A Generic Weaver for Supporting Product Lines

Brice Morin, Jacques Klein, Olivier Barais, Jean-Marc Jézéquel

A Generic Weaver for Supporting Product Lines

Brice Morin
IRISA / INRIA Rennes
Campus de Beaulieu
35042 Rennes Cedex, France
bmorin@irisa.fr

Jacques Klein
University of Luxembourg
Campus Kirchberg
L-1359 Luxembourg,
Luxembourg
jacques.klein@uni.lu

Olivier Barais,
Jean-Marc Jézéquel
IRISA / Université Rennes 1
Campus de Beaulieu
35042 Rennes Cedex, France
barais@irisa.fr,
jezequel@irisa.fr

ABSTRACT

Aspects have gained attention in the earlier steps of the software life-cycle leading to the creation of numerous ad-hoc Aspect-Oriented Modeling (AOM) approaches. These approaches mainly focus on architecture diagrams, class diagrams, state-charts, scenarios or requirements and generally propose Aspect-Oriented composition mechanisms specific to a given kind of models defined by its own meta-model. Recently, some generic AOM approaches propose to extend the notion of aspect to any domain specific modelling language (DSML). In this trend, this paper presents GeKo. GeKo has the following properties. i) It is a generic AOM approach easily adaptable to any DSML with no need to modify the domain meta-model or to generate domain-specific frameworks. ii) It keeps a graphical representation of the weaving between an aspect model and the base model. iii) It is a tool-supported approach with a clear semantics of the different operators used to define the weaving. GeKo relies on the definition of mappings between the different views of an aspect, based on the concrete (graphical) syntax associated to the DSML. To illustrate GeKo, we derive, from the Arcade Game Maker Pedagogical Product Line, a new product in which new features are woven into the Product Line models.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Design Tools and Techniques

General Terms: Design, Theory

Keywords: Aspect-Oriented Modeling, Model-Driven Engineering, Software Product Line, Composition, Weaving, Metamodel

1. INTRODUCTION

The Aspect-Oriented Software Development (AOSD) paradigm first appeared at the code level with concern-specific aspect oriented language such as COOL [7] for the synchronization or RIDL [7] for specifying the remote data transfer in systems. It is next popularized with the well-known general purpose aspect-oriented languages AspectJ [4]. AspectJ provides means to encapsulate cross-cutting concerns that were not well modularized in the Object-Oriented paradigm. To produce the system, all the concerns are woven into the base program. Over the last decade, aspects have gained attention in the earlier steps of the software life-cycle leading to the creation of numerous ad-hoc Aspect-Oriented Modeling (AOM) approaches. These approaches mainly focus on architecture diagrams, class diagrams, state-charts, scenarios or requirements and generally propose Aspect-Oriented composition mechanisms specific to a given kind of models defined by its own metamodel. Concepts manipulated, tooling and documentation of these approaches are not homogeneous. Consequently, users which design a system with different kinds of models have to adapt to different methodologies to weave aspects.

Recently, some generic AOM approaches (*e.g.*, Kompose [1, 2], Generic SmartAdapters [8, 9] or MATA [3, 14]) propose to extend the notion of aspect to any domain specific modeling language (DSML). These generic approaches propose means to extend the notion of aspect to any kind of models with a well-defined metamodel. Consequently, it is possible to use one of these approaches to weave aspects into any model instead of using different ad-hoc approaches.

In this paper, we presents GeKo our generic AOM approaches. GeKo has the following properties. i) It can easily be adapted to any DSML with no need to modify the domain metamodel or to generate domain-specific frameworks. ii) It keeps a graphical representation of the weaving between an aspect model and the base model. iii) It is a tool-supported approach with a clear semantics of the different operators used to define the weaving. GeKo relies on the definition of mappings between the pointcut elements and advice elements, based on the concrete (graphical) syntax associated to the DSML. To illustrate GeKo, we derive, from the Arcade Game Maker Pedagogical Product Line [15] a new product in which new features are woven into the product line models.

This paper is organized as follows. Section 2 presents a background on generic AOM. In Section 3 we extend the case study using GeKo. We formalize our approach in Section 4 and give the precise semantic of our weaver. Finally, Section 5 concludes and presents future works.

2. BACKGROUND ON GENERIC AOM

This section presents an overview of generic Aspect-Oriented Modeling approaches. Generic means that these approaches can be applied or adapted to any well-defined metamodel.

