Aspect Model Unweaving

Jacques Klein, Jörg Kienzle, Brice Morin, Jean-Marc Jézéquel

Aspect Model Unweaving [⋆]

Jacques Klein[1], Jörg Kienzle[2], Brice Morin[3], and Jean-Marc Jézéquel[3]

[1] Centre de Recherche Public Gabriel Lippmann,
klein@lippmann.lu
[2] School of Computer Science, McGill University,
Joerg.Kienzle@mcgill.ca
[3] INRIA, Centre Rennes - Bretagne Atlantique / IRISA, Université Rennes1
Brice.Morin@inria.fr | jezequel@irisa.fr

Abstract. Since software systems need to be continuously available, their ability to evolve at runtime is a key issue. The emergence of models@runtime, combined with Aspect-Oriented Modeling techniques, is a promising approach to tame the complexity of adaptive systems. However, with no support for aspect unweaving, these approaches are not agile enough in an adaptive system context. In case of small modifications, the adapted model has to be generated by again weaving all the aspects, even those unchanged. This paper shows how aspects can be unwoven, based on a precise traceability metamodel dedicated to aspect model weaving. We analyze traceability models, which describe how aspects were woven into a base, to determine the extent to which an aspect has affected the woven model in order to determine how it can be unwoven. Aspect unweaving is finally performed by applying inverse operations of a sub-sequence of the weaving operations in opposite order.

1 Introduction

Since software systems need to be continuously available, their ability to evolve at runtime is a key issue. A very promising approach is to implement such systems as Dynamically Adaptive Systems (DAS), including self-adaptation and dynamic evolution facilities. Modern execution platforms like Fractal [1], OpenCOM [2] or OSGi [3] propose low-level APIs to reconfigure (add/remove/update components, add/remove bindings, etc) a system at runtime. However, with no higher level support, reconfiguration rapidly becomes a daunting and error-prone task to specify, validate, implement and understand. Indeed, implementing a reconfiguration script consists in identifying the components and bindings involved in the reconfiguration, and writing the whole sequence of atomic actions in a correct order. It is really difficult to validate the effect of such a script before actually executing it, detect dependencies or interactions between different scripts, etc.

Recently, some approaches [4,5] use Model-Driven Engineering (MDE) and Aspect-Oriented Modeling (AOM) techniques at runtime (*models@runtime* [6]) to tame the complexity of DAS. Keeping a model synchronized with the running system offers a high-level support for reasoning about the system [4] before actual adaptation. The first step of the dynamic adaptation process consists in selecting, according to the context, the most adapted architectural model. Then,

[⋆] This work was partially funded by the SPLIT project (FNR and CNRS funding) and the DiVA project(EU FP7 STREP, contract 215412, http://www.ict-diva.eu/)

after validation of the model, the running system is automatically adapted by analyzing the selected model. This prevents the designer from writing low-level platform-specific reconfiguration scripts by hand.

However, AOM approaches and tools [7,8,9,10,11] were formerly designed to operate at design-time, where performance (especially time) issues are not so critical. The key problem of current AOM weavers is that in case of small modifications, the adapted model has to be generated by again weaving all the aspects, even those unchanged. In other words, if the configuration is currently composed of n aspects, and if one of them should be "unwoven" (because of a change in the context), we have to restart from a core base model (containing the mandatory elements) and weave the $n-1$ unchanged aspects. More precisely, this means that we should detect the join points of the $n-1$ aspects, matching their associated pointcut model, and weave these aspects. The weaving process itself is efficient: it simply consists in adding or removing some model element and setting attributes and references. However, the join point detection step is more complex: for example, it can rely on graph theory to match sub-graphs in a graph, or rely on Prolog (logic programming) back-end [12] to execute queries on a fact base. In the second case, this requires to transform back and forth the base model, its metamodel (fact) and the pointcut (query) into Prolog artifacts, before actually executing the query. With no real support for aspect unweaving, AOM is not agile enough in an adaptive system context.

This paper shows how aspects can be unwoven, based on a precise traceability metamodel dedicated to aspect model weaving. We analyze traceability models, which describe how aspects were woven into a base, to determine the extent to which an aspect has affected the woven model in order to determine how it can be unwoven. Aspect unweaving is finally performed by applying inverse operations of a sub-sequence of the weaving operations in opposite order.

The remainder of the paper is organized as follows. Section 2 presents an overview of the existing approaches used in this paper. Section 3 introduces essential definitions on the unweaving of aspect models. Section 4 describs a traceability metamodel for aspect model weaving, and shows how a traceability model can be exploited. The main section of this paper is Section 5 which details our unweaving method. Finally, Section 6 presents related work and Section 7 concludes this paper.

2 Background

This section presents first GeKo [13], a generic aspect model weaver, and then an operation-based model construction approach. The objective of this paper is to present how we combined GeKo with this approach to support the unweaving of aspect models.

2.1 GeKo: A Generic Aspect Model Weaver

GeKo [13] is a generic aspect-oriented model composition and weaving approach easily adaptable to any metamodel with no need to modify the domain metamodel or to generate domain specific frameworks. It keeps a graphical representation of the weaving between an aspect model and the base model. It is a tool-supported approach with a clear semantics of the different operators used to define the weaving. The formalization of GeKo allows clearly identifying the sets of removed, added and altered elements.

