A Generic Approach For Automatic Model Composition

Franck Fleurey[1], Benoit Baudry[1], Robert France[2] and Sudipto Ghosh[2]

[1]IRISA/INRIA – Université de Rennes 1, Campus Universitaire de Beaulieu,
35042 Rennes, France
{bbaudry, ffleurey}@irisa.fr
[2]Colorado State University, Fort Collins, CO, USA
{france, ghosh}@cs.colostate.edu

Abstract. Analyzing and modelling a software system with separate views is a good practice to deal with complexity and maintainability. When adopting such a modular approach for modelling, it is necessary to have the ability to automatically compose models to build a global view of the system. In this paper we propose a generic framework for composition that is independent from a modelling language. We define a process for adapting this framework to particular modelling language (defined with a metamodel) and illustrate how the generic composition has been specialized for class diagrams.

1 Introduction

When using aspect-oriented modelling to build the model for a large system, the modellers identify different views. They model each view and reason about these separate models. They can also perform validation tasks on each model, for example using model checking techniques. Once all these models have been correctly built in isolation, it is necessary to compose these different models. The composed model can be used to

- check the global consistency of the system's model,
- better understand the interactions across the composed views
- analyze interactions to identify conflicts and undesirable emergent behaviours

When the models are small enough and developed by a single or a couple of designers, they can be composed manually. However, in most cases, the models are too large to be composed manually and it is necessary to develop an automatic composition operator to ensure that all the elements in the model are handled. Moreover, an automatic composition operator allows the designer to try different solutions for composition that correspond to different decision (e.g., try different orders for composition, or different solutions for one view…).

There exist several solutions to automatically compose models in different languages (class diagrams [1, 2], statecharts [3], sequence diagrams [4]…). If there is no composition operator for one modelling language, it is necessary to build a completely new operator for this language. In the future, there might be more and more domain-specific modelling language developed for model-driven development. In that case, we do not want to generate a new algorithm and a new environment for

each language. It becomes necessary to reuse algorithmic and design knowledge from composition operators in other languages.

In this paper we propose a generic framework for automatic model composition. There are two main steps for composition: matching and merging. The matching step is specific to a modelling language. It specifies which element in the language can match and how they can match. The merge step can be defined independently from any language. The framework we propose implements a generic merge operator. It defines, how two model elements that match are merged, as well as a mechanism for conflict detection. The framework also implements a language to specify composition directives as they are defined by Reddy et al. [2] and specifies a clear interface to specify a match operator. This framework can then be specialized for a particular metamodel in order to add composition capabilities to a particular language.

In section 2 we discuss the generic framework for model composition. In section 3 we explain how this framework can be specialized to add composition capabilities to a particular metamodel and in section 4 we illustrate this specialization on an example.

2 A generic framework for model composition

This section describes the generic composition algorithm that is implemented in the generic composition framework. The composition mechanism implemented in this framework is structured in two major steps:

1 **Matching**: identifies model elements that describe the same concepts in the different models that have to be composed.

2 **Merging**: matched model elements are merged to create new model elements that represent an integrated view of the concepts.

The merging operator builds a new model from two models. It merges elements that match according to the matching operator and creates new elements in the target composed model. This operator is independent of a specific domain. It consists in going through the set of elements that match in both input models and if they can be merged the operator creates a new model element in the output model. If the elements can not be merged, a conflict is detected that has to be solved. This happens when elements that match based on a subset of their properties (*e.g.*, when merging two class diagrams, classes with same names match) can not be merged because of other properties that do not match (*e.g.*, if there is one concrete class and one abstract class). The whole process of conflict detection and model elements creation is generic.

The semantics of matching is domain specific. The knowledge for detecting model elements that describe the same concept is based on information dependent on the meaning of the model. Thus, the matching operation has to be specialized for each modelling language. However, in order to interact correctly with the merging operator, the matching operator has to have a clear interface.

The generic framework described in this section implements the behaviour of the merging operator and offers a precise interface for the definition of the matching operator. The framework can then be specialized by providing specific matching operators through this interface.

2.1 Generic framework for composition

Figure 1 describes the class diagram of the generic composition framework. It defines the interface of a matching operator for a specific metamodel and it implements the merging operator that is independent of any metamodel.