EA'08, May 12, 2008, Leipzig, Germany.

2.1 Kompose

Kompose [1, 2] is an AOM approach based on the systematic merging of matching elements. It generalizes the concepts proposed by France *et al.* in [12, 13] in the context of class diagrams, for any metamodel. Matching is realized by comparing signatures. For each mergeable element, users can customize the signature. For example, the signature of a method may be its name, as well as the type of its parameters. This allows to define precisely how to match elements, rather than simply comparing their names.

The merging phase is classic and is implemented in a generic way, using introspection. The elements that match are merged into a single element while the elements with no counter-part are added into the resulting model. Before merging, users can define some pre-directives to prepare the models they want to merge. For example, they can modify the signature of identical concepts to force the merge. After merging, it is also possible to define post-directives in order to finalize the weaving. For example, users may want to remove some elements *e.g.*, an association between two classes. Both pre- and post-directives are specific to the models to merge.

Kompose is clearly a symmetric approach that aims at merging different views of a given system. If the views are homogeneous the merging may be fully automated and require few directives. But, systematic merging is not the only way of composing aspects, as argued for example in [5, 14]. In Kompose, users can customize the matching process by defining the signature of the mergeable elements but cannot customize the merging process. Like other symmetric approaches, Kompose does not offer mechanisms to make an aspect reusable in different base models.

2.2 SmartAdapters

SmartAdapters is an AOM approach based on adaptation. It has formerly been applied to Java programs [6] and class diagrams [5]. More recently it has been generalized for any metamodel [8, 9]. An aspect is composed of three parts: *i)* a target model specifying **where** the aspect will be woven, *ii)* a reusable concern specifying the structure of the aspect (**what** will be woven) and *iii)* a composition protocol describing **how** to weave the aspect. The composition protocol is supported by adaptations that are basic weaving operations between elements from the target model and element from the reusable concern, automatically generated from a domain-metamodel. They directly manipulate the concepts defined in the metamodel.

The target model is a model fragment describing what the aspect expects from any base model. It is equivalent to the notion of pointcut but target models do not directly refer to particular base model elements. In fact, these target models are designed with a pattern matching framework [11] allowing users to design pointcuts with roles that only contain elements relevant for the composition, without considering all the constraints of the former metamodel. Then, a pattern matching engine search all the places (join points) that match the target model. Finally, user selects a subset of these join points to choose where to apply the adapter and weave the aspect.

The generic SmartAdapter approach [8, 9] is an asymmetric approach that aims at integrating reusable concerns into different base models. The composition of the aspect should totally be explicited by the designer using fine grain adaptations that manipulates meta-level concepts. Currently, it is difficult to realize a merge with no higher-level adaptations, because the generated adaptations only manipulates elements that are explicitly present in the target model and in the reusable concern. In this case, we have to match all the elements needed to realize the merge, making the target model and the composition protocol complex. However, users can easily customize the framework and propose additional adaptations.

2.3 MATA

MATA [3, 14] (Modeling Aspects Using a Transformation Approach) is an AOM approach for aspect weaving in UML models, based on the graph transformation formalism.

The idea of MATA is similar to the idea of the SmartAdapters approach *i.e.*, MATA models describe where and how aspects are woven into base models. They describe both the Left-Hand-Side and the Right-Hand-Side of their graph rules in the same model, using the UML stereotypes: $\ll$ create $\gg$ and $\ll$ delete $\gg$ for creating or deleting model elements in cascade. A $\ll$ context $\gg$ stereotype can be associated to model elements to specify that they should not be affected when creating/deleting their container. These graph rules are defined over the concrete syntax, making MATA intuitive to use. The concepts of MATA may be generalized to other well-defined metamodels. However, the use of stereotypes makes this approach quite specific to the UML metamodel.

Every MATA model is converted into a graph rule. The base model where the aspect will be woven is converted into a graph and the rule is executed onto this graph. Graph theory and tools allow them to perform some analysis such as aspect/feature interactions [3]. Finally, the transformed graph is converted back to UML. In MATA, only three stereotypes are defined to perform compositions. However, like in the SmartAdapters approach, it seems difficult to realize a simple merge composition without making the MATA model complex.

