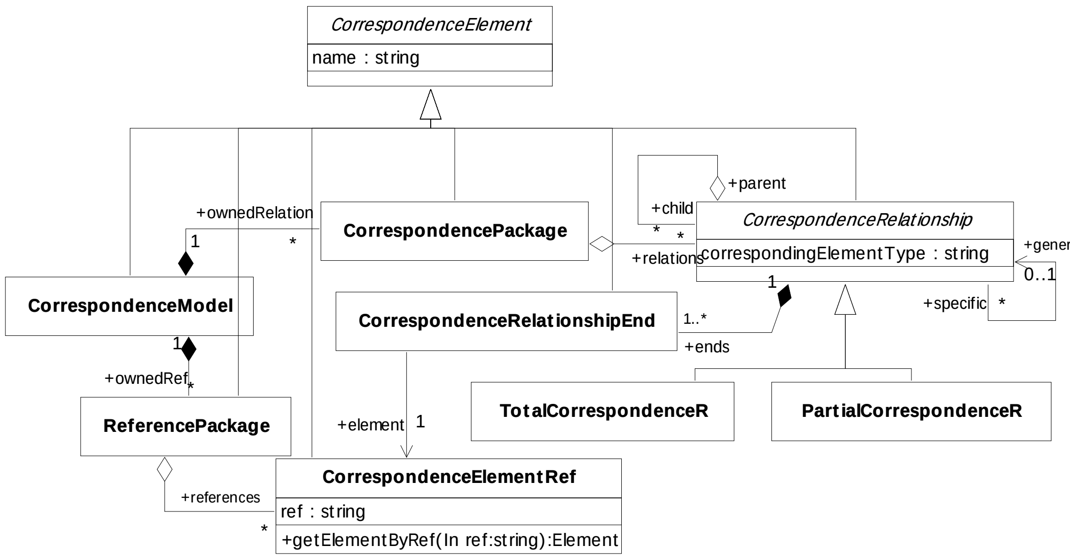

Figure 9. Generic correspondence metamodel.

The *CorrespondenceRelationship* metaclass must be specialized to create various types of relations. It is a practical solution to establish a given semantics for each correspondence relationship. For example, the *PartialCorrespondenceR* metaclass indicates that the elements related by an instance of this relationship are two views which represent the same concept, but differ by certain properties (for example: two classes with the same name but having different attributes or operations). The TotalCorrespondenceR metaclass defines a particular type of relation between elements which represent consistent views of the same concept (i.e equality in the sens that two elements appear in the same way in several models).

Transformation rules metamodel

The composition process is driven by transformation rules structured in three categories: correspondence, merging and translation. This enables first to establish correspondences between input models, second to merge these models into a global model. Elements that have no corresponding element in the opposite model are transformed according to translation rules. As transformation strategies depend on each specific domain, we do not consider them in the generic part of the metamodel. The transformation rules metamodel is a general description of transformation rules (Figure 10).

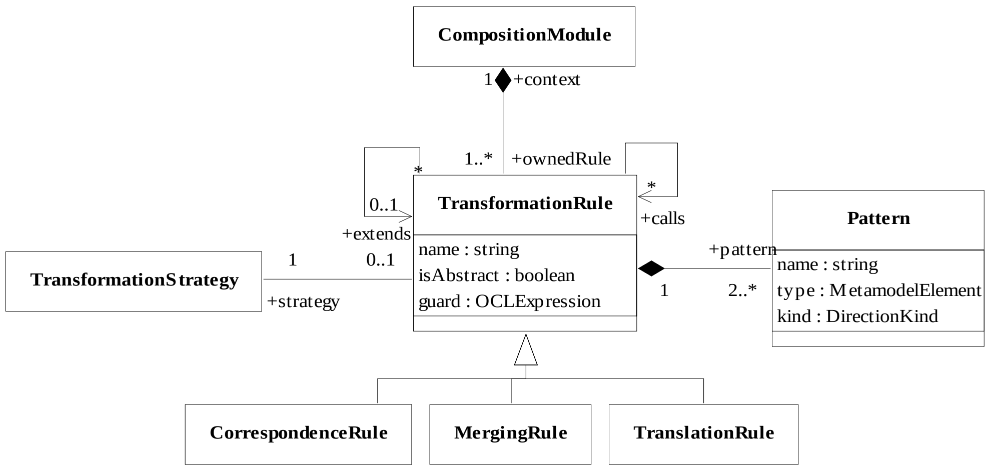

Figure 10. Transformation rules metamodel

Each transformation rule is composed of at least two patterns used to detect elements in source and target models. Pattern types come from the elements of the metamodels involved in the transformation. If we consider a correspondence rule for example, the type of the input pattern comes from the source metamodel, whereas the type of the output pattern comes from the correspondence metamodel.

In Section 5 below we will show how this metamodel allows to define (when instantiated) specific transformation rules to compose models in a particular domain.

