A Categorical Model of Model Merging and Weaving

Jonathan Marchand, Benoit Combemale, Benoit Baudry

A Categorical Model of Model Merging and Weaving

Jonathan Y. Marchand
ENS Cachan
Rennes, France
jonathan.yves.marchand@irisa.fr

Benoit Combemale
Université de Rennes 1 / IRISA
Rennes, France
benoit.combemale@irisa.fr

Benoit Baudry
INRIA
Rennes, France
benoit.baudry@inria.fr

***Abstract*—Model driven engineering advocates the separation of concerns during the design time of a system, which leads to the creation of several different models, using several different syntaxes. However, to reason on the overall system, we need to compose these models. Unfortunately, composition of models is done in an *ad hoc* way, preventing comparison, capitalisation and reuse of the composition operators. In order to improve comprehension and allow comparison of merging and weaving operators, we use category theory to propose a unified framework to formally define merging and weaving of models. We successfully use this framework to compare them, both through the way they are transformed in the formalism, and through several properties, such as completeness or non-redundancy. Finally, we validate this framework by checking that it correctly identifies three tools as performing merging or weaving of models.**

I. Introduction

The Model Driven Engineering (MDE) paradigm promotes separation of concerns to better handle the complexity of a software system, which leads to the creation several models However, in order to reason on the overall system, it is necessary to compose these models, *e.g.* to check their consistency or to generate code. As model composition is a key concern in MDE, a better comprehension of model composition techniques would greatly improve designers work. Unfortunately, composition of models is currently done in an *ad hoc* way, in the sense that each tool uses its own formalism and algorithm to perform it, preventing straightforward easy comparison.

In this article, we focus on merging and weaving operators. In order to get a better understanding of these two kind of operators, we propose a unified framework for model merging and weaving. As the high level of abstraction of category theory allows to identify different notions under a common vocabulary, this framework is based on category theory. More precisely, we use the pushout, a categorical operator which behaves like a merge when applied in a proper category [1]. The categorical framework includes a transformation from classical MDE models to our category **Model** of models in which we perform the pushout. This transformation allows us to pinpoint the difference between merging and weaving operators. We also give several properties inspired from [2], [3], that are verified by one or both approaches, and allow further comparison and comprehension of the two kind of operators. Finally, we validate our framework by checking that it correctly identifies three tools from the literature, UML Package Merge [4], Kompose [5] and ADORE [6], as using a merging or a weaving operator.

Section II gives a first intuition of the difference between merging and weaving. Section III gives the notions of category theory necessary to understand the article and defines our category of models. Section IV explains how we formalise the weaving and merging operators using the pushout in this category, compares them using this framework and validate the latter using three tools resorting to one or the other approach. We explore related work in section V, finally concluding in section VI.

II. Intuitive Definition of Weaving and Merging Operators

In this section, we give an intuition of the difference between merging and weaving models that underlies the formal comparison done in section IV. To do this, we first clarify the vocabulary used, then split the two operators in four steps and precise the relevant steps for the comparison of merging and weaving. Finally we explain the intuitive difference between the merging and weaving approaches.

A. Vocabulary

In this article, we use vocabulary taken from Aspect Oriented Programming to deal with the weaving process. To avoid any ambiguity, we hereby define this vocabulary, as well as the notion of merging.

Definition 1 (Merging)**.** Merging is the action of combining two models, such that their common elements are included only once and the other ones are preserved.

Definition 2 (Weaving)**.** A weaving involves two actors: an aspect and a base model. The aspect is made of two parts, 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.

B. Partition of the Merging and Weaving Processes

Merging and weaving processes can be separated in four steps. First, there is a pre-processing step, during which the input models are modified. This step is performed for the input models to conform to the tool expectation or ease automatic matching by renaming some elements.

Then, there is a match step. It is usually expressed at the metamodel level, by associating with two classes the criteria used to compare instances of these classes. In the rest of this article, we rely on the notion of *mapping*, which is instead expressed at the model level. It is a one-to-one relation, in which elements to be merged are linked. A mapping is the result of the application of the match on every elements of the two compared models.

