Panorama : A Unified Framework for Model Composition

Amine El Kouhen

Panorama : A Unified Framework for Model Composition

Amine El Kouhen
Concordia University
Faculty of Engineering and Computer Science, Montreal (QC), Canada
elkouhen@encs.concordia.ca

Abstract

Model Driven Engineering promotes the separation of concerns to deal with the design's complexity and maintainability. However, using this practice implies the creation of several heterogeneous models using different notations. It is then necessary to compose these models to reason on the overall designed system for many purposes such as : checking the global consistency of the models, understanding the interactions across the composed models, generating code, etc.

Currently, model compositions are done in ad-hoc ways. Each tool provides its own composition operators and tools, preventing thereby the reuse of these operators. In order to propose a unified methodology to compose models, we present in this paper a comparison of existing composition operators and a framework for integrating these operators to metamodeling languages to support automatic composition capabilities for the models that conform to these languages.

Keywords Model Composition, Composition Operators, Modeling Languages, Separation of Concerns

1. Introduction

In our research projects, we explore diversity as a foundation of software design. Increasing diversity in a system, provides a set of software solutions, which could eventually be adapted to unexpected situations at the design time. To achieve diversity in our design process, we use the Model Driven Engineering (MDE) paradigm, which emphasizes the separation of concerns to better handle systems complexity. This practice leads to create several heterogeneous models, describing for each one of them a point of view of the designed system. However, in order to reason on the overall system, it is necessary to compose these models for different reasons (e.g. checking their consistency, generating code, etc.)

In current state of the art, model compositions are done in ad-hoc ways, in the sense that each tool uses specific formalisms and algorithms to perform this technique, preventing by this the reuse of composition operators and avoiding their formal checking and verification.

Our Challenge is to uniformize composition methodologies and provide the adequate tooling to support an automatic (native) composition for models. Our proposal is to add composition capabilities to the metamodeling languages (e.g. UML, Ecore, Kermeta, etc.). Thereby the models conformant to these metamodeling languages, should be able to perform automatic compositions instead of using specific (ad-hoc) techniques. The advantage of our approach is to reuse composition operators and their formal validation.

Our goal is to provide an integrated environment for the design of complex ubiquitous systems by providing : i) a formal framework, which integrates the state of the art of model composition techniques; ii) an open-source modeling platform in which we integrate and reuse these composition techniques.

In this paper, we start by presenting the basics of model composition techniques at a high level of abstraction. We describe then, our approach to compose heterogeneous models in our Framework. Next, we apply our framework to the Ecore [1] language to show how can our Framework be integrated to a metamodeling language. In sections 4 and 5, we summarize respectively other tools providing composition capabilities and we conclude with our perspectives.

2. Foundations

A model represents a system according to a particular point of view. Facilitating by this the understanding and the validation of this particular aspect of the system [9].

Model composition consists in integrating several models representing different points of view of the designed system into a single one. A model composition has different parameters : Inputs, Operators and the Precision of the composition (deterministic, probabilistic, or fuzzy) [1]. In this paper, we focus on the composition operators and the operands (inputs). Composition precision may be a criterion among others when we validate a composition operator.

2.1 Composition Inputs

There exist different techniques to compose models. They are classified into two categories : symmetric and asymmetric.

When the inputs are of the same "types" and the order of the inputs does not matter (i.e. the composition is commutative), the *Symmetric* composition may be supported ($A \otimes B = B \otimes A$). However, when the input models are of different "types" (e.g., conformant to different metamodels) or the order of the inputs is important, the *Asymmetric* composition is probably supported in this case ($A \otimes B \neq B \otimes A$).

The composition inputs can be done explicitly through basic operators : inheritance (hierarchical composition), aggregartion and composition (as defined in modeling languages), packaging (grouping mechanism). It can be done also by pattern matching (e.g., a regular expression or by using a template), or by binding (i.e.,

[1] https://www.eclipse.org/modeling/emf/

an explicit specification that maps one model element to another model element) [6].

