# Exploring the Relationship between Model Composition and Model Transformation

Benoit Baudry, Franck Fleurey
IRISA – Campus de Beaulieu
35042 Rennes Cedex, France
{bbaudry, ffleurey}@irisa.fr

Robert France, Raghu Reddy
Colorado State University
601 Howes Street, Fort Collins, CO, USA
{france, raghu}@cs.colostate.edu

## 1 Introduction

In aspect-oriented modeling (AOM), a design is presented in terms of multiple user-defined views (aspects) and model composition is often carried out to obtain a model that provides an integrated view of the design. Typically, model composition involves merging two or more models to obtain a single model. Model composition and model transformations that incorporate new features into a model seem to serve the same purpose: In model composition, the new features that are to be incorporated into a model are explicitly described by one or more source models; in a transformation, the new features are implicitly defined in the transformation actions that are carried out on the input model.

The apparent similarities between model composition and model transformations often lead to the following question: Is model composition a special type of model transformation? Answering this question can lead to useful insights that can be used to develop technologies that leverage the relationship between composition and transformation. In this paper we show that there are a number of ways to implement composition as transformations. We give an overview of these approaches in terms of their generality, ease of use, and ease of implementation. The insights gained from our initial analysis suggest that one can implement model composition as model transformations. This is important in that it indicates that research on model composition can leverage research on model transformations.

## 2 Composition in Aspect-Oriented Modelling

The UML provides support for separating a fixed set of concerns during modelling through the use of model views, for example, design class models are used to describe designs from a static structural view and sequence models are used to describe a design from an interaction viewpoint. AOM techniques provide mechanisms that allow users to present designs in terms of a variety of non-orthogonal user-defined views (aspects), for example, security, error-recovery, and other views that show how dependability objectives are realized in a design [1]. In the Aspect-Oriented Model-Driven Framework (AOMDF) developed at Colorado State University (see [2], [3]) a design consists of a primary model and one or more aspect models [2]. A primary model describes the core functional structure of an application, and each aspect model is a description of a design feature that crosscuts the structure described in the primary model. An aspect model is a view that describes how a design objective is realized in a design. Aspect and primary models are composed to obtain a design model that integrates the views described by the primary and aspect models.

Aspect models are pattern descriptions of crosscutting features. The patterns are described using UML template diagrams [2]. An aspect model must be instantiated before it can be composed with a primary model. The instantiated forms of aspect models are UML models that are referred to as *context-specific aspects*. The instantiation of an aspect model is determined by *bindings*, where a binding associates an application-specific value with a template parameter. In this respect, composition of aspect and primary models can be viewed as a process in which the patterns described by aspect models are incorporated into a primary model. This process is sometimes referred to as *model refactoring*.

Composition of primary and context-specific aspect models involves merging model elements that represent different views of the same concept. In order to automate model composition, a syntactic approach to identifying matching model elements (i.e., elements that describe views of the same concept) across primary and aspect models is needed. Model element signatures are used in the AOMDF: elements with the same signature are viewed as representations of the same concept by the composition mechanism [4]. A signature is a subset of the syntactic properties associated with a model element in the UML metamodel. For example, if the signature of a class consists only of its name, then the AOMDF composition mechanism will merge all classes with the same name. In some cases, signatures may fail to identify similar concepts or may identify two elements as representing the same concept when they really represent different concepts. For this reason, it is necessary to analyze the automatically produced composed model to uncover such problems as well as unanticipated emergent properties that arise as a result of interactions across the primary and aspect views. In some cases, these problems can be resolved by transforming the aspect and primary views during composition using composition directives [2], [4]. For example, a composition directive that modifies the properties of a model

element so that it matches with another model element can be used in situations where the elements represent the same concept but their signatures in the original models do not match.

