On Some Properties of Parameterized Model Application

Alexis Muller[1], Olivier Caron[1], Bernard Carré[1], and Gilles Vanwormhoudt[1,2]

[1] Laboratoire d'Informatique Fondamentale de Lille,
UMR CNRS 8022 - INRIA Jacquard Project,
Université des Sciences et Technologies de Lille,
59655 Villeneuve d'Ascq cedex, France
[2] GET/ENIC Telecom Lille I
{mullera, carono, carre, vanwormh}@lifl.fr

Abstract. Designing Information Systems (IS) is a complex task that involves numerous aspects, being functional or not. A way to achieve this is to consider models as generic pieces of design in order to build a complete IS. Model composition provides a way to combine models and model parameterization allows the reuse of models in multiple contexts. In this paper, we focus on the use of parameterized models in model driven engineering processes. We outline the needs to compose parameterized models and apply them to a system according to alternative and coherent ordering rules. Such building processes raise open issues: Is the result influenced by the order of applications ? Can we compose independent parameterized models ? Is it possible to define composition chains and find equivalent ones that express the same resulting model ? These requirements are formalized through an apply operator. This operator guarantees properties which can help in the formulation of model driven system construction methodologies. Finally, we briefly describe a modelling tool that supports processes based on this operator.

1 Introduction

Illustrated by new approaches of software development [13,1,12], models are gaining more and more importance in the software development lifecycle. There is a growing need to use them as concrete artefacts [11] through operations like projections, translations or constructions. Projection techniques mainly aim to transpose a model from a technological space (UML for example) to another one (like EJB or CORBA). Translation techniques allow to express the same model in another language (UML to XMI for example). Finally, construction techniques ambition is to produce new models from existing ones.

Among these construction techniques, composition techniques, which permit the building a model from a set of smaller ones, are widely used. Indeed, in spite of new development approaches, systems and thus their models become more complex and bigger. It is thus necessary to deal with this complexity by providing the decomposition of such systems and their models.

A. Hartman and D. Kreische (Eds.): ECMDA-FA 2005, LNCS 3748, pp. 130–144, 2005.

Building new systems by composing prefabricated and validated models promises a quicker design of more reliable systems. It is possible to identify in any information system some concerns that should be applied to other systems [8,10,14]. Nevertheless, their models are made according to a particular system so that they are hard to reuse in order to build new systems. The design of reusable models calls for the usage of some sort of parameterized models or model templates [10,9,18,19]. As far as standardization is concerned, UML2 defines the notion of template [2] which allows representing such generic models as packages parameterized by model elements. In [5], we have formalized the UML2 template binding relationship and introduced OCL rules for checking the correct matching between the required model (specified in the template signature) and a model constructed using this template. This relationship is independent from any construction process.

In this paper, we focus on how to define model driven engineering construction processes with such parameterized models. This needs to express complex compositions of parameterized models which must be applied in a coherent way. Such building processes raise open issues: Is the result influenced by the application order? Can we compose independent parameterized models? Is it possible to define composition chains and find equivalent ones that express the same resulting model?

To support such processes, we introduce an operator (*apply*) to express the application of a template to an existing model. This operator allows to specify how to obtain a model from an existing one by the application and composition of generic ones. It is interesting to note that generic models are models so that template applications can be combined to design richer ones. The next section shows the needs for such an operator and specify its expected properties. Then the third section presents a formalization of its semantics and proves these properties. Section 4 discusses related works about composition and parameterization of models. Finally, we briefly describe a modelling tool that supports processes based on this operator and provides strategies to transform composition of parameterized models into platform specific models before concluding.

2 Applying Parameterized Models

It is possible to use UML 2 template packages to represent parameterized models. For example, let us consider a set of parameterized models designed for resource management systems (inspired from [8] and [21]) where each model provides a useful function such as searching, stock management and resource allocation.

Figure 1 shows a model offering resource management functionalities related to a stock (add, delete, and transfer operations). This model is specified by a template package owning a class diagram providing these functionalities. Elements required for its usage are exposed in the template signature. In our example, the required elements are the classes *Stock* and *Resource*, some of their corresponding properties *identifier* and *ref*, and the association *in*. This element set forms a unique parameter corresponding to the model structure required by the tem-

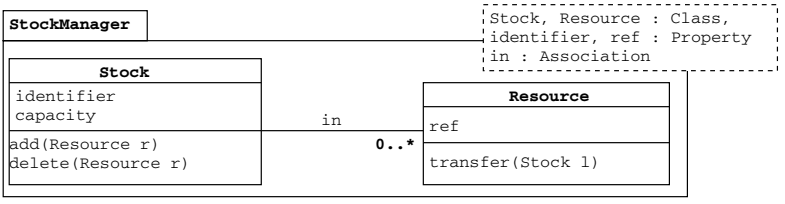

Fig. 1. Resource Management Template

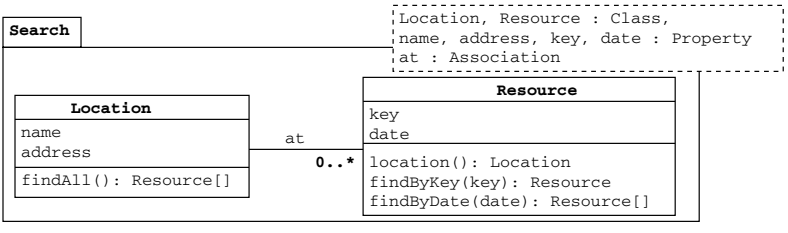

Fig. 2. Search Template

plate to be applied. The other elements correspond to the specific elements of the functionality defined by the template. Those specific elements will be added to which the template will be applied.