The abstract operation `getSignature` in MERGEABLE has to be specialized to define the algorithm for matching elements. The `getSignature` operation defines the signature of the model elements. This signature is compared with the signature of other model elements to check if these elements have to be merged. A default comparison is implemented in the `equals` operation of STRINGSIGNATURE. This operation can be specialized in other subclasses of SIGNATURE in order to compare signatures which are more complex than simple strings.

For example, two methods in a class diagram can match because they have the same name or because they have the same name and the same parameters. In the first case, the `getSignature` operation will return the name of the methods and the default `equals` is sufficient. In the second case `getSignature` will return the name and the list of parameters and it is necessary to redefine the `equals` operation.

The `merge` operation in class MERGEABLE implements the generic algorithm for merging two model elements. The complete algorithm is defined in [5]. If two model elements match according to their signature, this operation tries to merge them into a new model elements. The algorithm compares the values of each property of the elements to merge to detect possible conflict. If no conflict is detected the new model element is created, otherwise the conflict must be solved using composition directives.

The class CompositionContext contains the data structures and utility methods that are used by the `merge` operation in order to create the merged elements and keep a traceability information between the input models and the composed model.

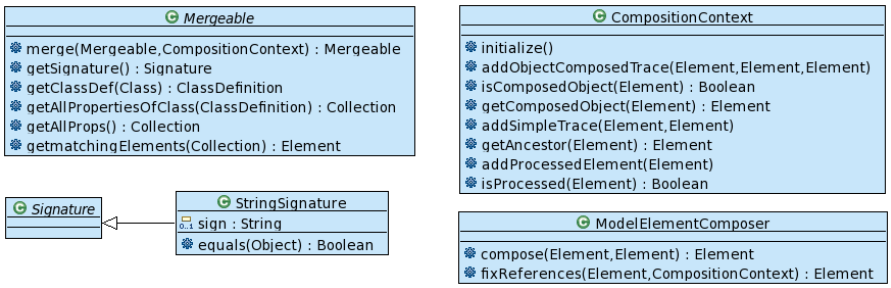

Figure 1 – Generic framework for Composition

2.2 The composition directives

Modellers can specify composition directives that are used during composition to force matches, disallow merges, and to override default merge rules. Two types of composition directives are currently supported in the composition metamodel:

- **Pre-Merge Directives**: These directives specify simple model modifications that are to be made before the models are merged. These changes will force or disallow element matches. These directives can specify renaming model elements, removing or adding elements.
- **Post-Merge Directives**: These directives specify simple modifications to the merged model. For example, it may be the case that a security view requires the removal of associations that are present in other views. This restriction can be specified as post-merge directives that remove these associations from the merged model.

The language for model directives is domain independent and is part of the generic composition framework. The metamodel for this language is shown Figure 2. There are two main types of directives: CREATE and CHANGE directives. CREATE directives are used to create new model elements. CHANGE directives are used to modify model elements. These directives can be used to remove an element from a namespace, set a property value associated with an element, and add an element to a namespace. A CHANGE directive is associated with a reference to the model element it modifies.

A SET directive is associated with two instances of ELEMENTREF one is the target property and the other is the new value for the property. Elements can be reference by (1) a name that is an instance of NAMEREF, (2) their literal value, or (3) a unique identifier that is an instance of IDREF.

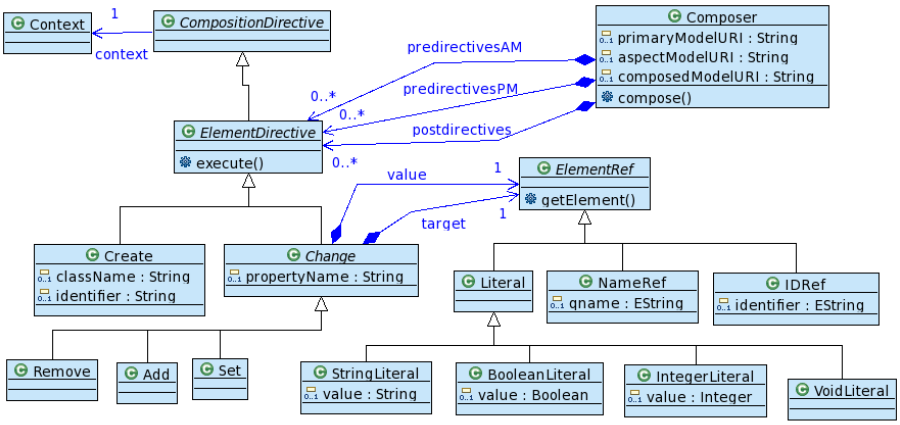

Figure 2 – The composition directives language

3 Specializing the framework for a particular metamodel

This section summarizes how to specialize the generic framework to add composition capabilities to a metamodel. The framework has been defined in such a way that it can be specialized for any metamodel that conforms to EMOF. The specialization process is described in Figure 3.