The third step is the sum, where the models are effectively merged. It uses the match step in order to identify elements that should be merged, and copy the other ones.

Finally, the post-processing step performs various operations on the output model, typically for it to conform to its metamodel. It can consist in the identification and break of containment cycles, or deletion of user-defined elements.

In the rest of this article, we focus on the match and sum steps. Indeed, the pre- and post-processing usually resort to Turing-complete languages, thus preventing any attempt to extract general properties for these two steps. Properties on the overall process of a specific tool can still be extracted from the match and sum step, and proven to be invariant w.r.t. the post-processing of this tool.

C. Intuition of the Difference between Weaving and Merging

From Definitions 1 and 2, we can extract an informal difference between merging and weaving. Merging is symmetric, in the sense that the result of the merge does not depend on the order of the inputs. On the other hand, as the two inputs of a weaving (the base model and the aspect) do not play the same role, weaving is asymmetric. More precisely, in the symmetric merging case, each element is considered unique and should appear only once in the result, whereas in the asymmetric weaving case, each element of the aspect does not represent something unique and should be duplicated as many times as there are joinpoints. This intuition is formalised in section IV-B.

III. A Category of Models

In order to formally compare the merging and weaving approaches, we choose the category theory formalism. Indeed, the high level of abstraction of this formalism allows the identification of different notions under a common vocabulary.

In this section, we first introduce the notions of category theory necessary to define our framework, then use them to define the **Model** category. This category will be used in section IV to formally define the merging and weaving operators.

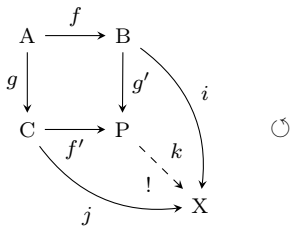

Figure 1. (P, f', g') is a pushout of f and g.

A. Category Theory

Definition 3 (Category)**.** A category consists of:

- a collection of objects[1]
- a collection of arrows

with the following properties:

- Each arrow has a source and a target, called respectively domain and codomain. We note $f : A \to B$ to show that the arrow f has for domain the object A and for codomain the object B.
- There is a composition operator for arrows, such that for every $f : A \to B$ and $g : B \to C$, there exists $g \circ f : A \to C$ in the category.
- This composition is associative: $h \circ (g \circ f) = (h \circ g) \circ f$.
- To each object A is associated an identity arrow $id_A : A \to A$, which is neutral for the arrow composition, *i.e.* $\forall f : A \to B, id_B \circ f = f = f \circ id_A$

Definition 4 (Pushout)**.** A pushout of a pair of arrows $f : A \to B$ and $g : A \to C$ is an object P and a pair of arrows $g' : B \to P$ and $f' : C \to P$ s.t. $g' \circ f = f' \circ g$ and if $i : B \to X$ and $j : C \to X$ are such that $i \circ f = j \circ g$ then there is a unique $k : P \to X$ s.t. $i = k \circ g'$ and $j = k \circ f'$.

This definition is illustrated on Figure 1. The symbol ⟳ means that the diagram commutes, *i.e.* that the equalities of the definition are satisfied, *e.g.* $i = k \circ g'$. The arrow k is dashed to emphasise that its existence is a consequence of the rest of the diagram. The exclamation mark "!" emphasises the uniqueness of the arrow k, in the sense that it is the only arrow which makes the diagram commute.

B. A Category for Models

We define in this section the **Model** category, a category of models. This category will be used in section IV to define the merge and weaving in terms of a pushout.

In order to define the **Model** category, we first define formally the notions of model (which corresponds to the usual notion of model in MDE) and model morphism (which can loosely be interpreted as a mapping function between models).