Transformation strategies metamodel

So far, we have introduced the structural aspects of our composition operator. However, these structural aspects are not sufficient to provide a comprehensive definition of this operator, as one must describe its behavior as well. We use transformation strategies to specify the behavioral aspect of each transformation rule (Figure 11). Strategies are defined by Kolovos et al. [Kolovos06] as pluggable algorithms that can be attached to transformation rules to implement a recursive and reusable functionality, they may be inferred from the metamodel structure. Using strategies has the advantage of minimizing the manual intervention of the developer.

Correspondence strategies define comparison logic between model elements. We distinguish three types of correspondence strategies. The first type is based on signatures as described in [Reddy06]. The signature of an element is described by a set of internal properties (name, type, cardinality, etc) defined in the metamodel. For elements of type Class, the strategy depends on values of the meta-properties of the metaclass. For example, if one considers the couple (name, isAbstract), then comparing two classes defined in two different models is reduced to compare the values of these two properties. Correspondence strategies may be also based on structural relationships between elements such as inheritance or containment; in this case, the correspondence strategy depends on information about the neighbors of each element in the models.

Unlike correspondence strategies, merging strategies depend on the type and semantics associated with correspondence relationships that link source elements, and on the structure and semantics of the elements to create in the target metamodel. *UnionMergingStrategy* is used when different source models contain classes with the same name but with different properties. A simple union of the initial properties gives the properties of the resulting class. *TotalMergingStrategy* is used for merging two classes, which are in conformity. *PartialMergingStrategy* is used to create two or more elements in the target model.

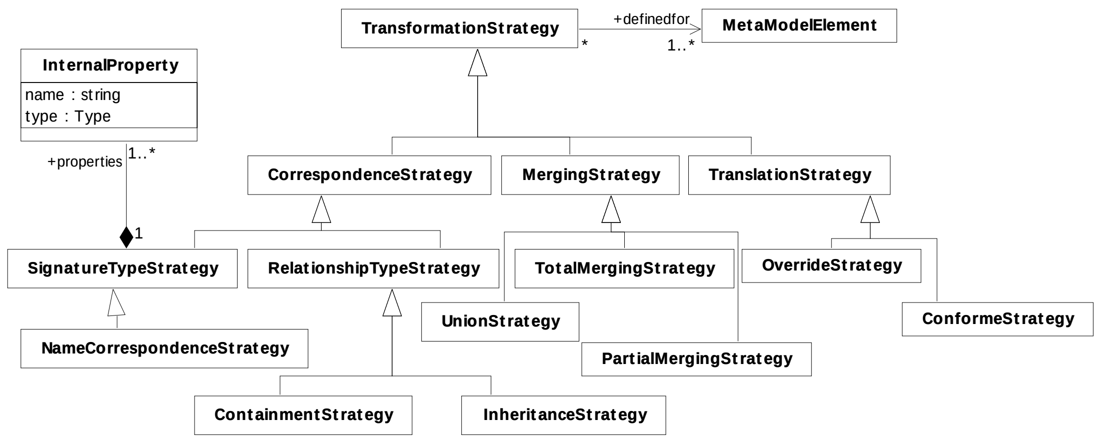

Figure 11 Transformation strategies metamodel

5 SPECIALIZATION PROCESS OF THE GENERIC FRAMEWORK

This section describes how the generic composition framework can be specialized to create a particular composition operator for a specific modeling language. This specialization can be divided into four steps (Figure 12): (1) Specialization of the Correspondence metamodel, (2) Definition of a strategy model, (3) Definition of a transformation rules model, (4) Generation of composition-oriented transformations.

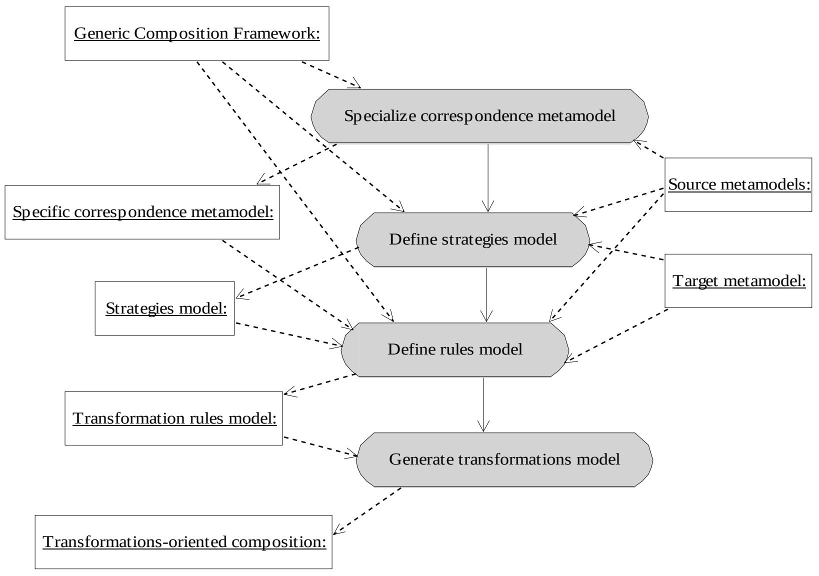
Generic Composition Framework:
Specialize correspondence metamodel
Source metamodels:
Specific correspondence metamodel:
Define strategies model
Target metamodel:
Strategies model:
Define rules model
Transformation rules model:
Generate transformations model
Transformations-oriented composition:

Figure 12. Specialization process of the generic framework

Correspondence metamodel specialization

The first step of the specialization process consists of extending the generic correspondence metamodel for a specific application domain (Figure 13). It is necessary to establish different kinds of relationships between metamodel elements according to their semantics, this task is not trivial, because it requires a depth knowledge of the underlying application domain [DelFabro 06]. For example, if we consider UML2 as a source language, to express the similarity between *class* elements, we define a new type of relationship called *ClassSimilarityRelationship*. The set of specific correspondence relationships constitutes the specific correspondence metamodel.