# 3 Transformation-Oriented Views of Composition

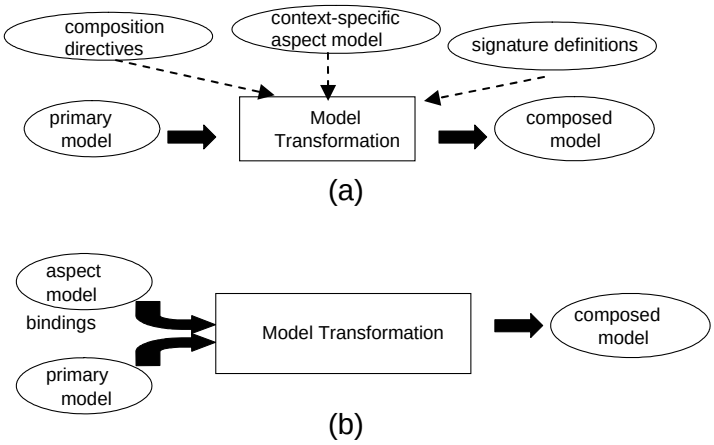


**Figure 1 - two transformation-oriented composition approaches**

Figure 1 illustrates two extreme transformation-oriented approaches to model composition. In Figure 1(a), model composition is implemented as a model transformation that transforms a primary model into a target model (a composed model) by incorporating features described by a context-specific aspect model. In this case, the context-specific aspect model specifies the information the transformation must incorporate into the primary model. The transformation in this approach is aspect-, directive-, and signature-specific: It is dedicated to the composition of a particular aspect model and a primary model using a specific set of signatures and composition directives.

In Figure 1 (b) the transformation takes a primary model and an aspect model with a set of bindings as input and produces a composed model as its output. Unlike the first approach, knowledge about the aspect model, primary model, composition directives, bindings and the signatures are not embedded in the transformation actions. This general-purpose transformation would be extremely difficult (if not impossible) to implement: The transformation would have to cater for all possible varieties of aspect and primary models and their composition peculiarities.

## *3.1 A Spectrum of Transformation-Oriented Composition Approaches*

In between the two extremes shown in Figure 1 are transformation-oriented composition approaches in which varying levels of knowledge about aspect models, signatures and bindings are embedded in the transformations. Figure 2 illustrates a possible spectrum of transformation-oriented approaches to model composition. The dedicated transformation view illustrated in Figure 1(a) is at the top of the spectrum and the more general transformation view illustrated in Figure 1(b) is at the bottom. It is interesting to examine the approaches in the spectrum with respect to the generality of the transformations, and the ease of creating and using the transformations.

The dedicated transformation shown in Figure 1(a) can be made more general by capturing the variable parts of a composition (e.g., the bindings used to instantiate an aspect model and the signatures used to determine matching elements) in transformation parameters. For example, in the second view from the top in Figure 2, knowledge about the pattern described by an aspect model is embedded in a transformation, but information about signatures and bindings are passed in as parameters to the transformations. The parameterized transformations can be used for different instantiations of an aspect model and for different sets of signatures. In this case, developers do not have to craft a dedicated transformation for composing a context-specific aspect using a particular signature. On the other hand, using this transformation requires more effort because more parameters must be passed into the transformation. Creating and using a more generic transformation can require more effort than creating a dedicated transformation, but the payoff is that the same generic transformation can be applied to a wider range of models.

The transformations at the top of the spectrum can be expressed as a series of relatively simple transformation actions. As one progresses down the spectrum, the transformation algorithms become more general and thus more complex. The transformations near the bottom of the spectrum are best implemented as frameworks that can be specialized to obtain a specific transformation algorithm. As an example, consider the approach presented

just above the idealistic, but impractical transformation approach shown at the bottom of Figure 2. The transformation in this case is obtained by specializing a framework that captures core composition functionality. For example, in the AOMDF, a composition metamodel defines the framework [4], and the framework is specialized by defining specializations of the framework classes that define the signatures and merge algorithms used to compose the models. Developing such a general mechanism requires more effort, but the mechanism can be used a very wide range of models, and, once the framework is specialized, is easy to use. This approach seems to be the most flexible of the practical approaches in the spectrum.