2.2 Composition Operators

According to [8], a model composition operator $\otimes$ is a function with two models as input, and produces a composed model as output: $\otimes : M_1 \times M_2 \longrightarrow M_3$. All composition operators use other atomic / primitive operators to achieve the composition of models. These atomic operators are : Union $\cup$, Intersect $\cap$, Add (Sum) $\oplus$, Replace (Substitution) $\equiv$, Wrap (inclusion) $\subseteq$ and Concatenation. In this paper, we focus on composition operators at a high level of granularity (methodic level).

2.2.1 Symmetric composition operators

1. Merging : The action of combining the input models by unifying their overlaps [3]: their common elements are included once and the other ones are preserved. Merging is symmetric, in the sense that the result of the merge does not depend on the order of the inputs, because in the end, the inputs are translated into a common notation.

   The merge mechanism has been introduced in several metamodeling languages such as UML and MOF to improve modularity. It takes as input two packages, and extends one of them (receiving one) with the other by merging their common content, and deep copying the other ones (Figure 1).

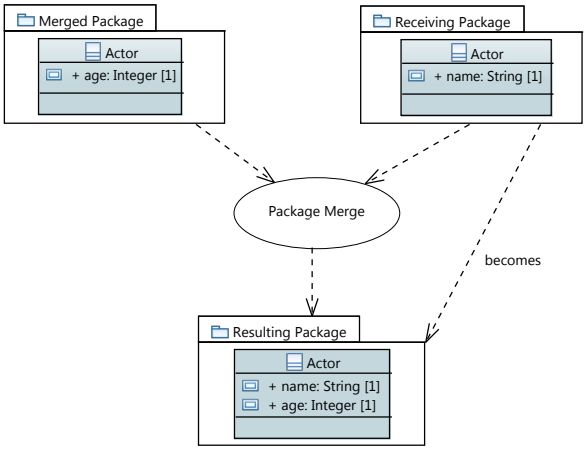

Figure 1. Merging Process

   In the case where some elements in these packages represent the same entity, we talk then about Match. A Match occurs when two elements that have the same identifier are detected (table 1). When two model elements have the same identifier, they form a match candidate. The next step is to compare the signatures of these elements : when signatures of two matching elements are unequal, a conflict is detected and their contents must be combined according to several rules. For UML, the conflict management of the Package Merge is defined in [7]. When there are no conflicts between the two inputs, the Merge is then a Union of these inputs.

   The output of the merge replaces the receiving (called also preserving or absorbing) package. However, except for the place where the merged model is stored, the result does not depend on the order of the inputs. The content of both packages and produces a new package (resulting) that merges the contents of the initial packages [10]. $\otimes : M_1 \times M_2 = M_2'$ (M_2 is the receiving input).

Element	***Identifier***	***Signature***
Package	Name	Qualified Name
Class	Name + Package Name	Identifier + Kind (abstract, final, static) + Visibility (public, private, protected) + SuperClasses
Method	Name + Number of Parameters + Class Name + Package Name	Identifier + Exceptions + List of Parameters (names and types) + Kind + Visibility + Return Type
Attribute	Name + Class Name + Package Name	Identifier + Kind + Visibility + Type
Association	Name + Roles Name + Ends Types	Name + Roles Names + Ends Types

Table 1. Model elements : identifiers and signatures

2. Parallel integration (commutative composition) $\parallel$: This operator is defined as a set union, which is commutative. However, the union operator may lead to inconsistent models in the output (redundancy, relation constraints, deletion constraints, etc.). $\otimes : M_1 \parallel M_2 = M_2 \parallel M_1 = M_1 \cup M_2$

2.2.2 Asymmetric composition operators

1. Weaving : In Aspect-Oriented paradigm, weaving is used to integrate cross-cutting concerns (aspects) into a base system. The aspect consists of a pointcut, which is the pattern to match in the base model, and an advice, which represents the modification made to the base model during the weaving. The parts of the base model that match the pointcut are called joinpoints. During the weaving, each joinpoint is replaced by the advice. $\otimes : M_1 \bullet JP \bullet M_2 \underset{w}{\longrightarrow} M_1 \bullet Ad \bullet M_2$.

   As the two inputs of a weaving (the base model and the aspect) are not idempotent, in the sense that they do not play the same role. The order of inputs is very important in the composition process, weaving is then asymmetric. More precisely, in the merging (symmetric), each element is considered as unique and should appear only once in the output, whereas in the weaving, each element of the aspect should be duplicated as many times as there are joinpoints.