3. GEKO, OUR GENERIC COMPOSER

In this section, we introduce GeKo, our generic aspect-oriented composer. We propose to extend the Arcade Game Maker product line with a new game. Then, we modify the basic behavior of the game with an aspect to make the game more realistic.

3.1 Extending the Arcade Game Maker Product Line

In the Arcade Game Maker (AGM) product line, all the games are based on the following principle: some elements are mobile and should extend *MoveableSprite* while the other elements are stationary and should extend *StationarySprite*. When a *tick* occurs, all the movable sprites move. Then, collisions are detected and resolved. If the stationary sprite is absorbing then the movable sprite is deleted. Otherwise it is reversed. The base AGM product line is composed of three games: pong, bowling and brickles.

We propose to extend this product line with a simple strategy game called "Command and Destroy". The basic idea of this game is to command a small set of tanks and destroy the enemy base. The tanks are movable sprites that move in an area containing some obstacles (stationary sprites). For each tank, the player chooses its initial direction and the

moment when the tank starts, after rapidly analyzing the area (dangerous places, bonuses, etc). The enemy can fire at the tanks to destroy them. The player wins if at least two tanks reach the enemy base.

Basically, this game is not so different from the other games of the AGM product line. It simply consists in moving some movable sprites from one place to another. But, the basic collision management proposed by the other games is not well adapted to our game. A tank being absorbed by an obstacle does not make sense. A tank that reverses its direction (*i.e.*, the opposite direction of the enemy base) when it collides with an obstacle, or remains blocked against this obstacle may rapidly become boring for the player. In the next sub-section, we propose to modify the behavior of tanks by weaving an path-finding aspect to make the game-play more realistic and funny.

3.2 Making tanks intelligent with an aspect

In the base AGM product line, a stationary object can either be absorbing or non absorbing (bouncy). We assume that a boolean attribute (*isAbsorbing*) allows programmers to make the distinction between these two kinds of object. We now want to consider another type of stationary objects that will simply block movable objects. The first part of the aspect aims at changing the structure of the base product line. It changes the type of the attribute discriminating the stationary object. This attribute is now a value from an enumeration that contains the three different types.

The most interesting part of the aspect aims at changing the behavior of movable objects. It is illustrated in Figure 1. The advice propose to circle around a blocking object, by moving around the bounding box of the blocking object until it is possible to move in the direction defined in the movable object. This advice will be applied to the tanks in order to refine the basic behavior of movable objects in order to circle around obstacles. Note that it is possible to define other advices to provide other behaviors. For example we can imagine that blocking stationary should act as standard "bouncy" element for most of the movable elements (*e.g.*, grenades and other projectiles), except for tanks. In the resulting model, the conditions of the transitions have changed to consider the three possible types and the state to move around an obstacle is introduced. This example is simple, so it would also be possible to directly modify (without an aspect) the base behavior in order to integrate the path-finding. But, integrating this refinement into an aspect allows us to reprent clearly the basic behavior and the modifications needed to integrate the new behavior. It provides a better traceability from one refinement to another.

The GeKo approach is based on the definition of mappings between the pointcut and the base model, and the pointcut and the advice. These mappings are defined over the concrete syntax of models by linking model elements. These links are totally generic and do not use any domain-specific knowledge, so that we can define mappings for any domain metamodel. We simply check that the bound elements are compatible.

Our example was rather specific to the AGM product line, thus we simply describe the pointcut as a subset of the base model. However, it is possible to describe abstract pointcut as explained in the previous subsection. The mappings between the pointcut and the advice are also straightforward. But it is possible to define more complex bindings *e.g.*, two