[1]We use a different font for the notion of object of a category in order not to confuse it with the notion of object as element of a model used later

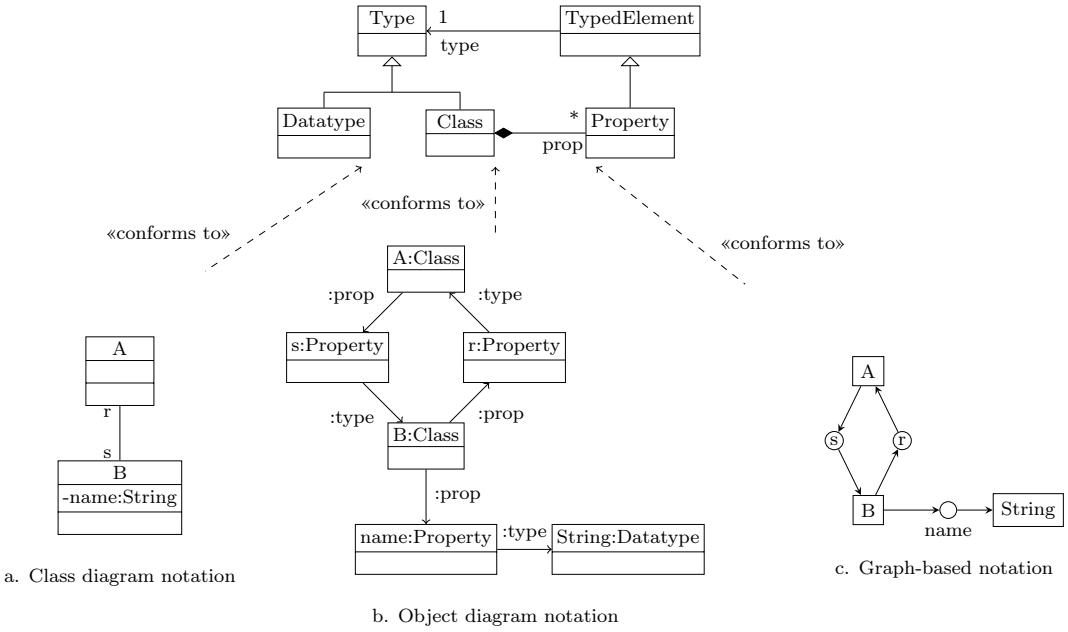

Figure 2. Three representations of a class diagram, and its metamodel.

A model is a set of objects linked through associations. In order to treat associations as first-class citizens and not as simple links, we introduce the notion of edge, which links associations and objects.

Definition 5 (Model)**.** A model is a set O of objects, a set R of associations and a set E of edges such that

1) $E \subseteq O \times R \cup R \times O$
2) $\forall (r, o) \in E \cap R \times O, \forall o' \in O, (r, o') \in E \iff o = o'$
3) $\forall (o, r) \in E \cap O \times R; \forall o' \in O, (o', r) \in E \iff o = o'$

Given an edge e, we use the notations $e^{(o)}$ and $e^{(r)}$ to represent the object and association ends of e.

The information on the direction of an association is stored in its edges: if $(o, r) \in E$, r is an association *from* object o, whereas if $(r, o) \in E$, r is an association *to* o. The points 2 and 3 state that an association is from at most one object and to at most one object. For the sake of simplicity, we may write that $x \in M$ instead of $x \in O_M \cup R_M \cup E_M$ to say that x is an element of the model M.

Figure 2 illustrates a model and its formal representation according to the previous definition. The upper part of the figure represents a simplified metamodel of a class diagram. The lower part shows three different equivalent notations for a class diagram conforming to the previous metamodel. Figure 2a depicts the model using the class-diagram notation. The same model is illustrated in Figure 2b, using the object diagram notation showing the conformance to the metamodel. Finally, Figure 2c, proposes the underlying graph according to the latter, which corresponds to the formal representation of the model, as expressed in Definition 5. Notice that we uniformly consider attributes as references to their relative datatypes.

We now define model morphisms as three functions on objects, associations and edges.