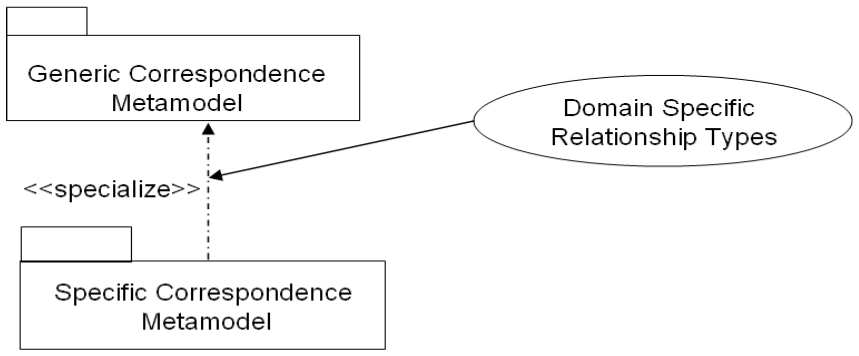
Generic Correspondence Metamodel
Domain Specific Relationship Types
<<specialize>>
Specific Correspondence Metamodel

Figure 13. Specialization of the generic correspondence metamodel

Strategies model definition

In this step, the goal is to define specific strategies for the composition of elements. Correspondence strategies specify comparison semantics between model elements; they specify the semantics of specific correspondence rules. Correspondence strategies are designed by a model conform to the transformation strategies metamodel (Figure 11). As for correspondence rules, specific merging and translation rules must be augmented with suitable transformation strategies. Merging strategies define how elements being related by a specific correspondence relationship are merged to create elements of the target metamodel. Translation strategies specify how elements, which have no corresponding elements in the opposite model, are transformed into the target model. By default, such elements are deeply copied into the target model. This default translation strategy may be overridden in order to cover specific cases.

Transformation rules model definition

This step aims to define a specific transformation rules model conform to the transformation rules metamodel (Figure 10). First, the *CorrespondenceRule* metaclass is instantiated. This instance has a name, it can be specified as concrete or abstract, and it is specified by a set of source and target patterns. The source patterns types are instantiated by elements of the source metamodels, and the target pattern type is instantiated by a specific correspondence relationship. Then we complete the specification of this rule by specifying an appropriate correspondence strategy. After, we specify a set of merging and translation rules according to the target metamodel. The *MergingRule* metaclass defined in the transformation rules metamodel is instantiated by specifying its source and target patterns. In this case, the source pattern has as type a specific correspondence relationship. Finally, for the metaclass *TranslationRule*, one has to define elements of source and target metamodels associated to the patterns.

The specific transformation model is then enriched with the composition strategies model defined in the previous step. This allows the integration of the composition capabilities specified by these strategies. The result is a specific transformation rules model. This model can be seen as a high-level specification of transformations rules, and can be used to automatically generate executable transformations for composition.

Generation of composition-oriented transformations

The last step of the specialization process consists of producing executable transformations implementing the composition operation. These executable transformations are obtained by transforming the specific transformation rules model. This task is carried out by a particular type of transformation called Higher-Order Transformation (HOT) [Del Fabro06] because it generates a transformation. A HOT transformation takes as input a transformation model and produces as output a transformation model which is conform to a transformation language metamodel (Figure 14). More precisely, elements of the composition model are transformed into specific composition code patterns. For example, if a merging rule uses the *TotalMergingStrategy*, the code pattern produced must implement the functionality of combining source elements into a single output element.

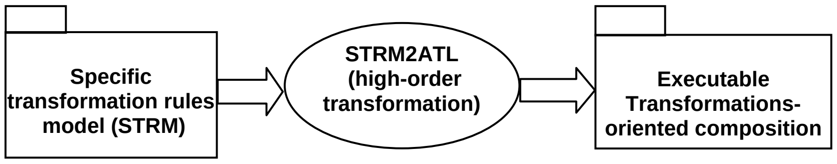
Specific transformation rules model (STRM)
STRM2ATL (high-order transformation)
Executable Transformations-oriented composition

Figure 14. Generation of executable transformations

6 EXAMPLE

To illustrate our approach, we describe in this section how the generic composition framework can be applied to compose models that conform to the UML metamodel. Let us consider the SMFMS example presented in the case study (section 2). In this example, we merge two class diagrams developed independently according to two viewpoints. The result of the composition is a VUML model conform to the VUML profile (Figure 15).