2. Sequential Integration (Ordered Composition) $\bullet$: It is possible to order a merge. In this case, the user needs an operator that allows the precedence on events. We can then talk about the Superimposition. Per example, this operator is useful when we need to compose two Sequence Diagrams SD. As we know, the order in such diagram (i.e. regarding the lifetime) is very important and the preservation of this order in the composition process is a major constraint, whereas the composition $\otimes$: $SD1 \bullet SD2 \neq SD2 \bullet SD1$.

3. Panorama : a Framework for Model Composition

According to [8], model composition has impacts on at least three different levels: Syntactic level, Semantic level and Tooling level. Our work aims to describe composition at the two first levels and to provide the adequate tooling to support native composition capabilities natively into metamodeling languages. Thus, the framework called *Panorama* allows composing models that conform to these languages. The composition mechanism implemented in our framework transcribes a four-steps process :

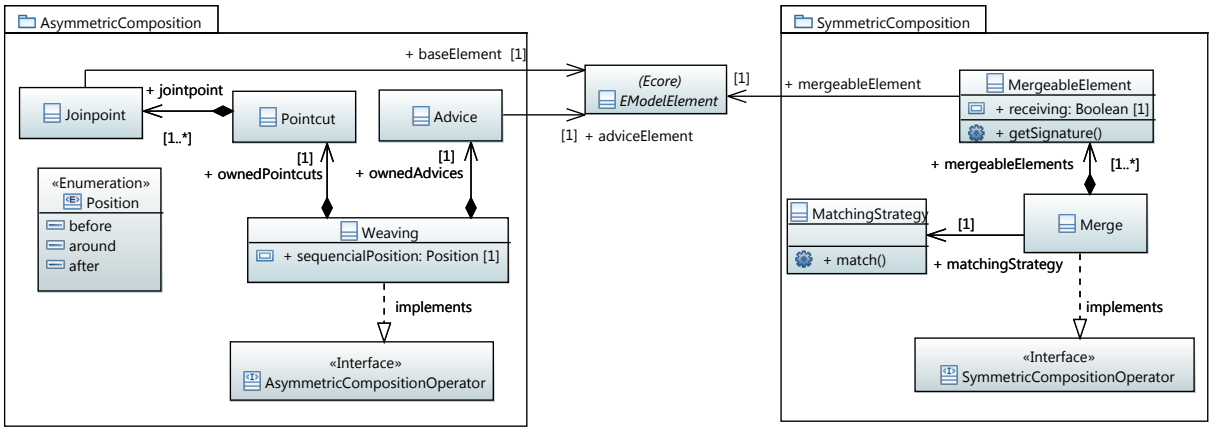

Figure 2. *Panorma* Composition Framework - Applied to Ecore

- First, there is a pre-processing step, during which the inputs may be modified. Some composition approaches require that only consistent models could be composed, implying that inconsistent models must be repaired prior to or during the composition process [4]. Inputs may necessitate some transformation, particularly when they are heterogeneous (i.e. represented in different notations or have different metamdodels), in this case they need to be translated into a common notation first [11].
- Then, there is a mapping step, during which a one-to-one relation, in which elements to be composed are linked. A mapping is the result of the application of the match on every elements of the two composed inputs. If two elements a and b are related by a binding (mapping), the pair (a, b) is called : alignment rule [11]. Note that we only consider binary bindings. In the symmetric case, n-ary bindings can be decomposed in a set of binary bindings. The n-ary bindings in the asymmetric compositions need further investigations.
- The third step is the composition operation (described in section 2.2), where the models are effectively composed. It uses the mapping step in order to identify elements that should be composed and resolve matching conflicts.
- Finally, the post-processing step performs various operations to the output model (e.g. to conform the output to its metamodel, the break of containment cycles...).