In this sub-section, we introduce GeKo through an example of class diagram weaving, but GeKo can be used to weave other models such as state diagrams, sequence diagrams, feature diagrams, etc... Fig. 1 shows an example of weav-

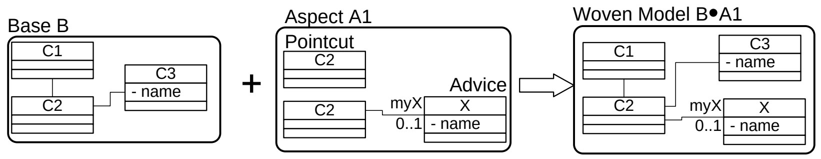

Fig. 1: Example of Class Diagram Weaving with GeKo

ing with GeKo. The result of the weaving of the advice class diagram into the base class diagram is shown in the $Woven\ Model$ of Fig. 1. The weaving process is two-phased. The first step consists in the detection of the match points corresponding to the *Pointcut* diagram. This detection step uses a Prolog-based pattern matching engine which yields a mapping from the pointcut model to the base model for each detected join point. In Fig. 1, the detection yields a mapping from the class $C2$ of the *Pointcut* model to the class $C2$ of the *Base* model. The second step consists in the composition of the advice model with the base model at the level of the match points previously detected (for each match point the advice model is composed). The composition is based on the definition of a mapping between the pointcut and the base model (automatically obtained from the detection step), and a mapping between the pointcut and the advice model (specified by the user). These mappings are defined over the concrete syntax of models by linking model elements. These links are fully generic and do not use any domain-specific knowledge, so that we can define mappings for any domain metamodel. These mappings allow the identification of several sub-sets of objects in the base and advice models characterizing the objects of *base* which have to be kept, to be removed and to be replaced with those of *advice*. Note that in the remainder of the paper, when the mapping between the pointcut and the advice model is obvious, we will omit to specify it.

2.2 Operation-Based Model Construction

In [14], the authors proposed to use a sequence of model construction operations to check consistency rules. In our paper, we present an approach which allows the generation the sequence of construction operation corresponding to the weaving of a sequence of aspects $A_1, A_2, \ldots, A_n$ in a base model B. Consequently, similar to [14], we can use the generated sequence of operations to check consistency rules, but in this paper, we will rather use the operation-based approach to efficiently unweave an aspect from a woven model.

More specifically, in [14] the authors propose to represent models by sequences of elementary construction operations, rather than by the set of model elements they contain. They propose four elementary operations inspired from the *MOF* reflective API [15] : 1) *create(me,mc)* corresponds to the creation of a model element instance *me* of the meta-class *mc*; 2) *delete(me)* corresponds to the deletion of the model element instance *me*; 3) *setProperty(me,p,Values)* corresponds to the assignment of a set of *Values* to the property p of the model element *me*; 4) *setReference(me,r,References)* corresponds to the assignment of *References* to the reference r of the model element *me*.

3 Unweaving Definitions

The *weave* Operation:

Let mp be a match point corresponding to pointcut of an aspect A_i and a base model B, i.e., mp is a place in B where the pattern defined by the pointcut model in A_i matches. The weaving of A_i in B at the level of mp can be defined by a sequence of construction operations:
$weave(A_i, mp) = \sigma^i_{mp,1} \bullet \sigma^i_{mp,2} \bullet ... \bullet \sigma^i_{mp,k}$

Fig. 1 shows a weaving example with class diagrams. Since there is only one match point mp at which the pointcut of A_1 matches, the sequence of construction operation to implement the weaving of A_1 is:
$weave(A_1, mp) = create(X, EClass) \bullet setProperty(X, name, \{X\}) \bullet$
$create(nameAtt, EAttribute) \bullet setProperty(nameAtt, name, \{\text{“}name''\}) \bullet$
$setReference(X, EAttribute, \{nameAtt\}) \bullet create(ref, EReference) \bullet$
$setProperty(ref, name, \{\text{“}myX''\}) \bullet setProperty(ref, EType, \{X\}) \bullet$
$setReference(C2, EReference, \{ref\})$

If A_i matches B h times, the weaving of the aspect A_i into B can be defined by the sequence of construction operations:
$weave(A_i) = weave(A_i, mp_1) \bullet weave(A_i, mp_2) \bullet ... \bullet weave(A_i, mp_h)$
$= \sigma^i_{mp_1,1} \bullet ... \bullet \sigma^i_{mp_1,k} \bullet \sigma^i_{mp_2,1} \bullet ... \bullet \sigma^i_{mp_2,k} \bullet ... \bullet \sigma^i_{mp_h,1} \bullet ... \bullet \sigma^i_{mp_h,k}$

The weaving of a sequence of aspects $A_1, A_2, ..., A_n$ is defined by:
$weave(A_1, A_2, ..., A_n) = weave(A_1) \bullet weave(A_2) \bullet ... \bullet weave(A_n)$

***Undoing* a weave operation:**

For an aspect A_i and a match point mp, we define the *undo* operation $undo(A_i, mp)$ as the execution, in opposite order, of the sequence of *inverse* construction operations of the construction operations of $weave(A_i, mp)$. More formally,
$undo(A_i, mp) = undo(\sigma^i_{mp,1} \bullet \sigma^i_{mp,2} \bullet ... \bullet \sigma^i_{mp,k}) = inverse(\sigma^i_{mp,k}) \bullet inverse(\sigma^i_{mp,k-1}) \bullet$
$... \bullet inverse(\sigma^i_{mp,1})$, where:

$\sigma_{mp,j}$	$inverse(\sigma_{mp,j})$
$create(me, mc)$	$delete(me)$
$delete(me)$	$create(me, mc)$ (in practice, mc is easily obtained from the sequence of construction operations)
$setProperty\ (me, p, value)$	if $\exists setProperty(me, p, value') \in weave(A_{k,k<i})$ then $setProperty(me, p, value')$ else $setProperty(me, p, \emptyset)$
$setReference\ (me, r, ref)$	if $\exists setReference(me, r, ref') \in weave(A_{k,k<i})$ then $setReference(me, r, ref')$, else $setReference(me, r, \emptyset)$

If we note $mp_k, k \in \{1, \dots, h\}$ the match points corresponding to the weaving of an aspect A_i in a base B, we can extend the notion of *undo* to all the match points by:
$undo(A_i) = undo(A_i, mp_h) \bullet undo(A_i, mp_{h-1}) \bullet \dots \bullet undo(A_i, mp_1)$

***Unweaving*:**

Let $A_1, A_2, ..., A_n$ be a sequence of aspects that have been woven into a base model B to result in a woven model BW. Unweaving of an aspect A_i from BW should result in a model that is equivalent to the model obtained by starting again with the base model B and weaving all aspects into B again in the same order, but omitting A_i. More formally, $\forall i \in \{1, ..., n\}$,

$$unweaving(A_i) = \begin{cases} weave(A_2, A_3, ..., A_n) & i = 1 \\ weave(A_1, A_2, ..., A_{i-1}, A_{i+1}, ..., A_n) & 1 < i < n \\ weave(A_1, A_2, ..., A_{n-1}) & i = n \end{cases}$$

4 Aspect Traceability Metamodel

4.1 Traceability Metamodel for GeKo

Weaving in GeKo is asymmetric, i.e., the weaving process is performed by applying a set of operations on a *base model*[4]. During the model weaving process, to compose an advice model with a base model, GeKo can: (1) *Remove* a model element from the base model; (2) *Add* a model element to the base model. The added model element is defined in the aspect's advice model; (3) *Replace* a model element of the base model by a model element of the aspect's advice model. This *replace* operation can be considered as a sequence of *remove* and *add* operations (remove the replaced element and add the element which replaces it); (4) *Update* the *properties* of a base model element (e.g. change the name of a model element); (5) *Update* the *references* that a base model element has towards other model elements.

We propose to keep a trace of the application of these operations as the weaving takes place. For this, we defined the traceability metamodel for aspect model weaving presented in Fig. 2. The $WovenAspectSequence$ class is the root class of the metamodel presented in Fig. 2. It contains a sequence of $AspectWeaving$. An $AspectWeaving$ references a $Base$, a $Result$ and an $Aspect$ model. An $Aspect$ model is composed of a $Pointcut$ and an $Advice$ model. All these models are defined by a list of $ModelElements$. The $AspectWeaving$ class is also associated with a list of $PointcutMatches$. Each $PointcutMatch$ stores the list of base model elements that were used to obtain this particular match when matching the pointcut of the aspect model to the base model. Each of the referenced base model elements is essential, i.e., if only one were omitted, the pointcut model would not match the base anymore. Finally, the class $PointcutMatch$ also stores the effects of the weaving of the advice model of the aspect at this particular match point. $PointcutMatch$ is associated with a sequence of $ConstructionOperations$. There are five types of possible operations corresponding to the five GeKo operations: $Replace$, $Remove$, Add, $UpdateReference$, $UpdateProperty$.

4.2 Using the Traceability Model

Once a sequence of aspects $A_1, A_2, \ldots, A_n$ is woven with a base model B to produce a woven model BW, the information stored in the traceability model contains the complete trace of operations that transformed B into BW. To re-execute the

[4] Since aspects can be applied to other aspects, the GeKo base model can, of course, be any model, even an advice model of some other aspect.

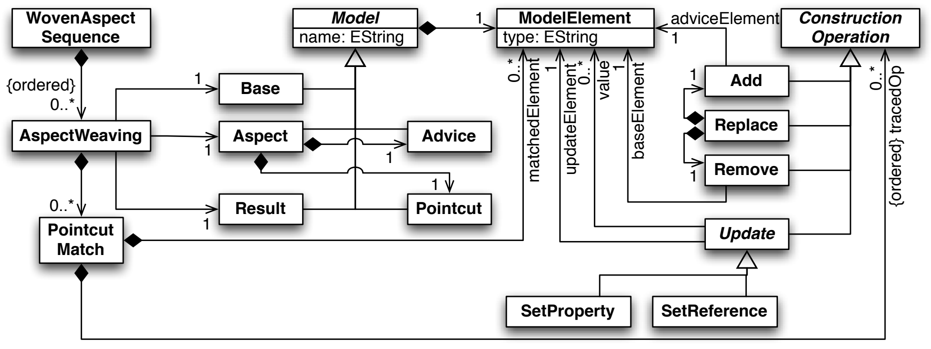

Fig. 2: Traceability Metamodel

weaving, it suffices to start with B, and execute the associated sequence of elementary construction operations, which can be easily obtained by concatenating the sequence of *tracedOp* for each aspect weaving and for each associated match point. This sequence of construction operations is used in the algorithms presented in the next section. [5]

The traceability model can also be used to determine the *impact* that the weaving of an advice model at a match point had on the final woven model. We define the impact of a match point of the pointcut model in A_i as all the model elements in the final woven model that were directly or indirectly changed because of the weaving of the advice model of A_i at the match point.

For instance, if an aspect A_i adds a model element, then this model element might be used in a match point of a following aspect $A_{k,k>i}$, which again triggers the weaving of the advice model of A_k. Hence, the impact of weaving the advice model of A_i is not limited to the changes specified in the advice model of A_i, but also includes the changes specified in the advice model of A_k. The same reasoning can be applied to A_k as well, and hence the impact of a weaving of A_i at a match point can potentially include changes in all advice models of aspects $A_{k,k>i}$.