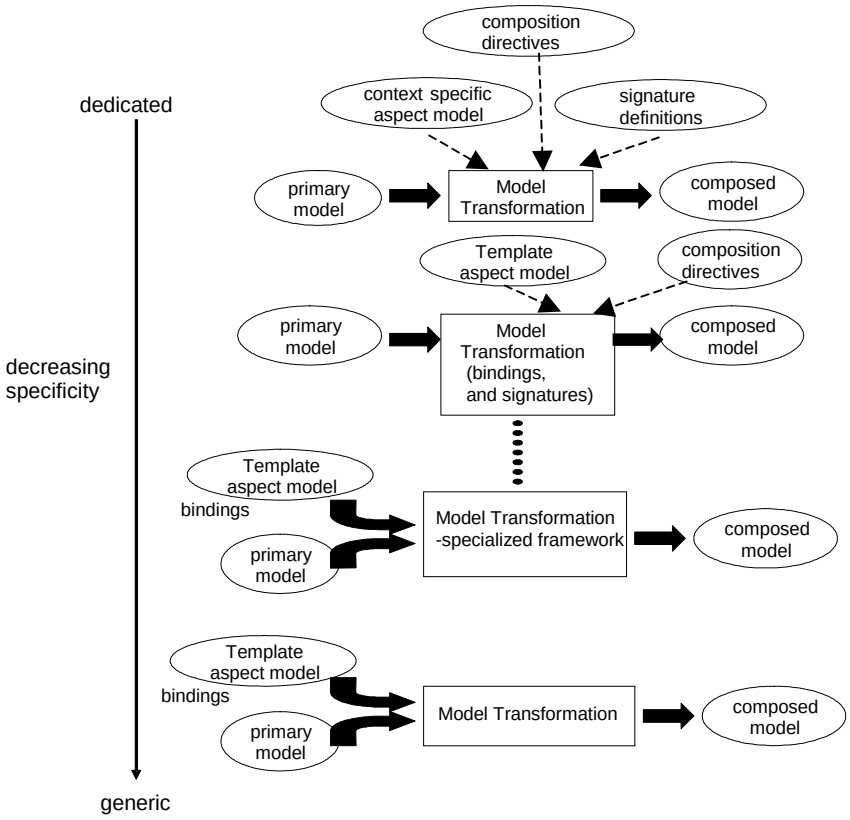


**Figure 2 - Intermediate variants from dedicated to generic composition**

## *3.2 Transformation-Oriented Composition Examples*

In the following we illustrate some of the approaches shown in Figure 2 using a small example taken from [2]. The example is shown in Figure 3. The example uses model element signatures that include only the name of the concepts and composition directives to indicate that certain model elements in the context-specific aspect model will completely replace similarly named elements in the primary model.