Definition 6 (Model morphism)**.** Given two models $A = (O_A, R_A, E_A)$ and $B = (O_B, R_B, E_B)$, a model morphism $f : A \to B$ is a triple (f_O, f_R, f_E) with $f_O : O_A \to O_B$, $f_R : R_A \to R_B$ and $f_E : E_A \to E_B$ three functions such that $\forall e \in E_A, f_O(e^{(o)}) = (f_E(e))^{(o)}$ and $f_R(e^{(r)}) = (f_E(e))^{(r)}$.

The goal of the latter constraint is to preserve the structure of models: if an edge is mapped with another edge, then their ends are mapped accordingly. For the sake of simplicity, we use *morphism* instead of *model morphism* in the rest of this article.

According to Definition 3, we need composable arrows and the identity arrows to define the category. We thus formally define model morphism composition and the identity model morphism.

Definition 7 (Model morphisms composition)**.** The composition of two model morphisms is done component-wise, *i.e.* $g \circ f = (g_O \circ f_O, g_R \circ f_R, g_E \circ f_E)$.

Proof: We need to check that the composed morphism satisfies the condition of Definition 6. Let $e \in E_A$ be an edge. As f and g are morphisms, we have that $f_O(e^{(o)}) = (f_E(e))^{(o)}$ and $g_O((f_E(e))^{(o)}) = (g_E(f_E(e)))^{(o)}$. So $g_O \circ f_O(e^{(o)}) = (g_E \circ f_E(e))^{(o)}$.
The arguments are the same for $e^{(r)}$. ■

Definition 8 (Identity model morphism)**.** Given a model A, the identity morphism of A is a model morphism $id_A : A \to A$ such that $(id_A)_O$,$(id_A)_R$ and $(id_A)_E$ are the identity functions on O_A, R_A and E_A respectively.

Proof: We need to check that id_A satisfies the condition of Definition 6. Let $e \in E_A$ be an edge. As $(id_A)_O$ is the identity function on O_A, $(id_A)_O(e^{(o)}) = e^{(o)}$. As $(id_A)_E$ is the identity function on E_A, $(id_A)_E(e) = e$. So $((id_A)_E(e))^{(o)} = e^{(o)} = (id_A)_O(e^{(o)})$.
The same arguments hold for $e^{(r)}$. ■

We now prove that models together with model morphisms form a category.

Definition 9 (Category **Model**)**.** The category **Model** has models as objects and model morphisms as arrows. Composition of arrows is the composition of morphisms. Identity arrows are identity morphisms.

Proof: We have to check if the four properties of Definition 3 are satisfied.

- Every morphism has obviously a source and a target
- Given two morphisms $f : A \to B$ and $g : B \to C$, the composed morphism $g \circ f : A \to C$ always exists.
- The composition of functions is associative and the composition of morphisms is done component-wise, so composition of morphisms is associative:

$$\begin{aligned} h \circ (g \circ f) &= (h_O \circ (g_O \circ f_O), h_R \circ (g_R \circ f_R), \\ &\quad h_E \circ (g_E \circ f_E)) \\ &= ((h_O \circ g_O) \circ f_O, (h_R \circ g_R) \circ f_R, \\ &\quad (h_E \circ g_E) \circ f_E) \\ &= (h \circ g) \circ f \end{aligned}$$

- The identity functions are neutral for function composition and the composition of morphisms is done component-wise, so the identity morphism is neutral for morphism composition. ■

IV. Categorical Merge and Weaving Operators

In this section, we use the pushout in the previously defined **Model** category to formally define merging and weaving. We begin by formally defining the notion of mapping in our category. Then we expose how the transformation from models to elements of the category allows to differentiate merging and weaving, and how the pushout is used to perform them. We then define several properties that are used to check the commonalities and differences between these two approaches. Finally, we check that the framework correctly classifies three tools from the literature: UML Package Merge, Kompose and ADORE.

A. Mappings in the *Model* Category

In order to treat merge in **Model**, we define a notion of mapping between elements of the category **Model**, which corresponds to the notion of mapping between models introduced in section II-B.