To determine the impact of a pointcut match, not only *Add* operations have to be considered. $SetReference$ and $SetProperty$ operations can also result in the creation of a match point of subsequent aspects $A_{k,k>i}$. For instance, the $SetProperty$ operation can be used by an aspect A_1 to change the value of the *balance* field of an object to 200. A following aspect A_2 might declare a pointcut which matches for all objects that have a *balance* attribute with a value ≥ 100. In this case, the impact of A_1 should only include the changes specified in the advice model of A_2 if the previous value of *balance*, i.e., the value that *balance* had *before* weaving A_1, was < 100. The detailed algorithm that calculates the impact of a match point *mp* of an aspect A_i is shown in Alg. 1.

5 Using the Traceability Model for Unweaving of Aspects

This section presents the core contribution of our paper. It shows how the tracing information gathered during the model weaving (see section 4) can be used to unweave aspects from a woven model in an efficient way.

[5] Note that the construction operations defined in the traceability metamodel are implemented using the MOF primitives presented in subsection 2.2.

```
Algorithm 1: $Impact(A_i, mp)$
  Input: the aspect $A_i$, the match point $mp$, the traceability model corresponding
         to the weaving of the aspects $A_1, A_2, \ldots, A_n$ in a base model $B$
  Output: the set of pairs $(A_k, mp')$ where the match point $mp'$ of the aspect
          $A_{k,k>i}$ is impacted by the weaving of $A_i$ at the match point $mp$
  $k \leftarrow i + 1$
  while $k \leq n$ do
      foreach operation $\sigma \in weave(A_i, mp)$ do
          foreach match point $mp'$ of $A_k$ do
              if $\sigma == add(elt, eltType)$ and elt is shared with $mp'$ then
                  $impact \leftarrow impact \cup (A_k, mp')$
              end
              if $\sigma == setReference(elt, ref, \{eltList\})$ and elt and $ref$ are shared
              with $mp'$ and the previous set of values for the reference $ref$ is not
              included in the set of values for the reference $ref$ of elt of the
              pointcut of $A_k$ then
                  $impact \leftarrow impact \cup (A_k, mp')$
              end
          end
      end
      $k \leftarrow k + 1$
  end
```

Let $A_1, A_2, ..., A_n$ be a sequence of aspects woven into a base model B resulting in a woven model BW. As presented in the definitions section, unweaving an aspect from BW is equivalent to re-weaving all aspects $A_1...A_n$ into B except for A_i. Re-weaving is, however, very inefficient. Not only does the tool have to re-execute all construction operations defined by the $n - 1$ advice models of the aspects $A_{j,j \neq i}$, but it also has to re-execute the pattern matching algorithm that searches for match points based on the patterns defined in the $n - 1$ pointcut models.

The technique presented in this paper allows a tool to unweave an aspect A_i from BW without having to re-weave all the aspects $A_{j,j \neq i}$ into B. Depending on the nature of the relation between the aspect A_i and the aspects $A_{k,k>i}$ that were woven into the base model after A_i, unweaving A_i is more or less complicated. In the following discussion, we distinguish 3 different cases. At this point, the reader is reminded that our solution is based on the use of the generic weaver called GeKo, which performs aspect weaving using the operations 1) *add* model element, 2) *remove* of a model element, 3) *replace* model element (which can be seen as a remove followed by an add), 4) *set property*, and 5) *set reference*.

5.1 Case 1, Independent Aspects:

Informal description: In the most advantageous case, A_i is independent of the aspects $A_{k,k>i}$ that were woven after A_i to obtain BW. This situation occurs when A_i neither introduced model elements which were used in a match point of one of the aspects $A_{k,k>i}$, nor removed model elements which could have formed a match point for a $A_{k,k>i}$, nor changed any properties or references that were used or could have been used in a match point of one of the aspects $A_{k,k>i}$.

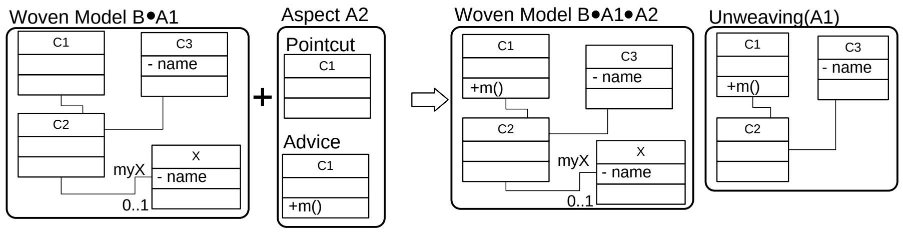

Fig. 3: Weaving of A_2 and Unweaving of A_1 (previously presented in Fig. 1)

Fig. 3 shows an example illustrating this case. The example presents the weaving of an aspect A_2 into the model obtained after the weaving of the aspect A_1 already presented in Fig. 1. The resulting model is $B \bullet A_1 \bullet A_2$. In this example, A_1 is independent from A_2, because the model elements introduced (the class X) and the changes to model elements (adding of the reference to $C2$) are not part of the match point of A_2 (which matches on the class $C1$). Also, A_1 does not remove any model elements from B.