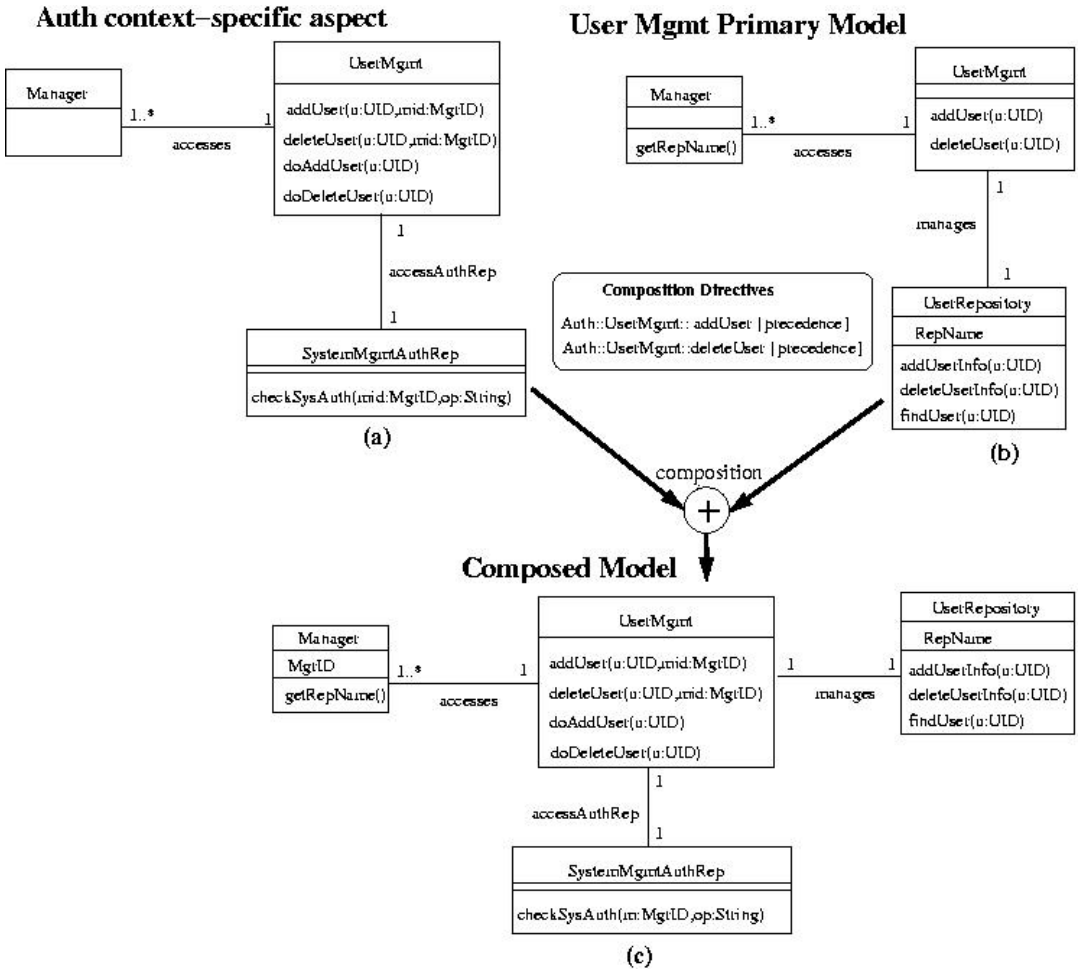


**Figure 3 - A simple model composition example**

The first transformation approach shown at the top of Figure 2 can compose a context-specific aspect model into a primary model. We will use the example in Figure 3 to illustrate how this can be done. In this example, the `User Mgmt` primary model is the source model for the transformation, and the composition directives, name-based signatures and the context-specific aspect model are used to create the transformation actions. The transformation can be expressed in terms of the following sequence of actions (in the following `PM` refers to the primary model)[1]:

1 rename the `addUser` method of the class `UserMgmt` of `PM` to `doAddUser`
2 rename the `deleteUser` method of the class `UserMgmt` of `PM` to `doDeleteUser`
3 add a class `SystemMgmtAuthRep` in `PM`
4 add a 1-1 association between the classes `SystemMgmtAuthRep` and `UserMgmt`

The transformation performs only the actions needed to incorporate the aspect elements into the primary model. Note that name-based matching criteria is implicit in the transformation steps: The transformation assumes there is a `UserMgmt` class in the primary model, and that this class has two methods called `addUser` and `deleteUser`. These assumptions make the transformation very easy to write: There are no specific cases that have to be treated separately, and there is no need to check the presence of a model element. On the other hand, these assumptions make it very difficult to reuse the transformation on other models. As pointed out earlier, this type of transformation is specific to the models being composed and the signatures used to determine matching elements.

The second transformation-oriented composition example uses the `Auth` template aspect model instead of the context-specific aspect model to create a parameterized transformation. Apart a primary model `PM`, it also needs, as input data, the bindings to build the context-specific aspect model. For our example, there are 11 parameters to express the binding: `Client`, `accesses`, `m`, `n`, `p`, `q`, `doOperation[*]`[2], `Server`, `AuthorizationRepository`, `checkAuth`, `mgrid`, `op`, `OpType`.