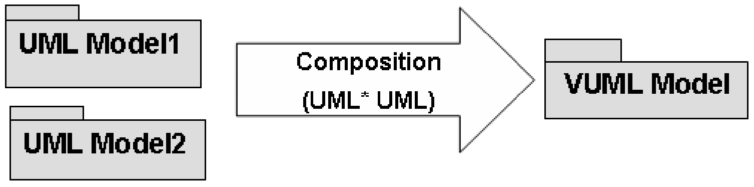

Figure 15. Composition scenario for the SMFMS application

Correspondence relationships specification

To define specific correspondence relationships for UML class diagram elements, we have specialized the generic correspondence metamodel according to the process discussed above. For example, for elements *Class* we have identified two correspondence relationships : ClassConformityRelationship and ClassSimilarityRelationship. We have also defined correspondence relationships for other elements such as attributes, operations, etc.

Transformation strategies specification

To define the composition of UML models producing a VUML model, it is firstly necessary to define correspondence strategies. These strategies specify the semantics of correspondences rules. A default correspondence strategy applied on elements whose metaclass is defined as a sub-class of NamedElement is based on the property name. This operation is strongly dependent on the modeling language. To this aim, for each application scenario, a specific correspondence strategies must be defined.

In our application, we have defined two correspondence strategies between elements Class : conformity and similarity. Conformity holds when two classes appear in two viewpoint models with the same name and the same properties (attributes, operations, associations) ; they are semantically equivalent (represent two views of the same concept in VUML terminology). Similarity holds when two classes appear with the same name but are not conform.

In the same way, merging strategies define the semantics of merging elements according to the correspondence relationship which relate them. For example, we define the TotalMergingStrategy as a merging strategy of two classes related by a ClassConformityRelationship. It describes how to merge source classes in order to create one target class. Whereas the merging strategy of similar classes specifies the creation of a multiview class (base and views).

Transformation rules specification

We have defined a set of transformation rules to compose UML class diagrams into a VUML class model. The transformation rules are reused from the generic framework. According to the specialization process detailed previously, we distinguish the correspondence rules that apply to elements of the source models and create specific relationships, from rules for merging corresponding elements or translating elements. Indeed, all the rules from the first category are derived according to UML metamodel elements. Transformation rules of the second category are derived from both source and target metamodels ; they specialize the generic framework to define a specific merging operator for structural UML models. Figure 16 shows a hierarchy of correspondence rules defined for this example. Some rules are defined as sub-rules of the *ModelElementCorrespondenceRule*. This permits to factorize a part of the common code, providing more reusability and flexibility for the correspondence operator.

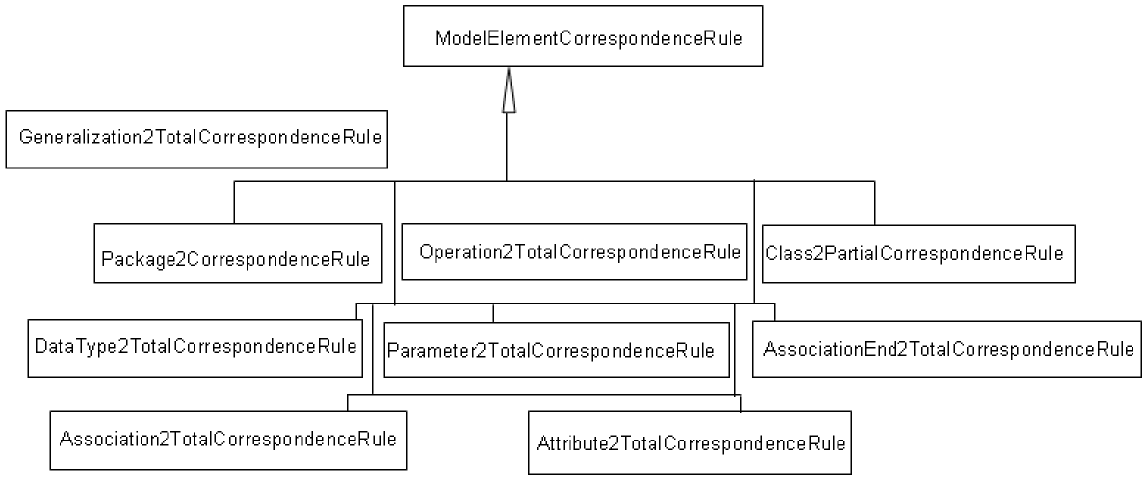
ModelElementCorrespondenceRule
Generalization2TotalCorrespondenceRule
Package2CorrespondenceRule
Operation2TotalCorrespondenceRule
Class2PartialCorrespondenceRule
DataType2TotalCorrespondenceRule
Parameter2TotalCorrespondenceRule
AssociationEnd2TotalCorrespondenceRule
Association2TotalCorrespondenceRule
Attribute2TotalCorrespondenceRule