Unweaving of Independent Aspects: Unweaving of aspect A_i simply consists in undoing the weave operation, i.e., in applying, for each match point, the inverse construction operations in opposite order of the construction sequence defined by $weave(A_i, jp)$. More formally:
If match point mp of A_i is independent of $A_{k,k>i}$:$unweave(A_i, mp) = undo(A_i, mp)$
$\Rightarrow$ If A_i independent of $A_{k,k>i}$: $unweave(A_i) = undo(A_i) = \forall mp_{A_i} : undo(A_i, mp)$.
Therefore, in Fig. 3, the unweaving of A_1 consists in applying $undo(A_1)$, i.e., applying inverse operations in opposite order of the construction sequence $weave(A_1)$.

$unweave(A_1) = undo(A_1) = setReference(C2, EReference, \emptyset) \bullet$
$setProperty(ref, EType, \emptyset) \bullet setProperty(ref, name, \emptyset) \bullet delete(ref) \bullet$
$setReference(X, EAttribute, \emptyset) \bullet setProperty(nameAtt, name, \emptyset) \bullet$
$delete(nameAtt, \emptyset) \bullet setProperty(X, name, \emptyset) \bullet delete(X)$

5.2 Case 2, General Aspects:

Informal description: In the worst case, when the weaving of an aspect A_i removes or changes model elements in the base model, it is possible that these elements could have been used to form a match point of an aspect $A_{k,k>i}$. As a result, the unweaving of the aspect A_i could introduce completely new match points for the following aspects $A_{k,k>i}$. In this case, the unweaving of A_i cannot be done by simply applying a sequence of undo operations. Unfortunately, the pattern matching operation that detects match points corresponding to the pointcut models in the aspects $A_{k,k>i}$ has to be launched again.

Fig. 4 shows an example of Final State Machine (FSM) weaving illustrating this case. The example presents the successive weaving of the aspect A_1 and A_2 into the base model B. The resulting model is $B \bullet A_1 \bullet A_2$. The weaving of aspect A_1 consists in replacing the state c by a state e and by removing the state a. The weaving of aspect A_2 consists in replacing the state a by a state b with a loop transition. After the weaving of A_1, A_2 matches only once, but without A_1, the pointcut of A_2 would also match against the first a state that was removed when

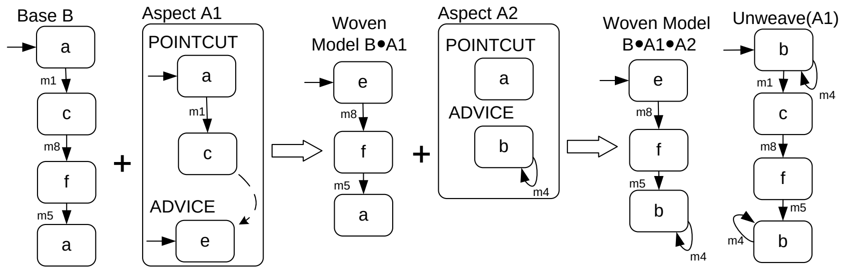

Fig. 4: Weaving of A_1 and A_2 into B, and Result of the Unweaving of A_1

A_1 was woven. This example shows that the only solution to unweave A_1 is first to unweave A_2, then to unweave A_1 and finally to weave A_2 again.

Unweaving General Aspect A_i: To unweave a general aspect A_i the idea is to first compute index $j > i$ such that A_i is not general for the aspects $A_{h,i \leq h<j}$. The second step consists in unweaving of the aspects $A_{k,k \geq j}$ in the opposite order of the sequence of weaving, i.e., A_n first, then $A_{n-1}, \ldots, A_j$. For these unweavings, since the aspect unwoven is always the last aspect that had been woven, the unweaving operation corresponds to the *undo* operation. The next step is to unweave A_i. The final step consists in weaving of the aspects $A_{k,k \geq j}$ in the same order as the initial sequence of weaving.

Depending on the value of i and j, it might be faster to start from scratch, i.e., start with B and re-execute the weave operations of the aspects $A_1 \ldots A_{i-1}$ stored in the traceability model rather than to unweave aspects $A_j \ldots A_n$ and A_i. The cut off values of i and j at which it is better to re-weave than to unweave depends heavily on the number of match points of each aspect, and the number of construction operations needed to implement the weaving of each match point. If the length of the sequence of construction operations for weaving aspects $A_1 \ldots A_{i-1}$ is smaller than the length of the sequence of operations for weaving $A_i, A_j \ldots A_n$, then re-weaving is more efficient than unweaving.

5.3 Case 3: Additive Aspects

Informal description: Some aspects A_i are not general, i.e., they did not remove or alter elements which could have been used in a match point of a following aspect $A_{k,k>i}$, but are also not independent, because they added or changed model elements which were later on used in a match point of at least one of the $A_{k,k>i}$. We call these *additive* aspects.

Fig. 5 presents an example of this case. A_1 introduces a message $m4$ from $O2$ to $O3$ after an exchange of messages $m1, m2$ between $O1$ and $O2$. A_2 introduces an message $m5$ after any message $m4$. If we consider the sequence of weaving $A_1 \bullet A_2$ as shown in Fig. 5, A_1 is clearly an additive aspect. The message $m4$ introduced by A_1 is matched by the pointcut model of aspect A_2 and hence creates a match point. Also, A_1 does not remove any model elements.