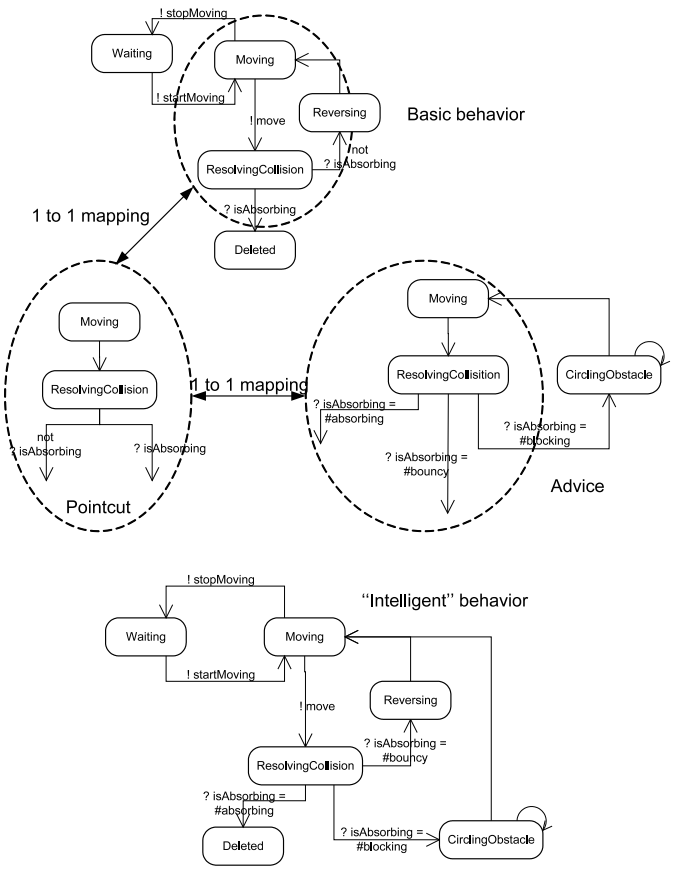

Figure 1: Weaving an path-finding aspect

states from the pointcut can be mapped on the same state of the advice. This will be formalized precisely and exemplified in Section 4.

3.3 Extending GeKo with existing tools

GeKo is a prototype implemented in Kermeta [10]. It can easily be extended with extending approaches also implemented in Kermeta. Kermeta is an extension of EMOF (Essential Meta-Object Facilities) that allows designers to define the static (OCL constraints) and the operational (behavior) semantics of their metamodel. It is available[1] under the open-source EPL license and compatible with the Eclipse Modeling Framework (EMF). For example, we have connected the pattern matching framework of Ramos *et al.* [11] to automatically identify join points in a base model that match the pointcut. The example presented above is rather specific to one particular context.

Most of the time domain-specific metamodels are too restrictive in order to design pointcuts. Indeed, they define lots of implicit and explicit constraints that models should respect. For example, in a state machine, all the transitions should target exactly one state and declare exactly one source state. But, if we want to match all the transitions that target a given state, representing the source state and the containing state machine is not useful.

In order to be able to describe more easily pointcuts, we construct on demand a more flexible metamodel [11] **MM'**,

[1]http://www.kermeta.org

using a model transformation written in Kermeta [10]. **MM'** is equivalent to **MM**, except that:

1. No invariant or pre-condition is defined in **MM'**;
2. All features of all meta-classes in **MM'** are optional;
3. **MM'** has no abstract element.

This model transformation is generic because instead of manipulating the domain elements (Vertex, Transition, ...), it manipulates higher-level concepts provided by ECore, MOF or EMOF for describing metamodels. Consequently, **MM'** can be generated for any input metamodel **MM**. Additionnaly, we can define any element from **MM'** as a role in order to design abstract pointcut. Then, these roles are substituted by join points from a base model. These abstract pointcut allow us to define reusable aspects such as security, authentication that can be reused into different base models.

In order to identify join points in a base model that match the pointcut, we use a Prolog-based pattern matching engine [11], implemented in Kermeta [10]. The domain metamodel and the base model is automatically mapped onto a Prolog knowledge base. Then, an abstract pointcut with roles is transformed into a Prolog query executed over this knowledge base. Finally, the Prolog results are converted back into a Kermeta data-structure.

Another possible extension is to use the adaptations provided by the generic version of SmartAdapters [8, 9] to realize fine-grained composition in order to finalize the composition in the case of our mappings are not expressive enough. Additionally, it would be interesting to integrate the variability mechanisms proposed in SmartAdapters [5, 8] in our GeKo tool to make our aspects more flexible. For example, the two possible advices discussed in the previous subsection may be handled in a single advice with variability.

4. COMPOSITION FORMALIZATION

This section provides a formalization of our generic aspect-oriented composer implemented in Kermeta.

4.1 EMOF: Essential MetaObject Facilities

Essential Meta-Object Facilities (EMOF) 2.0 is a minimal metamodeling language designed to specify metamodels. It is a subset of the OMG standard MOF[2]. It provides the set of elements required to model object-oriented systems. The minimal set of EMOF constructs required for the composition algorithm is presented in Figure 2.