Figure 16. Correspondence rules for UML class diagram elements

Implementation in ATL

To implement and validate our approach, we needed a rule-based transformation language. We wanted to focus on declarative approaches allowing defining transformations at a high level of abstraction without worrying about the final execution and the rules enforcement. Management of rules enforcement may be delegated to a rules engine, but it is often necessary to help this engine by introducing imperative code depending on the complexity of the transformation. For this aim, we chose ATL [Jouault05] which is a hybrid transformation language allowing to combine both declarative and imperative approaches. ATL is considered as a standard component of Eclipse for model transformation and is now integrated into the M2M project [Eclipse07]. Figure 17 illustrates an example in ATL of transformation rules used for the VUML composition. Note that composition has been implemented with two transformation modules (Lines 1-2) and (Lines 10-12) : the first module implements correspondence rules and generates a correspondence model, whereas the second implements both merging and translation rules and produces the VUML target model. Due to space constraints, only a subset of rules is discussed below.

```
1 module UML2Corresp;
2 create MC : MMC  from MPV1 : UML2, MPV2 : UML2;
3rule Class2PartialCorrespondence extends
4       ModelElementCorrespondenceRule{
5  from e1 : UML2!Class, e2 : UML2!Class(e1.name = e2.name)
6  to    r : MMC! PartialCorrespondenceR (
7              correspondingElementType <-'Class'
8       )
9  }
```

```
10 module CorrespUML2VUML;
11 create VUML : UML2 from  MPV1 : UML2, MPV2 : UML2, PRO :
12 UML2, MC : MMC;
13rule PartialCorrespondence2Base{-- specefic to VUML
14 from r : MMC! PartialCorrespondenceR
15 to c : UML2!Class(
16       name <- thisModule.getRefElement(r).name,
17       visibility<-
18       thisModule.getRefElement(r).getElement.visibility,
19       isAbstract<-
20       thisModule.getRefElement(r).getElement.isAbstract
21            )
22 do{
23       thisModule.VUMLModel.packagedElement<-c;
24       c.applyStereotype(thisModule.base);
25            for (iterator in r.getListeAttrEquality){
26            c.ownedAttribute <-
27       thisModule.AttributeEquality2Attribute(iterator);
28        }
29        for (iterator in r.getListeOpEquivalence){
30            c.ownedOperation <-
31       thisModule.OperationEquivalence2Operation(iterator);
32        }
33       if (not r.general.oclIsUndefined()){
34        c.generalization<-
35                thisModule.resolveTemp(r.general,'g');
36       }
37 }}
38 rule Class2Class{
39  from c1: UML2!Class(c1.isNotMultiviewClass)
40  to c2: UML2!Class(
41 name <- c1.name,
42 isAbstract <- c1.isAbstract,
43 visibility <- c1.visibility,
44 ownedAttribute<-c1.ownedAttribute,
45 ownedOperation<-c1.ownedOperation
46 )}
```

Figure 17. UML2VUML transformation in ATL.

The rule *Class2PartialCorrespondence* (Lines 3-9) states that two class elements will be linked by a *PartialCorrespondenceRelationship* if they are defined in two different models with the same name. This rule is declared as a specialization (keyword *extend*) of *ModelElementCorrespondenceRule* which is defined as an abstract rule. The inheritance mechanism allows to factorize common code among several transformation rules.

The rule *PartialCorrespondence2Base* (Lines 13-37) specifies that for each defined *PartialCorrespondenceRelationship* wich relates two classes, an UML2 element *Class* is created in the composed model. This rule implements the *TotalMergingStrategy*. Finally, the rule *Class2Class* (Lines 38-46) implements the default translation strategy which expresses that a class having no corresponding class in the opposite is translated (copied) as it is in the target model.

7 RELATED WORK

Many researchers have developed model composition approaches in different application domains: requirements engineering [Nuseibeh03] [Chitchyan07], design by aspects [Baniassad04] [France04], development by subjects [Clarke02] [Ossher96], software merging [Mens02], composition of concerns in software architectures [Barais05]. We focus in this section on works that are close to ours, including two important approaches in the field of model composition: EML [Kolovos06] and AMW [Del Fabro05].

In VUML, the first phase of requirements analysis produces a system model according to several actors' viewpoint. Our view-based approach is very close to that of requirements engineering. One of the most fundamental contributions of viewpoints in requirements engineering is the possibility to take several (possibly contradictory) concerns into account. Finkelstein [Fienkelstein90], Nuseibeh, [Nuseibeh03], Easterbrook [Easterbrook95] or Jakson [Jackson01] consider that viewpoints can be expressed in different formalisms and then be linked together in order to obtain a global coherent system. For our part, we adopted the principle that UML can be a unifying formalism to model the various viewpoints; we propose a UML profile to describe a shared composed model rather than connected models. VUML is a fine-grained modeling language that allows to integrate viewpoints at the very core of the system model (class diagram level).