Unweaving of Additive Aspects: In the case of an additive aspect A_i, the unweaving of the aspect does not simply consist in undoing the weave operation of A_i as for the case of independent aspects, because some elements added by the weaving of A_i have been used to form match points of aspects $A_{k,k>i}$, and hence

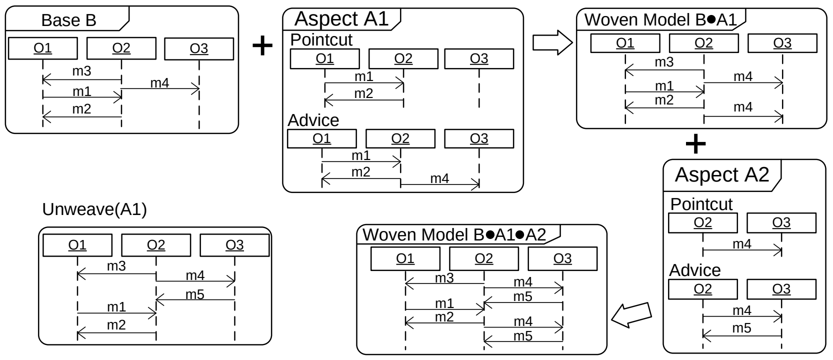

Fig. 5: Weaving of A_1 and A_2 into B, and Result of the Unweaving of A_1

resulted in further changes to the model. Therefore, the unweaving operation has to also undo the weaving of all advice of aspects $A_{k,k>i}$ that were woven because of a pointcut match that contained elements that A_i added or changed.

Let us consider the example of Fig. 5. To unweave the aspect A_1, we have to remove both the operations directly related to the weaving of A_1 (i.e., the introduction of a message $m4$) and the operations related to the match point of A_2 formed by elements introduced by A_1. In Fig. 5, the pointcut model of A_2 matched twice in the model $B \bullet A_1$, once on each message $m4$. However, only the second match is due to A_1, and therefore only the second introduction of a new message $m5$ has to be undone.

More formally, to unweave an aspect A_i, we apply the algorithm described in Alg. 2 for all the match points of A_i. It describes that, to unweave an additive aspect A_i, we have to also recursively unweave the *impacted* match points of aspects $A_{k,k>i}$. These impacted match points can, of course, be of any type, i.e., independent, additive, or general.

5.4 Classification of Aspects

This section presents how our tool classifies a match point of an aspect into one of the 3 cases described above, followed by the complete algorithm of classification.

Conditions for Detecting General Aspects: To determine if mp of A_i is *general*, we must check if A_i modified or removed model elements that could have created a match point for one of the pointcuts of one of the following aspects $A_{k,k>i}$. Note that when the advice model in A_i *removes* an element which corresponds to an element of a match point of an aspect $A_{k,k>i}$, mp is immediately classified as general. However, as specified in Algorithm 3, additional conditions have to be respected in the case where A_i only modifies (with $setReference$ or $setProperty$) an element which corresponds to an element of a pointcut of an aspect $A_{k,k>i}$. For instance, when A_i modifies the reference of an element elt, this modification cannot remove a match point if elt was previously added by A_i.

Condition for Detecting Additive Aspects: To determine if mp of A_i is *additive*, we check if A_i is not general and if A_i adds and modifies any model elements which are used to form a match point of a following aspect $A_{k,k>i}$. Note that when A_i *adds* an element used to form a match point mp' of an aspect $A_{k,k>i}$, mp' cannot be a match point of A_k anymore when A_i is unwoven. However, when

Algorithm 2: $additiveUnweave(A_i, mp, A_{k,1 \leq k \leq n})$

```
Input: the aspect $A_i$, the additive match point $mp$ to unweave, the traceability
        model corresponding to the weaving of the sequence of aspects
        $A_1, A_2, \ldots, A_n$ in a base model $B$. The sequence $A_{k,1 \leq k \leq n}$ is such as
        there is no genral match point $mp'$ of an aspect $A_j$ impacted by $A_i$ or
        recursively by a match point of an aspect impacted by $A_j$
Output: the sequence of unweaving operations $\sigma$

$\sigma \leftarrow undo(A_i)$
Let $impact$ be the set of model elements added or modified by the advice model
of $A_i$ when applied to $mp$ (see Alg. 1)
$j \leftarrow i + 1$
while $j \leq n$ do
    foreach match point $mp'$ of the aspect $A_j$ do
        Let $me$ be the set of model elements associated with the match point $mp'$
        if $me \cap impact \neq \emptyset$ then
            $\sigma \leftarrow unweave(A_j, mp') \bullet \sigma$
        end
    end
    $j \leftarrow j + 1$
end
```

A_i modifies an element that used to form a match point mp' of an aspect $A_{k,k>i}$ (with *setReference* or *setProperty*), mp' can still be a match point even after A_i is unwoven. This is the case if the *previous* value of the property or reference being modified also resulted in the creation of the same match point.

Condition for Detecting Independence: mp of A_i is *independent*, if *mp* is neither general nor additive.

More formally, to classify the match point of an aspect A_i with respect to the following aspects $A_{k,k>i}$, we apply Alg. 3. Note that for space reasons, Alg. 3 does not show how to handle the *setProperty* operation, but the conditions are exactly the same as for the $setReference$ operation. To extend this classification for a mach point of A_i to the aspect A_i itself, we use the following rules: 1) A_i is general if for all match points of A_i, at least one match point is general; 2) A_i is additive if A_i is not general and at least one match point of A_i is additive; 3) A_i is independent if A_i is neither general nor additive.

5.5 Complete Unweaving Algorithm

To unweave A_i, the first step consists in the determination of the lowest index $j \geq i$ of an aspect that contains a *general* match point *mp* that needs to be unwoven because of the unweaving of A_i. In other words, j is the index of the first aspect A_j in the sequence of aspects whose match point *mp* cannot be unwoven by applying a sequence of undo operations.