Figures 2, 3, and 4 respectively illustrate generic models for searching, resource allocation, and counting. These template examples show that elements of a required model can be either properties, operations, associations or classes.

To illustrate the use of these templates, let us take the example of a car hiring system. Figure 5 shows the primary model of this system. This base describes the structure of the different domain classes used by the system (here *Car*, *Agency*, and *Client*).

The desired system must be able to search a specific car or a specific client, and also to manage the different car allocations. To achieve this, we will use the generic models described before.

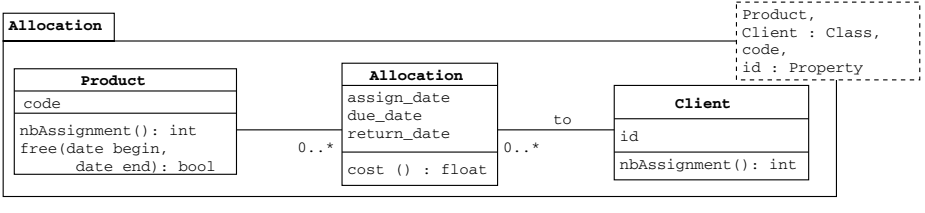

Fig. 3. Resource Allocation Template

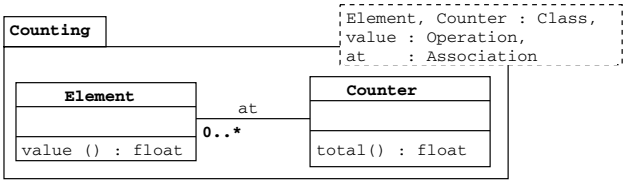

Fig. 4. Counting Template

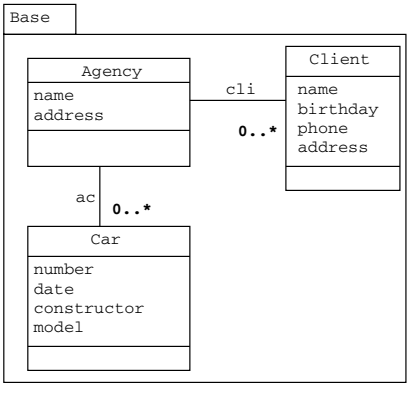

Fig. 5. The Base System

Thus, we need to specify an assembly of these models. For that, we introduce a parameterized model application operator called *apply*. Figure 6 shows how the *Stock Manager* template is applied to the base system. The *apply* operator is specified via an UML stereotyped dependency <<*apply*>> between the template source model and the system target model which establishes correspondence relationships between their respective model elements. This dependency includes the substitution of formal parameters (source model elements) by effective parameters (target model elements). The effective parameters must form a model that matches the required model of the template.

A formulation of a resulting system where corresponding elements are linked by <<trace>> dependencies[1] is shown in Figure 7. Our formulation does not impose any mapping for these <<trace>> dependencies. It is possible, for example, to merge linked elements or to use split representation mechanisms. We already have studied some of these targeting strategies [16,6].

These sketches show how to obtain an extended model from an existing one by the application of some parameterized models. To gain more reuse, composition of parameterized models should be supported too, in order to design complex generic models from simpler ones. This facility is illustrated in Figure 8(a) where

[1] The <<trace>> dependency is a standard UML2 relationship. It is used to link elements that represent the same concept.

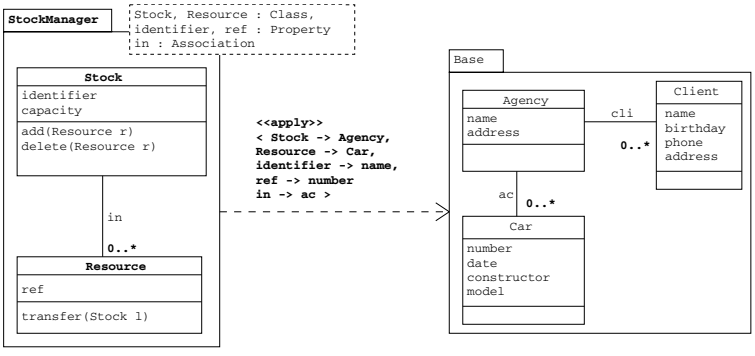

Fig. 6. Applying Stock Manager to Car Hiring System

the *Search* template applied to the *StockManager* template allows to build a new generic model (see Figure 8(b)). The set of parameters of this new model is determined by the union of the target model parameters (from *StockManager*) and the source model unsubstituted parameters (*address* and *date* from *Search*). Figure 8 also emphasizes the ability to apply parameter elements of the source model to parameter elements of the target model. In this case, the first ones (for example *Location*) are substituted by the second ones (*Stock*) in the resulting template.