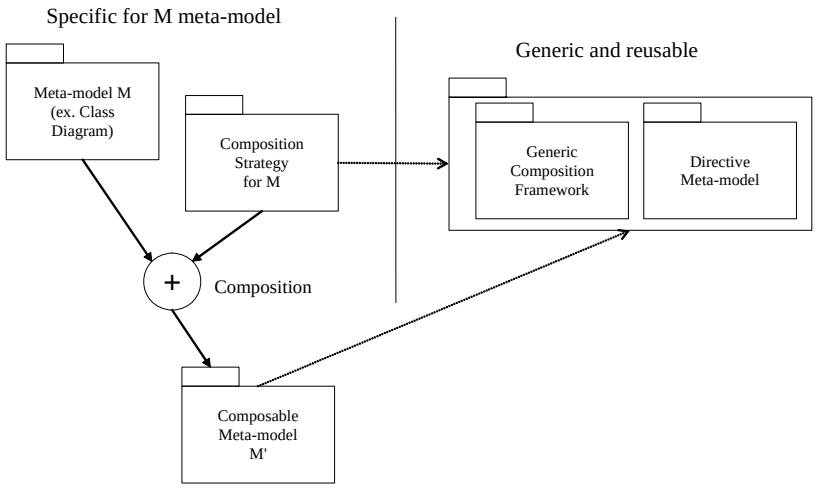

Figure 3 – Adding composition capabilities in a metamodel

To specialize the generic composition framework, for a metamodel M, it is necessary to define a *composition strategy*. This strategy specifies which elements of the metamodel can be merged. This consists in selecting the classes in the metamodel that will have composition capabilities, *i.e.*, the classes which instances will be composable in models that conform to M. The strategy defines the equality between elements, and the signature of these elements.

The composition strategy is a model with major parts: the classes of the metamodel M that are mergeable and how the signatures are computed and the classes that define comparison between signatures. The classes of M that are mergeable inherit from MERGEABLE in the strategy and they implement the `getSignature` operation. The other classes of the strategy specialize the SIGNATURE class of the generic framework. These subclasses specialize the `equals` operation that defines the equality between two model elements. We can notice that the generic framework defines a default behaviour for the `equals` operation. Thus, it is not mandatory to specialize the SIGNATURE class.

Once the strategy is modelled it can be composed with the metamodel M. As a result of the composition, the metamodel M is augmented with the classes and methods that are necessary to provide composition capabilities. Moreover, the resulting metamodel M' inherits form all the operations implemented in the generic framework through inheritance relationships with the MERGEABLE and SIGNATURE classes.

M' is obtained using the composition mechanism of Kermeta[1] [6]. This composition only allows adding operations and classes in a metamodel and ensures that every model that conforms to the original metamodel can be viewed as an

[1] It can be noticed that the composition operation in Kermeta is a specialization of the generic framework presented here..

instance of the composed metamodel. Thus the models that conform to M also conform to M' and can be composed thanks to the capabilities added in M'.

4 An example for composing Ecore models

In this section we illustrate the specialization of the framework to compose models that conform to the Ecore metamodel. Figure 4 shows the Ecore metamodel that is very close to the metamodel for class diagrams. The metamodel defines packages that are composed of classifiers. There are two types of classifiers: classes or data types. The enumeration is the only data type present here. Classes are composed of structural features which can be attributes or references to other classes. Classes also contain operations which have attributes. This metamodel can be seen as a simplified metamodel for class diagrams.

For the composition of two models that conform to the Ecore metamodel, it is necessary to define a specific matching operator. The merging operator is reused from the generic framework. Figure 5 shows the model of the composition strategy for Ecore models. This model specializes the generic framework to define a specific matching operator for Ecore models.