Figure 2 describes the metamodel on which *Panorama* is based. At the current state of our work, we applied our framework to Ecore metamodel to make models that are conformant to this language composable. The architecture of our composition framework consists of three packages :

1. *CompositionUtil* : we define *CompositionOperator* as Interface to be implemented by other composition operators. The *execute()* operation has to be specialized to define the composition rules of the operator. As we explain above, a Composition Operation may need pre-processing and post-processing treatments (e.g. to force matches, to override default merge rules of the inputs, etc.) For this reason, we associate the composition operation to Pre and Post processing statements. The operation *execute()* in Statement has to be specialized to define the algorithm for pre and post processing steps in a composition process (Figure 3).
2. *Asymmetric Composition* : we specialize the *CompositionOperator* to define asymmetric operators such as weaving or superimposition. In this case, the weaving operator defines an *advice* and a *pointcut*. The Pointcut may be composed of several *joinpoints*. As the order of composition is important in the asymmetric composition, we define in the *AsymmetricOperator* an attribute to define the position of the advice in the base model.

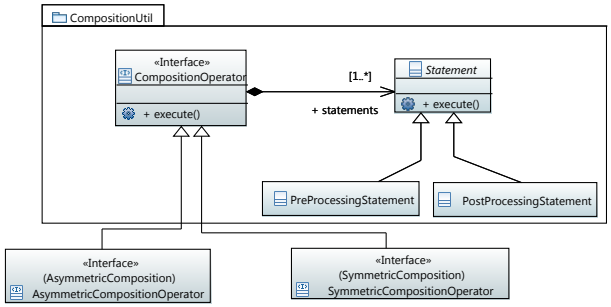

Figure 3. *CompositionUtil* Package

3. *Symmetric Composition* : defines the concepts related to unordered composition like merging. A Merge operator needs to define MergeableElements and uses the *getSignature()* operation to define the matching rules into a matching strategy. We can also define if a *MergeableElement* is a receiving element or not. When all mergeable elements in a merge operation are not receiving elements, Panorama interprets this as a parallel composition. We may have then, redundancy and some inconsistency in the output as we explain in section 2.2 (Parallel integration). 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. 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 pre-processing statements. The signatures of elements are defined as we present in Table 1.

Figure 4 summarizes how *Panorama* adds composition capabilities to a metamodel. The framework has been defined in such a way that it can be used on top of any language that conforms to EMOF. *Panorama* metamodel consists of basic concepts of composition techniques. In this level, these concepts are generic and could be reused for different metamodeling languages. Next, we create an

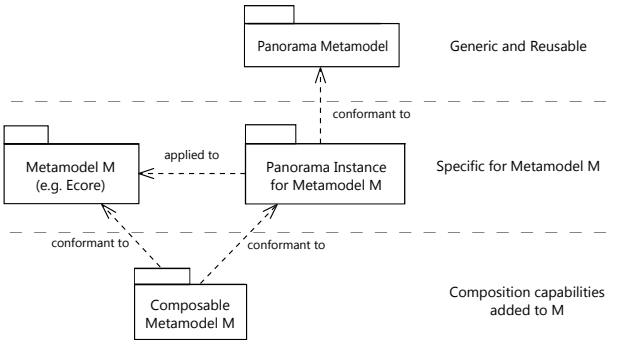

Figure 4. Adding composition capabilities to a Metamodeling language

instance of Panorma that fits with a specific language (in this case Ecore) and we apply the Panorama instance to the core elements of Ecore. We get then, a composable version of Ecore in which we can describe models supporting native composition capabilities.

In the current state of our work, *Panorama* still at the prototype state. We succeed to implement the symmetric composition capabilities and to apply them to the Ecore language. The next step will be the implementation of the asymmetric operators and to provide a public release of our tooling. We have also to validate our tool regarding composition properties [3] which are: completeness, minimality, totality, idempotency, validity and non-redundancy (only for symmetric operators). This validation and the complete implementation will be the objects of a detailed publication.

4. Related Work

There exist several tools, which can compose models such as Kompose [2] [5], EMF Diff/Merge [3]. All of them allow to merge two homogeneous models (instances of the same metamodel) by comparing the signatures of their elements. However, these tools are limited to merge only structural languages and did not support behavioural languages composition. TreMer+ [4] proposes to match and merge behavioural languages as state chart diagrams while preserving their semantics by ensuring bisimulation. However, preserving bisimulation may lead to the duplication of states in the two diagrams, thus not respecting the non-redundancy [12] property in the merge operation. The only one that allows asymmetric composition is KerTheme [2], which is based on Aspect-Oriented paradigm. However, as for TreMer+, it uses asymmetric operators to perform symmetric compositions that may lead to redundancy and other inconsistencies in the output models.