A mapping is a relation between two models, defined through three relations on objects, associations and edges. If two elements a and b are related by a mapping, we call the couple (a, b) an alignment rule. Note that we only consider binary mappings. In the symmetric case, n-ary mappings can be decomposed in a set of binary mappings. The notion of n-ary mapping in the asymmetric case should need further investigations.

Definition 10 (Model mapping)**.** Given two models $B = (O_B, R_B, E_B)$ and $C = (O_C, R_C, E_C)$, a model mapping map between B and C is a triple (map_O, map_R, map_E) where $map_O \subseteq O_B \times O_C$, $map_R \subseteq R_B \times R_C$, $map_E \subseteq E_B \times E_C$ are three sets of alignment rules on objects, associations and edges respectively, such that $\forall e_b, e_c \in map_E, (e_b^{(o)}, e_c^{(o)}) \in map_O$ and $(e_b^{(r)}, e_c^{(r)}) \in M_R$.

The latter constraint has the same role than the similar constraint on model morphisms: to preserve the structure of models.

B. From MDE Models to a Pushout Configuration

As the pushout is fixed by its definition, the only way to make a difference between merging and weaving in the sum process lies in the transformation from the input models (conform to their metamodel) to a pushout configuration: two arrows with a common domain. Hence, this transformation takes as input not only the models to be composed and the mapping between them, but also the type of composition (weaving or merging) expected.

1) Symmetric Case: Given B and C two models and map a mapping between them, we want to merge them w.r.t. map. To do this, we first transform map in a model A together with two morphisms f and g, using the following procedure. For every alignment rule $(o_b, o_c) \in map_O$, we add an element o_a to O_A and define $f_O(o_a) = o_b$ and $g_O(o_a) = o_c$. We use the same procedure for map_R and map_E to define R_A, E_A, f_R, f_E, g_R and g_E. We can prove that $f = (f_O, f_R, f_E)$ and $g = (g_O, g_R, g_E)$ are model morphisms, using the structure preserving property of map which leads to the structure preserving property of f and g.

As an example, consider Figure 3, where two models B and C and a mapping map between them are pictured. After the transformation of map into a model A, we obtain the pushout configuration picture on Figure 4. As f and g are morphisms with common domain, we can compute their pushout.

2) Asymmetric Case: If we want to weave an aspect to several joinpoints, the pointcut of this aspect will have to be mapped on each joinpoint. By transitivity of the mapping relation, all the joinpoints will be considered equivalent, hence they will all be merged during the merge process, not yielding the expected result. Thus, in order to make an asymmetric merge, we have to duplicate the aspects so that each joinpoint can have its own.

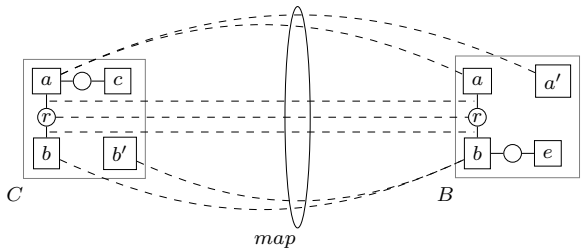

Figure 3. Two models B and C and a mapping map between them, represented in dashed lines

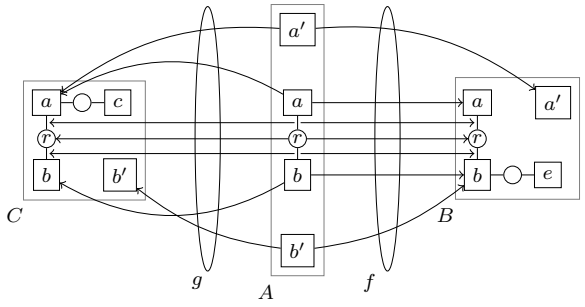

Figure 4. Two models B and C mapped by a third model A and two morphisms $f : A \to B$ and $g : A \to C$

An example of the procedure is depicted on Figure 5. The upper part of the figure represents the base model B and the aspect model C. C is made of one advices, which pointcut is a. The advice is linked to the joinpoint in B using dashed lines. The lower part represents the result of the translation to the model category. As the advice a-c has to be weaved in two joinpoints, it has been duplicated in model C. The model A has been created from the dashed lines of the upper part, and represents the pointcuts (duplicated if needed).