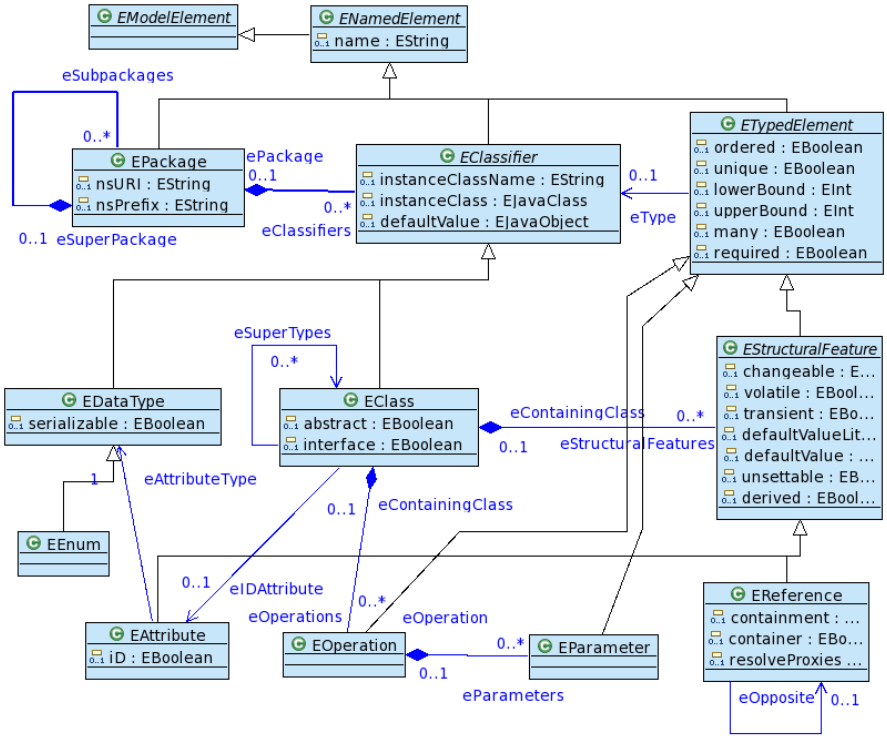

Figure 4 – The Ecore metamodel

According to the model in Figure 5, EMODELELEMENT (of the Ecore metamodel) inherits from MERGEABLE (from the generic framework). This means that all model elements from Ecore have to implement the `getSignature` operation. However, the operation only has to be implemented three times in the case of the Ecore meta-model. A default signature corresponding to their name is associated with all ENAMEDELEMENT. This signature is used to match classes, data types, attributes and references. For operations and parameters specific signatures have to be defined. Two operations will match only if they have the same names, parameters that match and the same return type. Two parameters will match only if they have both the same name and the same type.