This is done by the *lowestGeneralIndex* algorithm presented in Alg. 4. This index is then used in the general unweaving algorithm presented in Alg. 5. The first foreach loop unweaves the general aspects. The *lowestGeneralIndex* allows us to ensure that for all unweaving operations in the second foreach loop (even

Algorithm 3: $classify(A_i, mp)$

```
Input: the aspect $A_i$, the match point $mp$, the traceability model corresponding
       to the weaving of the sequence of aspects $A_1, \ldots, A_n$ in a base model $B$
Output: Classification of the application of the advice model of $A_i$ at the match
        point $mp$ and the aspects $A_{k,k>i}$

if $i = i$ then
    $mp$ is independent
else
    if $\exists delete(elt) \in weave(A_i, mp)$ such as elt corresponds to a model element of
    a pointcut of $A_{k,k>i}$ OR $\exists setReference(elt, ref, \{eltList\}) \in weave(A_i, mp)$
    such that elt is shared with a match point of $A_{k,k>i}$ and elt is an element not
    added by $A_i$ and the previous set of values for the reference $ref$ is not
    included in the set eltList then
        mp is general
    else
        if $\exists add(elt, eltType) \in weave(A_i, mp)$ such as elt is shared with a match
        point of $A_{k,k>i}$ OR
        $\exists setReference(elt, ref, \{eltList\}) \in weave(A_i, mp)$ such as elt and $ref$
        are shared with a match point of $A_{k,k>i}$ and the previous set of values for
        the reference $ref$ is not included in the set of values for the reference $ref$
        of elt of the match point of $A_k$ then
            mp is additive
        else
            mp is independent
        end
    end
end
```

for the operations recursively called), the match point *mp* is either independent or additive, but never general. As a result, the unweaving operation consists in either the undo operation or the operation described in Alg. 2. Finally, the last for loop executes the necessary re-weaving, if any.

6 Related Work

Although the method described in this paper is applied in the context of the GeKo aspect model weaver, the same ideas can easily be generalized to other model weavers (such as [9,16,17,18]), once an appropriate traceability model is constructed.

In [17], the authors present an interesting way to modify models before and after their composition, by means of the use of a language of directives. This support is not automatised, but in our approach, the directive language could be used to apply the generated unweaving sequence of elementary operations.

To the best of our knowledge and belief, no Aspect-Oriented Modeling approach provides support for aspect model unweaving. At the platform level, however, some approaches provide support for weaving and unweaving.

FAC (Fractal Aspect Component) [19] is an open-source aspect-oriented extension to the Fractal component model [1]. It combines Component-Based Software

```
Algorithm 4: $lowestGeneralIndex(A_i)$
Input: the aspect $A_i$, the traceability model corresponding to the weaving of the
       sequence of aspects $A_1, \ldots, A_n$ in a base model $B$
Output: the lowest index of a general aspect impacted by $A_i$

Let $j$ be the smallest index $> i$ such that $A_i$ is not general for the sequence of
aspects $A_{h,i \leq h < j}$
lowestGeneralIndex $\leftarrow j$
foreach couple $(A_h, mp')$ impacted by the match points of $A_i$ do
    if $\exists l$ such that $h < l < j$ and the match point $mp'$ of $A_h$ is general for a
    match point of $A_l$ then
        lowestGeneralIndex $\leftarrow l$
    end
end
```

```
Algorithm 5: $unweave(A_i)$
Input: the aspect $A_i$, the traceability model corresponding to the weaving of the
       sequence of aspects $A_1, A_2, \ldots, A_n$ in a base model $B$

$j \leftarrow lowestGeneralIndex(A_i)$
for $k = n \ldots j$ do
    apply $undo(A_k)$
end
foreach match point mp of the aspect $A_i$ do
    if classify($A_i$,mp) = independent then
        apply $undo(A_i, mp)$
    else
        $additiveUnweave(A_i, mp, A_{k,1 \leq k \leq j-1})$
    end
end
for $k = j \ldots n$ do
    $weave(A_k)$
end
```

Development (CBSD) and Aspect-Oriented Programming (AOP) by integrating CBSD notions into AOP, and vice-versa. FAC introduces new aspect-oriented structures into the Fractal platform: Aspect Component (AC), Aspect Domain (AD) and Aspect Binding (AB). An Aspect Component is a regular component that encapsulates a crosscutting concern providing advice pieces of code as services. An Aspect Binding is a binding that links an AC to other components. Finally, an AC and all the aspectized components bound via ABs constitute an Aspect Domain (AD). Note that FAC leverages the notion of shared components provided by Fractal to allow components to be contained in several ADs. Basically, weaving an aspect component consists in creating the composite component corresponding to the AD, containing the components of the aspect itself as well as the components impacted by the aspect (still contained by their former container), and introducing bindings. Unweaving the aspect consists in removing all these previously introduced elements. However, no real description of the unweaving process is provided, especially when an aspect depends on other aspects.

Very similar to FAC is AOpenCOM [20], which provides aspect-oriented construction for the OpenCOM component model [2]. Again, very few details are provided about the unweaving process.

CaesarJ [21] extends Java with aspect-oriented constructs. It combines AspectJ-like constructions (pointcut/advice) with advanced modularization and composition techniques such as virtual classes and mixins. Aspects can dynamically be deployed or undeployed on all the join points currently identified in the JVM matching the pointcut. Similarly to above mentionned approaches, no detail is given on the unweaving strategy, especially when aspects are interacting.