C. Merging and Weaving Models using Pushout

Using the procedure described in the two previous sections, from a mapping map we obtain two arrows f and g with common codomain A, both in the symmetric and asymmetric case. If we compute their pushout[2] (D, f', g') in **Model**, we can check that it yields the expected result in the two cases.

Every couple of elements $(b, c) \in B \times C$ which have the same antecedent $a \in A$ by f and g respectively will have the same image $d \in D$ by f' and g' respectively, thanks to the commutativity property $f' \circ g = g' \circ f$ of the pushout. As each element a of A has been constructed from an alignment rule of map, and as its images by f and g are the ends of this alignment rule, every couple of elements mapped by map will have the same image in D, which means that they have been merged. If we take up the previous symmetric example, and do the pushout of f and g, we indeed get the merge D of B and C w.r.t. map, as pictured on Figure 6.

[2]The pushout exists in **Model** as long as the mapping is consistent, *i.e.* if two references are mapped, the classes they are linked to should be mapped as well.

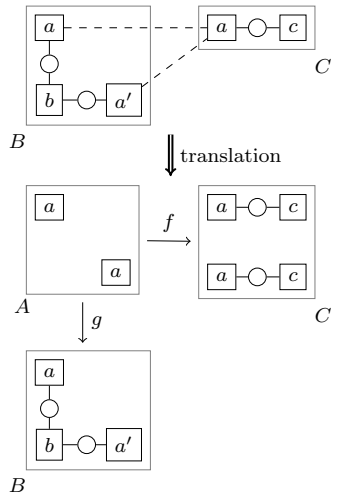

Figure 5. Transformation towards a pushout configuration in the case of an asymmetric merge.

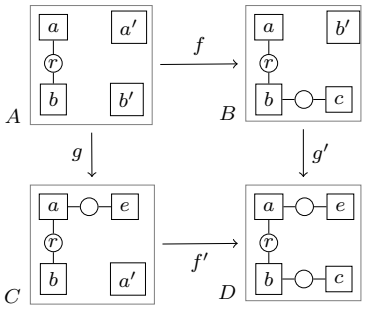

Figure 6. D is the sum of B and C with respect to A.

Nevertheless, this procedure gives only the structural part of the merging and weaving and does not give the values of the attributes of the resulting model. Indeed, consider the attribute `name:String` of some merged element, it may be the concatenation of the names of its antecedents, or the name of one of them, or any other possibility. However, the pushout is not only D but the triple (D, f', g'). Thanks to the morphisms f' and g', we can trace back the origin of each element of the resulting model, including the attributes. Knowing the exact origin of an attribute, we can compute its expected value, but this operation requires some post-processing out of the categorical framework, such as the application of a preference function [7].

D. Composition Operators Properties

In both cases (symmetric and asymmetric), the merging operators satisfy the following properties, adapted from Brunet and Chechik [2], [3]:

- **Completeness:** Ensure that no data is lost along the merging process: each model element from the input models should be represented in the merged model.
- **Minimality:** The merged model contains information that originates solely from the input models.
- **Totality:** Given any pair of models and any mapping, a merged model should always exists.
- **Idempotency:** Merging a model with itself should produce a model that is an exact copy of the original model.
- **Validity:** The operator merges only (transitively) matching elements.