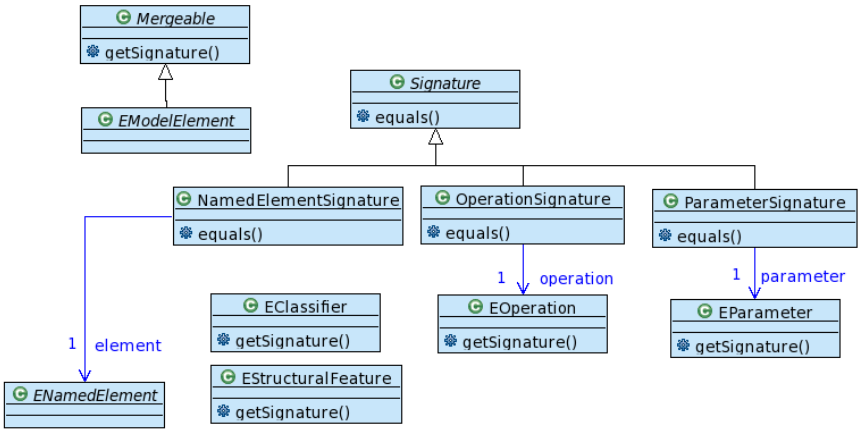

Figure 5 – The composition strategy model for Ecore

Both the generic framework presented in the previous section and its specialization for adding composition capabilities to the Ecore meta-model have been implemented in an open-source tool called Kompose [7]. Kompose was built using the Kermeta language and integrated in the Eclipse IDE as a plug-in. A complete demo of the tool can be found in [7]. Figure 6 presents an excerpt of the Kermeta listing corresponding to the extension depicted in Figure 5. The first line specifies the working package and the second line imports the original Ecore meta-model. The class EMODELELEMENT is then reopened to add the inheritance relation to class MERGEABLE of Kompose and the class ENAMEDELEMENT is reopened to specify how signatures should be computed. These extensions do not break the conformance of existing Ecore models and thus allow directly composing them.

```
package ecore;

require "http://www.eclipse.org/emf/2002/Ecore"

@aspect "true"
class EModelElement inherits kompose::Mergeable {}

@aspect "true"
class ENamedElement
{
  method getSignature() : kompose::Signature is do
    var s : kompose::StringSignature init
      kompose::StringSignature.new
    s.sign := name
    result := s
  end
}
```

Figure 6 – The composition strategy model for Ecore

5 Conclusion

In this paper we detailed a reusable framework for model composition. This framework implements a generic model composition operator that can be specialized for any specific meta-model. This operator is based on signatures associated with model elements for matching objects and uses a generic algorithm for merging objects. We have defined a generic composition directive language for the resolution of potential composition conflicts. It allows both to adapt input models and to fix the composed model. The proposed technique has been implemented as an open-source tool using the Kermeta language.

The main advantage of the proposed approach is to allow easily defining composition operators for new modelling languages. This is especially interesting in a context where domain specific modelling languages are more and more popular. The generic framework is specialized by decorating the meta-model of the language with signatures. These signatures allow capturing semantic elements of the modelling language in order to produce a meaningful composition operator.

The principal limitation of the proposed approach is that to be reusable the framework only relies on the structure of the models to compose. The signatures are the only elements which can be used to take into account some semantics of models to compose. Our current experiments show that it is not an issue when working with structural models such as class diagrams, database schemas or components model but it becomes a clear limitation when working with modelling languages such as sequence diagrams.

To produce a meaningful composition operator for sequence diagrams, the order in which events and messages have to be composed is based on the semantics of sequence diagrams [8]. Using the current version of our composition framework, the only way to implement such a composition operator is to redefine the generic merge operation for the classes of the sequence diagram meta-model which contain

properties that have to be semantically composed. For these classes there is no clear benefit from extending the generic framework as the merging algorithm has to be fully redefined.

As a future work to what is presented in this paper we are currently investigating a finer-grained redefinition mechanism that allows redefining independently the composition strategy for each property of the modelling language meta-model. This allows focusing on properties that require special semantic composition and benefit from the generic implementation for the others.

6 References

1. E. Baniassad and S. Clarke. *Theme: An Approach for Aspect-Oriented Analysis and Design*. In Proceedings of *ICSE'04 (Int. Conference in Software Engineering)*, p. 158-167. Edinburgh, Scotland, 2004.
2. R. Reddy, S. Ghosh, R. France, G. Straw, J.M. Bieman, N. McEachen, E. Song, and G. Georg, *Directives for Composing Aspect-Oriented Design Class Models.* Transaction on Aspect Oriented Development, 2006. **1**(1): p. 75-105.
3. S. Nejati, M. Sabetzadeh, M. Chechik, S.M. Easterbrook, and P. Zave. *Matching and Merging of Statecharts Specifications*. In Proceedings of *ICSE'07 (International Conference on Software Engineering)*, p. 54 - 63. Minneapolis, USA, May 2007.
4. J. Klein, L. Helouet, and J.-M. Jézéquel. *Semantic-based Weaving of Scenarios*. In Proceedings of *AOSD' 06 (International Conference on Aspect-Oriented Software Development)*. Bonn, Germany, 2006.
5. R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry. *Model composition - a signature-based approach*. In Proceedings of *Aspect Oriented Modeling (AOM) Workshop associated to MoDELS'05*. Montego Bay, Jamaica, October 2005.
6. P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. *Weaving executability into object-oriented meta-languages*. In Proceedings of *MoDELS'05*, p. 264 - 278. Montego Bay, Jamaica, October 2005.
7. F. Fleurey. *Kompose : a generic model composition tool*. 2007. Available from: http://www.kermeta.org/kompose/.
8. J. Klein, F. Fleurey, and J.-M. Jézéquel, *Weaving multiple aspects in sequence diagrams.* Trans. on Aspect Oriented Software Development, 2007.