5. Conclusion

In this paper we introduced a reusable framework for model composition. This framework implements symmetric and asymmetric composition operators that can be specialized for any specific meta-modeling language. The proposed technique has been implemented to unify composition capabilities into the meta-modeling languages.

The proposed approach can be integrated to any metamodeling language based on EMOF and allows the reuse of existing composition operators and the definition of new composition operators as needed.

[2] http://www.kermeta.org/mdk/kompose

[3] http://eclipse.org/diffmerge

[4] http://se.cs.toronto.edu/index.php/TReMer+

The existing approaches to compose models are limited to the structural languages (e.g. structural models such as class diagrams, database schemas or components model). However, it becomes a clear limitation when we have to compose behavioural models such as sequence diagrams. Our proposal allows to overcome this limitation by proposing sequential asymmetric composition, which can specify the element to be composed and the location (pointcut) of the composition. In addition, we can specify the position of this composition, which can be a real advantage in behavioural languages composition.

The next step of our work is to provide an open-source implementation and validate the composition operators against a set of formal properties to be respected in the outputs.

References

[1] M. Aksit. The 7 c's for creating living software: A research perspective for quality oriented software engineering. *Turkish Journal of Electrical Engineering and Computer Sciences*, 12(2):61–95, 2004. URL `http://doc.utwente.nl/48771/`.

[2] O. Barais, J. Klein, B. Baudry, A. Jackson, and S. Clarke. Composing multi-view aspect models. In *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*, pages 43–52, Feb 2008. .

[3] M. Chechik, S. Nejati, and M. Sabetzadeh. A relationship-based approach to model integration. *Innov. Syst. Softw. Eng.*, 8(1):3–18, Mar. 2012. ISSN 1614-5046. . URL `http://dx.doi.org/10.1007/s11334-011-0155-2`.

[4] D. Fischbein, N. D'Ippolito, G. Brunet, M. Chechik, and S. Uchitel. Weak alphabet merging of partial behavior models. *ACM Trans. Softw. Eng. Methodol.*, 21(2):9:1–9:47, Mar. 2012. ISSN 1049-331X. . URL `http://doi.acm.org/10.1145/2089116.2089119`.

[5] F. Fleurey, B. Baudry, R. France, and S. Ghosh. Models in software engineering. chapter A Generic Approach for Automatic Model Composition, pages 7–15. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-69069-6. . URL `http://dx.doi.org/10.1007/978-3-540-69073-3_2`.

[6] G. Georg, A. Shaukat, B. H. Cheng, B. Combemale, R. France, J. Kienzle, J. Klein, P. Lahire, M. Luckey, A. Moreira, and G. Mussbacher. Modeling approach comparison criteria for the cma workshop at models 2012. ACM / IEEE, 2012.

[7] O. M. Group. Unified modeling language 2.5, 2014. `http://www.omg.org/spec/UML/2.5`.

[8] C. Herrmann, H. Krahn, B. Rumpe, M. Schindler, and S. Volkel. An algebraic view on the semantics of model composition. Springer. ISBN 978-3-540-72900-6. . URL `http://dx.doi.org/10.1007/978-3-540-72901-3_8`.

[9] J.-M. Jezequel, B. Combemale, and D. Vojtisek. *Ingenierie Dirigée par les Modèles : des concepts à la pratique*. Editions ellipses, Paris, 2012.

[10] M. Kezadri Hamiaz, M. Pantel, B. Combemale, and X. Thirioux. A formal framework to prove the correctness of model driven engineering composition operators. Springer. ISBN 978-3-319-11736-2.

[11] J. Y. Marchand, B. Combemale, and B. Baudry. A categorical model of model merging and weaving. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering*, MiSE '12, pages 70–76, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1757-3. URL `http://dl.acm.org/citation.cfm?id=2664431.2664442`.

[12] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of variant feature specifications. *IEEE Transactions on Software Engineering*, 38(6):1355–1375, 2012. ISSN 0098-5589. .