The Atlas Model Weaver [Del Fabro05] is a model composition framework that uses model weaving and transformation to define and execute the composition operation. The tool support is available as an Eclipse plug-in. The composition operation is divided into two phases. The first phase builds a weaving model that captures links between input models according to a weaving metamodel. The second phase uses the weaving model to generate a transformation to produce the composed model. This technique is generic and flexible thanks to the extension mechanism of the weaving metamodel; however, manual definition of links between model elements is a tedious work.

The Epsilon Merging Language (EML) proposed by [Kolovos06] is a rule based language for merging models. EML belongs to the Epsilon platform which is a model driven framework for developing integrated languages for model management tasks such as comparison, transformation, validation, etc. Close to our work, this approach proposes to merge models trough three categories of rules: MatchRule, MergeRule and TransformRule. Our correspondence rules produce a set of links between model elements, whereas in the EML approach this information is stored at run time in a temporary memory called 'MatchTrace'. We consider that it is more convenient to separate the comparison and the merging steps in an objective of reuse. So we generate a correspondence model based on comparisons which is exploited during the merging step. This correspondence model may be used for other aims such as management of dependencies between views in order to ensure the system consistency.

The composition of design models in aspect oriented approaches has also been much studied. In [Reddy06], the proposed technique allows to manage the composition of a primary model with transversal aspects (persistency, security, etc.). This approach uses an algorithm of composition and a set of elementary actions called directives. These directives are applied either on the elements of the models — for instance creation/deletion, modification — or on aspect models by specifying for example the order in which two aspect models must be composed. A metamodel of composition extends that of UML by adding a specification of the composition's behavior and includes meta-operations implementing a signature-based comparison between elements. The use of these directives is a way to solve the conflicts, which may appear during the composition phase. It allows to partially automate an activity which was often manually done by the designer. However, this approach is not compatible with a system based on declarative rules such as ours.

The work presented in [Baudry05] discusses some similarities between model composition and model transformation. Comparison criteria between approaches are based on the degree of generality, ease of use and ease of implementation. The authors explore the possibility of composing a set of models based on crosscutting concerns (aspects), with a primary base model. By varying the level of knowledge about aspect models, signatures and bindings, a number of composition-oriented transformations have been identified. In the continuity of this work, the model composition approach presented in [Fleurey07] offers a generic framework that is independent from any modeling language. The authors propose a metamodel describing structural and behavioral feature of a composition operator. This metamodel supports the composition directives concept introduced in [Reddy06]. These directives are specified in a domain-independent language and implemented in Kermeta [Muller05]. This approach can be said imperative because it describes the operation of composition in an algorithmic way. So it is not easily compatible with our approach which is mainly based on declarative rules that specify what should be transformed rather than how it should be done.

8 DISCUSSION

The work presented in this paper has naturally some limits and raises certain questions that are discussed below.

Inconsistencies and conflicts Resolution. In our view-based system analysis, conflicts among viewpoint models may appear due to separate designs. We have identified such inconsistencies that may be syntactic (e.g. homonymy conflicts, structural conflicts) or semantic (e.g. synonymy conflicts). Management of these conflicts is not easy at the composition stage and hence is out of the scope of this paper. As future work, an interesting track search would consist in using ontologies (see [Dhamanka04][Aumueller05]) to build a shared repository and thus to solve some semantic conflicts according to reconciling policies.

Taking into account semantic aspects in source models. So far, source model elements comparison is based on syntactic properties defined of the source metamodel. For example, two operations having the same signature (name, parameters and return type) are considered as equivalent that is obviously not always the case. To solve this problem one may either perform a semi-automatic reconciliation step among designers, or reinforce the semantics (especially behavioral aspects) associated to the source metamodel so as to elaborate finer comparison strategies. We are working on both tracks.

Rules implementation language. To validate our work, we have chosen ATL so as to gain advantage of a semi-declarative approach. This choice seems accurate for a case study but may present some scalability limits with large models and many rules. Our goal is to experiment imperative transformation-based languages such as Kermeta.

Exploitation of the composed VUML model. The VUML model resulting from the composition process may be used in several ways. A classical way consists in producing object code via a code generator. We have developed a generic code generator and a first instance of this generator targeting Java [Nassar09]. Other languages such as C++ or Eiffel could be supported. Besides, if one needs to extend a given design in VUML so as to integrate a new viewpoint, the VUML model may be composed in turn with a new viewpoint model produced separately. To do that, one may apply our approach by replacing the source UML metamodel by the VUML metamodel. Our MDE-based approach is well adapted to such evolution but we have now to implement it.

9 CONCLUSION AND FUTURE WORKS

In this paper, we have presented a Model Driven based approach for model composition. This approach combines the use of metamodeling and model transformation techniques. The originality of this paper is to propose a composition process made of two sub-processes: a generic process for the definition of a model composition operator, and a specialization process to apply this generic operator to specific modeling domains.

The core composition operator is generic since it allows to design composition requirements at a high level of abstraction. We have defined a core metamodel that is composed of three separate metamodels: (1) a relationship metamodel that contains abstract constructs defining the correspondences between model elements, (2) a rules metamodel that contains generic transformation rules used for the implementation of the composition operator, and (3) a strategy metamodel that defines common transformation strategies. To validate our approach, we have implemented transformation rules in ATL and we are working on the development of a model composition environment implemented as an Eclipse plug-in.