[1] We use an informal transformation notation.
[2] The * indicates that this parameter must be bound to a collection of values

With these parameters, the transformation can be described as follows:

1 rename the `addUser` method of the class `Server` of `PM` to `doOperation[1]`
2 rename the `deleteUser` method of the class `Server` of `PM` to `doOperation[2]`
3 add a class `AuthorizationRepository` in `PM`
4 add a method `checkAuth` in class `AuthorizationRepository` with parameters `q:mgid` and `op:OpType`
5 add a p-q association between the classes `SystemMgmtAuthRep` and `UserMgmt`

In addition to these transformation rules, the transformation should also process the following bindings: `Client` and `Server` are bound to `Manager` and `UsrMgmt` and `m` and `n` are the cardinalities of the association between these two classes.

There are fewer assumptions about the input primary model than in the previous transformation, since all the names that were built into the previous transformation are now parameterised variables (e.g., `doOperation[1]`) in this transformation. A wider variety of primary models can be passed as input to this transformation. However, note that this transformation is far from being a completely generic transformation since it is still dedicated to one particular aspect. This transformation is also more difficult to write than the previous one: more cases must be taken into account depending on the values of the parameters. These parameters also make this transformation more difficult to use than the previous transformation because a large set of parameters must be passed to the transformation, in addition to the primary model.

The third variant shown in Figure 2 takes a template aspect model, a set of bindings and a primary model as input to a transformation. The transformation is obtained by specializing a transformation framework. By specializing the framework, the user specifies how the elements in the models are to be merged. An example of this type of transformation has been described elsewhere [4]. While this approach can be applied to wider range of models, it is more difficult to create and its use involves specializing a framework. On the other hand, a single specialized transformation may be applicable to a wide range of primary and aspect models, and thus it may not be necessary to specialize the framework for every application.

# 4 Conclusion

The ideas presented in this paper reflect our early thoughts on the relationship between model composition and transformation. We identify several transformation-oriented composition approaches with varying degrees of generality and usability. The ideas presented in this paper may be applicable to other types of transformations, and seem to be consistent with ideas around the application of reflective MDD [5]. It is important to think about the way the transformation is designed (e.g., make it adaptable by using a framework or a library, or make it more dedicated but easier to tune and use), the way it is implemented, and the intent of a transformation (e.g., a transformation used only to compose models). An interesting future direction would be to look at the different types of transformations (e.g., those identified by Czarnecki et al. in [6]) and see in which cases the different variants we have identified for composition can be applied.

Our future work involves exploring how transformation technologies can be used to support model composition. In this respect, this paper provides a preliminary context for this work.

# 5 References


[1] Y. R. Reddy, R. B. France, and G. Georg. An aspect-based approach to modeling and analyzing dependability features. Technical Report CS04 - 109, Colorado State University, November 2004.

[2] R. B. France, I. Ray, G. Georg, and S. Ghosh. An aspect-oriented approach to design modeling. *IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, **151 (4)** : 173–185, August 2004.

[3] D. Simmonds, A. Solberg, R. Reddy, R. France, S. Ghosh. An Aspect Oriented Model Driven Framework. Accepted to Ninth IEEE "The Enterprise Computing Conference" (EDOC 2005), Enschede, Netherlands, 19-23 September, 2005.

[4] G. Straw, S. Ghosh, R. France, J. Bieman, G. Georg, E. Song, N. McEachen. Directives for Composing Models. Technical Report 05-101, Colorado State University, August 2005.

[5] J. Bézivin, N. Farcet, J.-M. Jézéquel, B. Langlois, and D. Pollet. Reflective model driven engineering. In G. Booch P. Stevens, J. Whittle, editor, *Proceedings of UML 2003*, volume 2863 of LNCS, pp 175-189, San Francisco, October 2003.

[6] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. Anaheim, CA, USA. October 2003.