All objects have a class which describes their properties and operations. An Object extends an Element. The *getMetaClass()* operation returns the Class that describes this object. The *container()* operation returns the containing parent object. It returns null if there is no parent object. The *equals(element)* determines if the element (an instance of Element class) is equal to this Element instance. The *set(property, element)* operation sets the value of the property to the element. The *get(property)* operation returns a List or a single value depending on the multiplicity.

The isComposite attribute under class Property returns true if the object is contained by the parent object (call container). Cyclic containment is not possible, i.e. an object can be contained by only one other object. To remove an object from a model, this object is removed from its container.

[2]http://www.omg.org/mof/

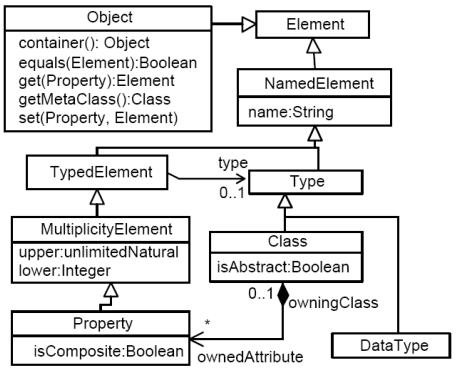

Figure 2: Fragment of EMOF classes required for composition

The *getAllProperties()* operation (not shown in the figure) of the Class returns all the properties of instances of this Class along with the inherited properties. The attributes, upper and lower, of class MultiplicityElement, represent the multiplicities of the associations at the metamodel level. For example, "0..1" represents a lower bound "0" and an upper bound "1". If the upper bound is less than or equal to "1" then the property value is null or a single object; otherwise it is a collection of objects.

A model instance of a metamodel written with EMOF contains an elements reference with a set of EMOF object instances.

4.2 Formalization

The main idea of our generic composition of two models *base* and *advice* is the use of a third model called *pointcut* and two morphisms allowing the identification of the objects of *base* which have to be kept, to be removed and to be replaced with those of *advice*.

Let *base*, *pointcut* and *advice* be three models (defined by a set of objects). Let f and g be two morphisms such as (1) f is a bijective morphism from *pointcut* to a subset $jp \subseteq base$ (*jp* being a join point) and (2) g is an injective morphism from *pointcut* to *advice*. (Let us note that f is automatically obtained from the detection step).

The two morphisms partition the models *base* and *advice* in five sets:

- the set $\mathcal{R}_{keep}$ representing the set of objects of *base* which have to be kept. An object *obj* of *base* is in $\mathcal{R}_{keep}$ if there is no object obj' in *pointcut* such as $f(obj') = obj$. More formally, $\mathcal{R}_{keep} = \{obj \in base | \nexists obj' \in pointcut, f(obj') = obj\}$.
- the set $\mathcal{R}_{-}$ representing the set of objects of *base* which have to be removed. An object *obj* of *base* is in $\mathcal{R}_{-}$ if there exists $obj' \in pointcut$ such as f maps obj' on *obj* and if there is no $obj'' \in advice$ such as g maps obj' on obj''. More formally, $\mathcal{R}_{-} = \{obj \in base | \exists obj' \in pointcut, \nexists obj'' \in advice, f(obj') = obj \wedge g(obj') = obj''\}$.
- the set $\mathcal{R}_{\pm}$ representing the set of objects of *base* which have to be replaced with elements of *advice*. An element *obj* of *base* is in $\mathcal{R}_{\pm}$ if there exists $obj' \in pointcut$ and $obj'' \in advice$ such as f maps obj' on *obj* and

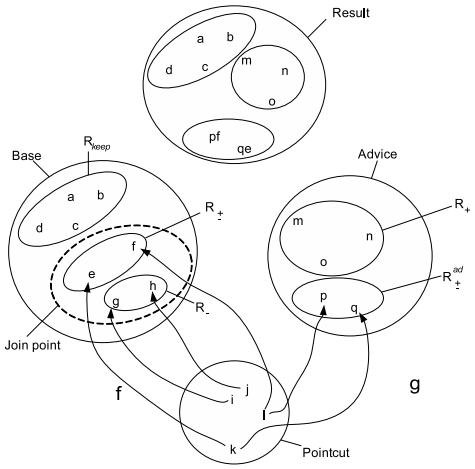

Figure 3: Illustration of the sets identification

g maps obj' on obj''. More formally, $\mathcal{R}_{\pm} = \{obj \in base | \exists obj' \in pointcut, \exists obj'' \in advice,\ f(obj') = obj \wedge g(obj') = obj''\}$.