Proofs:

- **Completeness** The pushout operator does not remove any information, as the merged model contains the image of all the objects, associations and edges of the input models. Indeed, as the arrows f' and g' produced by a pushout are morphisms, made of three functions, they necessarily provide an image for each element.
- **Minimality** Every model element in the merged model has an antecedent by f' or g'. This guarantees that no information is added during the pushout. This property comes directly from the co-universal property of the pushout [8].
- **Totality** Let M_1 and M_2 be two models and map be a mapping between M_1 and M_2. The sum of M_1 and M_2 w.r.t. map exists as long as map verifies the following property: if a reference $r_1 \in R_{M_1}$ is mapped with a reference $r_2 \in R_{M_2}$, then their incoming and outgoing edges are mapped accordingly. From a MDE perspective, this condition says that when two references are mapped together, then their containers and containees are mapped as well.
- **Idempotency** In the symmetric case, if you merge a model with itself using the identity mapping (which maps each element to itself), the result is the model itself. In the asymmetric case, consider the identity mapping as the pointcut, and the model as both the advice and the receiving model. As the advice is identical to the pointcut, it means that we weave an empty advice on the receiving model. Hence, in the asymmetric merge, the idempotency property is preserved.
- **Validity** In the two cases, the merged elements are the elements (transitively) linked by alignment rules. Indeed, these merged elements are the ones that have the same antecedents in A by f and g. As A, f and g are generated from the alignment rules, the operators merge only matching elements. ■

The following property, also inspired from [2], [3], is true for a symmetric merge, but false for an asymmetric one:

- **Non-redundancy:** Any pair of matched model elements leads to the creation of a single element in the merged model.

Indeed, in the asymmetric case, elements of the advice model may be duplicated during the transformation to a **Model** element, hence this latter property does not hold.

Proof for symmetric case: This property is guaranteed by the construction of A, f and g from the mapping. If x from B and y from C are linked by the mapping, they have the same antecedent in A by f and g respectively, hence they have the same image by f' and g' thanks to the commutativity property $f' \circ g = g' \circ f$ of the pushout. ■

E. Validation Using Existing Tools

In order to validate our framework, we check if it correctly classifies three tools from the literature as using a merging (symmetric) or a weaving (asymmetric) operator. This is done by checking which of the transformation processes exposed in section IV-B is adapted to simulate the tool using the **Model** category. A fourth tool, TreMer+ [10], is introduced to show the limits of our approach.

UML Package Merge: Package merge has been introduced in UML2 to improve modularity [4]. It takes as input two class diagrams, and extends the first with the second by merging their common classes, and deep copying the other ones. These common classes are identified by their names and types. The merge is done recursively, following the containment links of the models. The output of the merge replaces the first class diagram, hence the merge is asymmetric, in the sense that only the receiving model is modified. However, except for the place where the merged model is stored, the result does not depend on the order of the inputs. Moreover its algorithm is symmetric. This algorithm fits seamlessly in the merging process of our framework.

Kompose: Kompose[3] is a model merging tool [5] implemented in the Kermeta language [9]. It merges two homogeneous models (instances of the same metamodel) by comparing the signatures of their elements. These signatures can be arbitrarily complex, using the element's name, type, field or method names and types, and so on. Elements with the same signature are merged, while the other ones are deep copied. Kompose proposes a system of pre- and post-directives to modify the models before and after the merge. The two models are called base model and aspect model, as in Aspect-Oriented Modeling, which suggests an asymmetric treatment, yet the tool is symmetric [5]. Only the merging process of our framework simulates correctly Kompose.

ADORE: ADORE[4] is a tool for service orchestration using PROLOG. It allows the merging of partial orchestrations (fragments) in a main orchestration using a set of user-defined relationships [6]. The treatment is clearly asymmetric, as the fragments are not orchestration in themselves (they do not conform to the metamodel), and must be merged

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

[4]http://rainbow.i3s.unice.fr/adore/wiki/doku.php

in the main orchestration in order to make sense. As a fragment might be weaved in several places in the main orchestration, only the weaving process of our framework is relevant.

TreMer+: In [10], Nejati *et. al.* propose an approach, implemented in TreMer+[5], to match and merge statechart diagrams while preserving their semantics by ensuring bisimulation. Thus, their operator is not only structural, and does not fit in our framework. Indeed, preserving bisimulation may lead to the duplication of states in the two diagrams, thus not respecting the non-redundancy property [11]. From our framework point of view, it is as if the merge was asymmetric (as states are duplicated) in a symmetric manner (this duplication may occur in both models, hence they play the same role).