Some of our future work is already discussed in Section 8 above. We also intend to automate the weaving activity of the specialization process. To do that, we will apply the HOT technique to generate the executable transformation rules in a specific transformation language. Therefore, the manual programming step of transformation will be dramatically reduced. Another interesting issue is to compose behavioral diagrams of

UML (mainly state charts or sequence diagrams). This work is the subject of a PHD thesis in progress in our team. First results on this topic are described in [Ober08]

ACKNOWLEDGMENTS

This work was supported by the COMPUS Project MA/06/152, VOLUBILIS 2006.

REFERENCES

[Nassar09] Nassar M., Anwar, A., Ebersold, S., El Asri, B., Coulette, B., Kriouile, A. Code Generation in VUML profile: a Model Driven Approach. IEEE/ACS AICCSA 2009. Rabat, May 10-13, 2009. IEEE Computer Society Press.

[Anwar08a] Anwar, A., Nassar, M., Ebersold, S., Coulette, B., Kriouile, A. A QVT-Based Approach For Model Composition: Application to the VUML Profile. ICEIS 2008, pp 360-367. Spain, Juin 2008.

[Anwar08b] Anwar, A.., Ebersold, S., Nassar, M., Coulette, B., Kriouile, A.. Towards a generic approach for model composition. ICSEA 2008, IEEE Computer Society press, pp 83-90. Malte, 2008.

[Aumueller05] Aumueller, D., Do, H H, Massmann, S, Rahm, E. Schema and ontology matching with COMA++. SIGMOD 2005. pp 906-908.

[Baniassad04] Baniassad, E., Clarke, S. Theme: An approach for aspect-oriented analysis and design. ICSE’04. (2004), pp 158 -167.

[Batini86] Batini, C., Lenzerini, M. and Navathe, S. B. A Comparative Analysis of Methodologies for Database schema Integration. ACM Computing Surveys, Vol. 18, No. 4, Dec. 1986.

[Baudry05] Baudry, B., Fleurey, F., France, R., Reddy, R., Exploring Relationship between Model Composition and Model transformation. Aspect Oriented Modeling Workshop. MODELS'05. Montego Bay, Jamaica, October 2005

[Bézivin06] Bézivin J., Bouzitouna S., Del Fabro M.D., Gervais M., Jouault F., Kolovos D., Kurtev I., Paige R. A Canonical Scheme for Model Composition. ECMDA-FA, LNCS 4066, Springer-Verlag, 2006, p. 346-360

[Bézivin05] Bézivin J., Jouault F. Using ATL for Checking Models. Intl Workshop on Graph and Model Transformation (GraMoT), Tallinn, Estonia. 2005.

[Chitchyan07] Chitchyan R., Rashid A., Rayson P., Waters R. Semantics-based Composition for Aspect-Oriented Requirements Engineering. AOSD 07, pp 36- 48. March 12-16, 2007, Vancouver Canada.

[Clarke02] Clarke, S.: Extending Standard UML with Model Composition Semantics. Science of Computer Programming, 44 (2002) 71.100

[Cousot90] Cousot P. Methods and Logics for Proving Programs. Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B), p. 841-994, 1990.

[Dhamanka04] Dhamanka, R, Lee Y, Doan, A, Halevy, A, Domingos P. iMAP: Discovering Complex Semantic Matches between Database Schemas. SIGMOD 2004, pp 383-394.

[Del Fabro05] Didonet Del Fabro M., Bézivin J., Jouault F., Breton E., Gueltas G. AMW: a generic model weaver. IDM'05. Paris, France, juin 2005, p. 105-114.

[Del Fabro06] Didonet Del Fabro, M, Bézivin, J, and Valduriez, P : Model-driven Tool Interoperability: an Application in Bug Tracking. In: ODBASE'06 (OTM Federated Conferences). Montpellier, France. 2006.

[Easterbrook95] Easterbrook, S. M., Nuseibeh, B. Managing inconsistencies in an evolving specification. International symposium on Requirements Engineering, pp 48-55, York, England. IEEE Computer Society, March 1995.

[Eclipse07] Eclipse/M2M Project Web Page. http://www.eclipse.org/m2m/, 2007.

[Finkelstein90] Finkelstein, A., Kramer, J., Goedicke, M. Viewpoint Oriented Software Development. ICSSEA. Toulouse, France, pages 337-351, 1990.

[Fleurey07] Fleurey F., Baudrey B., France R., Ghosh S. A Generic Approach for Automatic Model Composition. Aspect Oriented Modeling Workshop, MODELS 2007, Nashville USA 2007.

[France04] France, R.B., Ray, I., Georg, G., Ghosh, S. An aspect-oriented approach to design modeling. IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design 151 (2004) 173.185.

[Jackson01] Jackson, M. Problem Frames. Addison-Wesley, 2001.

[Jouault05] Jouault F., Kurtev I. Transforming Models with ATL. Model Transformations in Practice Workshop, MODELS 2005, Montego Bay, Jamaica 2005.