In this paper, we have proposed clear and formalized unweaving strategies at the model level, independently from any metamodel. Depending on how an aspect interacts with the other, we precisely determine the way the aspect should be unwoven. With the emergence of the notion of *models@runtime*, it becomes important to optimize approaches and tools usually used at design-time. In previous work [4], we use AOM and MDE in order to manage complex dynamic software product lines at runtime. Each dynamic feature of the system is represented as an aspect model [22] (an architecture fragment), which is selected depending on the context. When the context changes, new aspects can be selected while others are discarded. Working at the model level provides a better basis for reasoning, validation, and automation [4,5]. However, without support for unweaving, we had to systematically restart from a core base model and weave again all the aspects, which is very inefficient.

7 Conclusion

In this paper we have presented a method to efficiently unweave an aspect A_i from a sequence of aspects $A_1 \ldots A_n$ woven into a base model B. Our method is based on the use of a traceability model recording construction operations at weave-time. The traceability model allows the determination of the relation between A_i and the following aspects. According to this relation, which is either *independent*, *additive* or *general*, the unweaving is more or less complicated. If no general relationship is detected, unweaving can be performed by directly applying a set of undo operations to the woven model.

Although the method described in this paper is applied in the context of the GeKo aspect model weaver, we believe the same ideas can easily be applied in other tools, once an appropriate traceability model is constructed.

In the future, we plan to apply and evaluate the performance of our unweaving method in the context of dynamic, adaptive systems.

References

1. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.: The FRACTAL Component Model and its Support in Java. Software Practice and Experience **36**(11-12) (2006) 1257–1284
2. Blair, G., Coulson, G., Ueyama, J., Lee, K., Joolia, A.: Opencom v2: A component model for building systems software. In: IASTED Software Engineering and Applications, USA (2004)

3. The OSGi Alliance: OSGi Service Platform Core Specification, Release 4.1 (May 2007) http://www.osgi.org/Specifications/.
4. Morin, B., Barais, O., Nain, G., Jézéquel, J.: Taming Dynamically Adaptive Systems with Models and Aspects. In: ICSE'09, Vancouver, Canada (May 2009)
5. Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.M., Solberg, A., Dehlen, V., Blair, G.: An Aspect-Oriented and Model-Driven Approach for Managing DynamicVariability. In: ACM/IEEE MoDELS'08, Toulouse, France (October 2008)
6. N. Bencomo, G. Blair, R.F.: Proceedings of the international workshops on models@run.time (2006-2008) (2006-2008)
7. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design: The Theme Approach. Number ISBN: 0-321-24674-8. Addison Wesley (2005)
8. Whittle, J., Jayaraman, P.: Mata: A tool for aspect-oriented modeling based on graph transformation. In: AOM at Models'07. (2007)
9. Groher, I., Voelter, M.: Xweave: Models and aspects in concert. In: AOM Workshop'07 at AOSD. (March 12 2007)
10. Kienzle, J., Abed, W.A., Klein, J.: Aspect-oriented multi-view modeling. In ACM, ed.: AOSD'09, Charlotteville, Virginia, USA (March 2009) 87–98
11. Klein, J., Hélouet, L., Jézéquel, J.M.: Semantic-based weaving of scenarios. In: AOSD'06, Bonn, Germany, ACM (2006) 27–38
12. Ramos, R., Barais, O., Jézéquel, J.M.: Matching Model Snippets. In: MoDELS'07: 10th Int. Conf. on Model Driven Engineering Languages and Systems, Nashville USA (October 2007) 15
13. Morin, B., Klein, J., Barais, O., Jezequel, J.M.: A generic weaver for supporting product lines. In: Early Aspects Workshop at ICSE, Leipzig, Germany (May 2008)
14. Blanc, X., Mounier, I., Mougenot, A., Mens, T.: Detecting model inconsistency through operation-based model construction. In: ICSE 2008, Leipzig, Germany, ACM/IEEE (2008) pp. 511–520
15. OMG: Mof core specification , v2.0. OMG Document number formal/2006-01-01 (2006)
16. Whittle, J., Jayaraman, P.: MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation. In: AOM@MoDELS'07: 11th International Workshop on Aspect-Oriented Modeling, Nashville TN USA (Oct 2007)
17. Reddy, R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., Song, E., Georg, G.: Directives for composing aspect-oriented design class models. Transactions on Aspect-Oriented Software Development (TAOSD) **LNCS 3880** (2006) 75–105
18. Sanchez, P., Fuentes, L., Stein, D., Hanenberg, S., Unland, R.: Aspect-oriented model weaving beyond model composition and model transformation. In: MoDELS'08, LNCS 5301 (2008) pp. 766–781
19. Pessemier, N., Seinturier, L., Coupaye, T., Duchien, L.: A Model for Developing Component-based and Aspect-oriented Systems. In: SC'06: 5th International Symposium on Software Composition. Volume 4089 of LNCS., Vienna, Austria (2006) 259–273
20. Surajbali, B., Coulson, G., Greenwood, P., Grace, P.: Augmenting reflective middleware with an aspect orientation support layer. In: Proceedings of the 6th Workshop on Adaptive and Reflective Middleware (ARM 2007). (2007)
21. Aracic, I., Gasiunas, V., Mezini, M., Ostermann, K.: An Overview of CaesarJ. In: Transactions on Aspect-Oriented Software Development I. Volume 3880 of LNCS. (2006) 135–173
22. Perrouin, G., Klein, J., Guelfi, N., Jézéquel, J.M.: Reconciling Automation and Flexibility in Product Derivation. In: 12th International Software Product Line Conference, Limerick, Ireland, IEEE Computer Society (2008) 339–348