- In the same way, we define the set $\mathcal{R}^{ad}_{\pm}$ representing the objects of *advice* which replace the objects of $\mathcal{R}_{\pm}$. An object $obj'' \in \mathcal{R}^{ad}_{\pm}$ replaces an object $obj \in \mathcal{R}_{\pm}$ if and only if there exists an object *obj'* in *pointcut* such as $f(obj') = obj$ and $g(obj') = obj''$. Formally, $\mathcal{R}^{ad}_{\pm} = \{obj \in advice | \exists obj' \in pointcut, \exists obj'' \in base, g(obj') = obj \wedge f(obj') = obj''\}$.

- the set $\mathcal{R}_{+}$ representing the set of objects of *advice* which have to be added in *base*. An object *obj* of *advice* is in $\mathcal{R}_{+}$ if there is no $obj' \in pointcut$ such as g maps obj' on obj. More formally, $\mathcal{R}_{+} = \{obj \in advice | \nexists obj' \in pointcut, g(obj') = obj\}$

An example of partition can be found in Figure 3.
We can now define the composition of two models

Definition 1 (Generic Composition). *Let* base, pointcut *and* advice *be three models. Let* f *and* g *be two morphisms as defined previously. The composition of* base *with* advice *is two-phased: 5*

- *First,* $result = \mathcal{R}_{keep} \cup \mathcal{R}_{+} \cup \mathcal{R}^{ad}_{\pm} \setminus \mathcal{R}_{-} \setminus \mathcal{R}_{\pm}$,
- *Second, the properties of the objects of* result *are cleaned.*

More specifically, in the first phase we keep the objects of $\mathcal{R}_{keep}$ and we remove the objects of $\mathcal{R}_{-}$ and $\mathcal{R}_{\pm}$. The objects of $\mathcal{R}^{ad}_{\pm}$ replace those of $\mathcal{R}_{\pm}$, i.e., if we note $obj'' \in \mathcal{R}^{ad}_{\pm}$ the object which replaces the object $obj \in \mathcal{R}_{\pm}$, the properties of obj'' are modified according to those of obj. Formally, let p be a property of obj'', if $p.upper > 1$ then p is complemented by the corresponding property of obj, *i.e., $obj''.get(p) = obj''get(p) \cup obj.get(p)$. Moreover, a property p of an object obj' which targeted[3] obj are updated. Now, if $p.upper > 1$ then $obj'.get(p) = obj'.get(p) \cup obj'' \setminus obj$, else $obj'.get(p) = obj''$.*

[3]A property p of a object obj targets an object obj'' if $obj'' \in obj.get(p)$ and if p is not a composite property

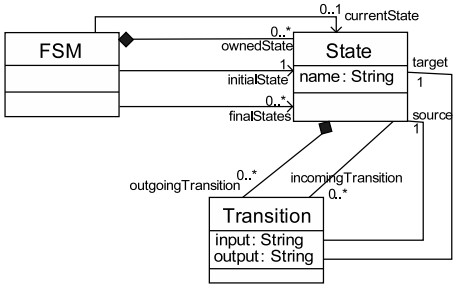

Figure 4: Metamodel of FSM

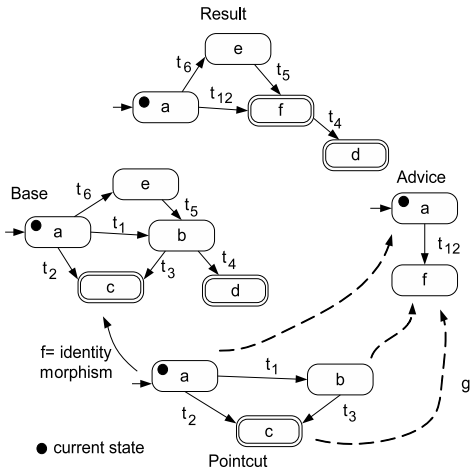

Figure 5: Example of FSM composition

The second step consists in the deletion of the "references" to objects removed in the first phase (objects of $\mathcal{R}_{-}$). Let us consider an object obj removed, an object $obj' \in result$ and a property p such as $obj \in obj'(p)$. Then, if $p.upper > 1$, we remove obj from the list $obj'(p)$. If the cardinality of p is 0..1*, we remove obj from $obj'(p)$. Finally, if the cardinality of p is* 1..1*, we remove obj' from* result *to avoid the obtaining of non-consistent model. We recursively apply the clean operation on* result *while there exist objects which have to be removed from* result.