[Kiczales97] Kiczales, G., Lampng, J., Mendhekar, A., Maeda, C., Videira, L.C. Aspect-Oriented Programming. ECOOP'97. Springer-Verlag LNCS 1241. Finland, 1997.

[Kleinner07] Kleinner S. F. Oliveira and Toacy Cavalcante de Oliveira. A Guidance for Model Composition. ICSEA 2007. Cap Esterel, France. 2007.

[Kolovos06] Kolovos, DS., Paige, RF., Polack, FAC. Merging Models with the Epsilon Merging Language (EML). MODELS 2006, Genova, Italy, October, 2006.

[Kruchten99] Kruchten Philippe, Rational Unified Process - An Introduction, Addison-Wesley, 1999.

[Mens02] Mens, T. A state-of-the-art survey on software merging. Transactions on Software Engineering, vol. 28, no 5, May 2002.

[Muller05] Muller P.-A, Fleurey, F. and Jézéquel, J.M. Weaving executability into object-oriented meta-languages. MODELS'05, p. 264 - 278. Montego Bay, Jamaica, October 2005.

[Nassar03] Nassar, M., Coulette, B., Crégut, X., Ebersold, S and Kriouile., A. Towards a View based Unified Modeling Language. ICEIS'03, Angers, France, 2003.

[Nuseibeh03] Nuseibeh, B., Finkelstein A., and Kramer, J. ViewPoints: meaningful relationships are difficult. ICSE 2003, Portland, Oregon, 2003.

[Ober08] Ober, I., Coulette, B., Lakhrissi, Y. Behavioral Modelling and Composition of Object Slices Using Event Observation. MODELS 2008, Toulouse, 28/09/2008-03/10/2008, Springer, LNCS 5301, p. 219-233, september 2008.

[OMG02] OMG 2002, OMG/MOF Meta Object Facility (MOF) 1.4. Final Adopted Specification Document. Formal/02-04-03, 2002.

[OMG03a] OMG 2003, UML 2.0 Superstructure Final Adopted specification, Document-ptc/03-08-02.

[OMG03b] OMG 2003, UML 2 OCL Final Adopted Specification, 2003. http://www.omg.org/docs/ptc/03-10-14.pdf.

[Ossher96] Ossher, H., Kaplan, M., Katz, A., Harrison, W., Kruskal, V. Specifying subject-oriented composition. Theory and Practice of Object Systems, Wiley & Sons 2 (1996).

[Reddy06] Reddy Y. R., Ghosh S., France R. B., Straw G., Bieman J. M., McEachen N., Song E., Georg G. Directives for Composing Aspect-Oriented Design Class Models. Transactions of Aspect-Oriented Software Development, Vol.1, No. 1, LNCS 3880, p75-105, 2006, Springer.

[Sabetzadeh05] Sabetzadeh M and S. Easterbrook. An Algebraic Framework for Merging Incomplete and Inconsistent Views. 13th IEEE International Requirements Engineering Conference, September 2005.

[Soley00] Soley et al. MDA Model Driven Architecture. Richard Soley and the OMG Staff Strategy Group, Object Management Group White Paper, Draft 3.2 – Nov. 2000.

About the authors

Adil Anwar works as a contractual teacher in computer science at the University of Toulouse, and as a member of the MACAO team of IRIT laboratory. In 2009, he received a Ph.D degree in Computer Science at the University of Toulouse. He works on model-driven engineering, mainly exploring the relationship between model composition and model transformation. He can be reached at anwar@univ-tlse2.fr

Sophie Ebersold works as a lecturer at the University of Toulouse, and as a member of the MACAO team of IRIT laboratory. Her research fields of interest are mainly integration of viewpoints in Object-Oriented Analysis/Design and more generally MDE. She has directed one PHD thesis in the context of an international collaboration with Morocco, and several master thesis. She can be reached at ebersold@univ-tlse2.fr

Bernard Coulette works as a full professor at the University of Toulouse, and as a member of the MACAO team of IRIT laboratory. His research fields of interest are mainly integration of viewpoints in Object-Oriented Analysis/Design (VUML profile), modeling and enactment of Model Driven Processes. He has directed several PHD thesis in the context of international collaborations (Vietnam, Morocco). He can be reached at coulette@univ-tlse2.fr

Mahmoud Nassar is Professor and Head of the Software Engineering Department of ENSIAS Engineering school of Rabat, and member of SI2M laboratory. He received his Ph.D in Computer Science in 2005 from the INPT Institute of Toulouse. His research interests are integration of viewpoints in Object-Oriented Analysis/Design (VUML profile), Service-Oriented Computing, Model-Driven Engineering. He can be reached at nassar@ensias.ma

Abdelaziz Kriouile works as a full Professor in the Software Engineering Department of ENSIAS Engineering school of Rabat, and member of SI2M laboratory. His research interests include integration of viewpoints in Object-Oriented Analysis/Design, Service-Oriented Computing, and speech recognition. He has directed several Ph.D thesis in the context of French-Moroccan collaborations. He can be reached at kriouile@ensias.ma