V. Related Work

In [12], Jeanneret *et. al.* propose a categorisation of model composition operators based on three criteria: What, Where and How. The What criteria tells which concepts of the models will be composed, the Where criteria indicates where the composed concepts will be inserted, and the How criteria describes how the composition of the concepts is done. This qualitative approach shares the same purpose than ours, as it allows the comparison of different composition operators and gives a better understanding of them. However, our categorical framework provides an operational operator, which can be used to promote re-use and evolution management in future work.

The use of colimits (a generalisation of pushouts in category theory) to perform merging has been encouraged since the 90s by Goguen [13] and already been applied to merge models, *e.g.* in the TreMer tool [14], [1]. While they use a categorical framework to define a new merging operator, we aim at capitalising the sum process of different merging approaches. Moreover, these approaches, as well as the algebraic graph transformation approaches to model transformation [15], consider models as (possibly attributed) graphs and do not take into account the specificity of models. Indeed, they consider associations as simple edges whereas we consider them as first-class constructs, which allows us to manipulate them.

VI. Conclusion

This paper proposes a formal framework based on category theory to unify the common mechanisms of model merging and model weaving. Using this framework allow us to compare these two kind operators, bringing a better comprehension of their commonalities and differences. This comparison is based on two points: the transformation from the inputs to a pushout configuration, and the properties checked by the two approaches. This framework provides a clear definition of the two kinds of composition, and is validated by classifying correctly three tools from the literature in one or the other category.

We envisage to extend the framework to support more model composition operators, *e.g.* intersection and override, in order to bring a better comprehension of this key concern in MDE.

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

Acknowledgement

This work has been partially supported by VaryMDE, a collaboration between INRIA and Thales Research and Technology. The authors warmly thank Robert B. France for insightful feedback and comments on early versions of this paper.

References

[1] M. Sabetzadeh and S. M. Easterbrook, "An algebraic framework for merging incomplete and inconsistent views," in *RE*, 2005, pp. 306–318.

[2] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh, "A manifesto for model merging," in *GaMMa*, 2006, pp. 5–12.

[3] M. Chechik, S. Nejati, and S. Mehrdad, "A relationship-based approach to model integration," *Journal of Innovations in Systems and Software Engineering*, 2011.

[4] OMG, "Uml infrastructure specification v2.4," 2010. [Online]. Available: http://www.omg.org/spec/UML/2.4/

[5] R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh, "Providing support for model composition in metamodels," in *EDOC*, 2007, p. 253.

[6] S. Mosser, M. Blay-Fornarino, and R. France, "Workflow design using fragment composition," in *TAOSD VII*, 2010, vol. 6210, pp. 200–233.

[7] R. Pottinger and P. Bernstein, "Merging models based on given correspondences," in *Proceedings of the 29th VLDB-Volume 29*, 2003, p. 873.

[8] B. C. Pierce, *Basic category theory for computer scientists*, ser. Foundations of computing. MIT Press, 1991.

[9] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, "Weaving executability into object-oriented meta-languages," in *MODELS/UML*, ser. LNCS, vol. 3713, 2005, pp. 264–278.

[10] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of statecharts specifications," in *ICSE*, 2007, pp. 54–64.

[11] ——, "Matching and merging of variant feature specifications," *IEEE Transactions on Software Engineering*, 2011.

[12] C. Jeanneret, R. France, and B. Baudry, "A reference process for model composition," in *Workshop on Aspect-Oriented Modeling*. ACM, 2008, pp. 1–6.

[13] J. A. Goguen, "A categorical manifesto," *Mathematical Structures in Computer Science*, vol. 1, no. 1, pp. 49–67, 1991.

[14] M. Sabetzadeh, S. Nejati, S. Easterbrook, and M. Chechik, "Tremer: A tool for relationship-driven model merging," in *FM*, 2006.

[15] H. Ehrig, U. Prange, and G. Taentzer, "Fundamental theory for typed attributed graph transformation," in *Graph Transformations*, ser. LNCS, 2004, vol. 3256, pp. 161–177.