Let us illustrate this definition of generic composition by the simple example in Figure 5 where we compose the finite state machine (FSM) *base* and *advice*. The *base* FSM contains the objects: $\{FSM : base,\ ^4State : a^b, State : b, State : c, State : d, State : e, Transition : t_1, Transition : t_2, Transition : t_3, Transition : t_4, Transition : t_5, Transition : t_6\}$. The *advice* FSM contains the objects:$\{FSM : advice, State : a^{ad}, State : f,\ Transition : t_{12}\}$. The morphism f is the identity morphism. The morphism g associates respectively *State a*, *State b*, and *State c* of *pointcut* to *State* a^{ad}, *State f*, and *State f* of *advice*.

The morphisms allow the identification of the following sets:

- $\mathcal{R}_{keep} = \{FSM : base, State : d, State : e, Transition : t_4, Transition : t_5, Transition : t_6\}$.

[4]we add $.^{ad}$ and $.^{b}$ to distinguish the object $State : a^{ad}$ coming from the *advice* and the object $State : a^{b}$ coming from the *base*.

- $\mathcal{R}_- = \{Transition : t_1, Transition : t_2, Transition : t_3\}$.
- $\mathcal{R}_\pm = \{State : a^b, State : b, State : c\}$.
- $\mathcal{R}^{ad}_\pm = \{State : a^{ad}, State : f\}$.
- $\mathcal{R}_+ = \{Transition : t_{12}\}$

Consequently, the result of the composition of *base* and *advice* is equal to $result = \{FSM : base, State : d, State : e, Transition : t_4, Transition : t_5, Transition : t_6, State : a^{ad}, State : f, Transition : t_{12}\}$ where the properties of $State : a^{ad}$ and $State : f$ have been updated but also the properties of objects which target $State : a^{ad}$ and $State : f$. According the FSM metamodel, the class *State* is characterized by three properties: $outgoingTransition[0..*]$, $incomingTransition[0..*]$ and $name[1..1]$. For the properties with a cardinality higher than 1, we have:

$\bullet State : a.outgoingTransition =$
$State : a^{ad}.get(outgoingTransition) \cup$
$^5State : a^b.get(outgoingTransition)$.
$State : a.outgoingTransition = \{Transition : t_{12}\} \cup \{Transition : t_6\}$.

$\bullet State : a.incomingTransition =$
$State : a^{ad}.get(incomingTransition) \cup$
$State : a^b.get(incomingTransition)$.
$State : a.incomingTransition = \emptyset \cup \emptyset$.

$\bullet State : f.outgoingTransition =$
$State : b.get(outgoingTransition) \cup$
$State : c.get(outgoingTransition) \cup$
$State : f.get(outgoingTransition)$.
$State : f.outgoingTransition = \{Transition : t_4\} \cup \{\emptyset \cup \emptyset$.

$\bullet State : f.incomingTransition =$
$State : b.get(incomingTransition) \cup$
$State : c.get(incomingTransition) \cup$
$State : f.get(incomingTransition)$.
$State : f.incomingTransition = \{Transition : t_5\} \cup \emptyset \cup \{Transition : t_{12}\}$.

Furthermore, let us consider the properties of the objects which targeted the objects which have been replaced, i.e., $State : a$, $State : b$ and $State : c$:

objects Replaced	targeted by the properties:
$State : a^b$	$FSM : base.\{initialState, currentState\}$ $Transition : t_6.source$
$State : b$	$Transition : t_5.target$
$State : c$	$FSM : base.finalState$

After the composition, these properties target $State : a^{ad}$ instead of $State : a^b$, and $State : f$ instead of $State : b$ and $State : c$ (for instance, now $State : f$ is a $finalState$ instead of $State : c$).

Finally, the *clean* operation removes no additional object from properties because the three removed objects $Transition : t_1$, $Transition : t_2$, and $Transition : t_3$ have been only targeted by properties of objects already removed ($State : a$, $State : b$, and $State : c$).

5. CONCLUSION

In this paper, we have presented our generic aspect-oriented composer. This AOM approach called GeKo (generic composition with Kermeta) has a well-defined semantic and can easily be connected with other existing tools like a pattern matching engine to automatically identify join points. In the context of software product lines, our approach can facilitate the refinement of models to easily derive products. We have shown how to modify the Arcade Game Maker product line in order to integrate a new game with a refined behavior. The formalization of GeKo, as well as its implementation in Kermeta, is based on the definition of five sets. We are planning to investigate how to use these sets to propose powerful traceability mechanisms in order to visualize the impact of an aspect on a base model. It will be possible to clearly identify what has been removed, added or replaced by the aspect. We are also interested to collaborate with other early-aspect workshop participants to apply our generic AOM approach to DSML used to capture some parts of the requirements or used to model some views of the architecture. The idea can be to quickly prototype formalisms and tools to specify new products with aspects for these modeling languages.

[5]The transition t_1 and t_2 have been removed.

6. REFERENCES

[1] F. Fleurey, B. Baudry, R. France, and S. Ghosh. A Generic Approach For Automatic Model Composition. In *AOM@MoDELS'07: 11th International Workshop on Aspect-Oriented Modeling*, Nashville TN USA, Oct 2007.

[2] R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh. Providing Support for Model Composition in Metamodels. In *EDOC'07: Proceedings of the 11th IEEE International Enterprise Computing Conference*, 2007.

[3] P. K. Jayaraman, J. Whittle, A. M. Elkhodary, and H. Gomaa. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis. In *Proceedings of MoDELS'07*, LNCS, page 15, Nashville TN USA, Oct. 2007. Vanderbilt University, Springer-Verlag.

[4] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *ECOOP'97: Proceedings of the 11th European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Berlin, Heidelberg, and New York, 1997. Springer-Verlag.

[5] P. Lahire, B. Morin, G. Vanwormhoudt, A. Gaignard, O. Barais, and J. M. Jézéquel. Introducing Variability into Aspect-Oriented Modeling Approaches. In *Proceedings of MoDELS'07*, LNCS, pages 498–513, Nashville TN USA, Oct. 2007. Vanderbilt University, Springer-Verlag.

[6] P. Lahire and L. Quintian. New Perspective To Improve Reusability in Object-Oriented Languages. *Journal Of Object Technology (JOT)*, 5(1):117–138, 2006.

[7] C. V. Lopes. *D: A Language Framework for Distributed Programming*. PhD thesis, College of Computer Science, Northeastern University, 1997.

[8] B. Morin, O. Barais, and J. M. Jézéquel. Weaving Aspect Configurations for Managing System Variability. In *VaMoS'08: 2nd International Workshop on Variability Modelling of Software-intensive Systems*, Essen, Germany, January 2008.

[9] B. Morin, O. Barais, J. M. Jézéquel, and R. Ramos. Towards a Generic Aspect-Oriented Modeling Framework. In *3rd International ECOOP'07 Workshop on Models and Aspects*, Berlin, Germany, August 2007.

[10] P. Muller, F. Fleurey, and J. M. Jézéquel. Weaving Executability into Object-Oriented Meta-languages. In *Proceedings of MoDELS'05*, volume 3713 of *Lecture Notes in Computer Science*, pages 264–278, Montego Bay, Jamaica, Oct 2005. Springer.

[11] R. Ramos, O. Barais, and J. M. Jézéquel. Matching Model Snippets. In *Proceedings of MoDELS'07*, LNCS, page 15, Nashville TN USA, Oct. 2007. Vanderbilt University, Springer-Verlag.

[12] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg. Directives for Composing Aspect-Oriented Design Class Models. *Transactions on Aspect-Oriented Software Development I*, LNCS 3880:75–105, 2006.

[13] G. Straw, G. Georg, E. Song, S. Ghosh, R. B. France, and J. M. Bieman. Model Composition Directives. In T. Baar, A. Strohmeier, A. Moreira, and S. Mellor, editors, *Proceedings of UML'04*, volume 3273 of *LNCS*, pages 84–97. Springer, Oct 2004.

[14] J. Whittle and P. Jayaraman. 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.

[15] Software Engineering Institute - Carnegie Mellon Arcade Game Maker Pedagogical Product Line. http://www.sei.cmu.edu/productlines/ppl/