The *apply* operator must support different construction processes. For the construction of complex systems from a set of parameterized models, it should be possible to elaborate sequences of applications and guarantees consistency properties of the resulting system. We want to exhibit how far it is possible to build the same system model using alternative composition sequences of parameterized models.

For example, Figure 9 shows the design of a complex car hiring system by composing many generic models to the base model shown in Figure 5. This example will serve to illustrate some needs for such sequences. A first need

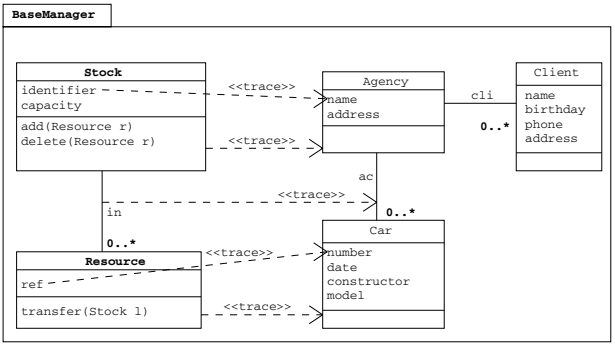

Fig. 7. Base System with Car Management Functionality

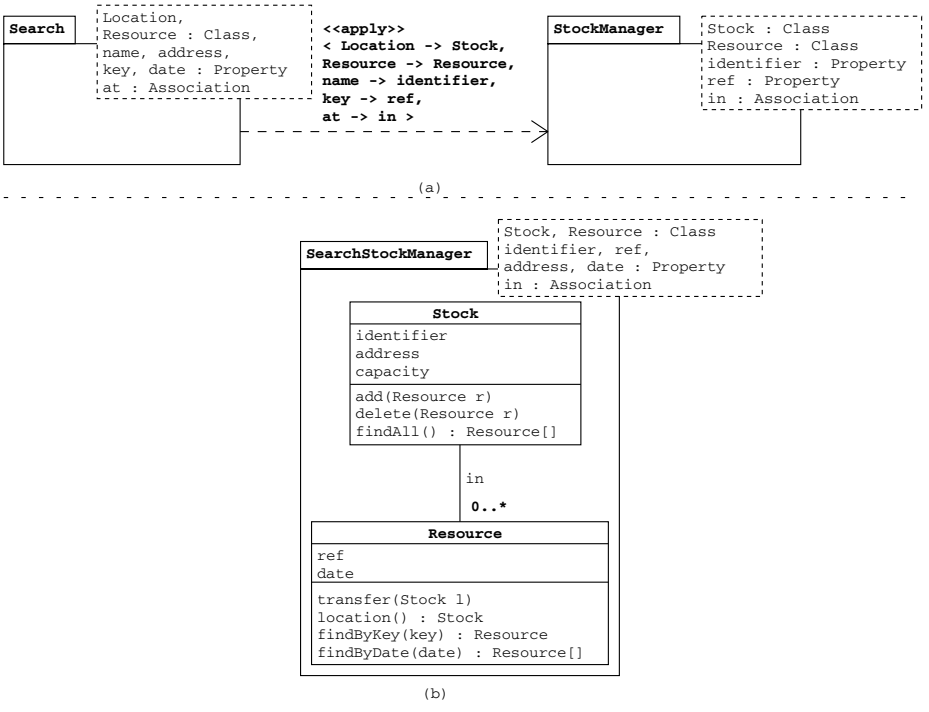

Fig. 8. Template to Template Application

is the ability to apply multiple parameterized model to the same base. When such applications are independent, their evaluation order must not influence the result. It is the case of *Search* and *Allocation* applied to the base.

Another requirement is to express chains of application. Such a chain can be used to apply complex parameterized model resulting from simpler ones. This is illustrated by the application of the *Search* template to the *StockManager* template, explained previously (Figure 8) which produces the *SearchStockManager* model. This new model is then used to add stock management and search functionalities on cars to our system. Note that an alternative construction chain would be to apply the *StockManager* template to the base first, and then the *Search* template to the resulting model. Both chains would produce the same result. We will show this property in the following section. Another example is the application chain *Counting* to *Allocation* to *Base*. Alternative evaluation order of this chain, first *Counting* to *Allocation* then the resulting parameterized model to base or first *Allocation* to *Base* then *Counting* to the enriched *Base*, produce exactly the same result.

In order to provide consistency rules on such building processes, we need a better formalization of parameterized model application. It is the goal of the following section.

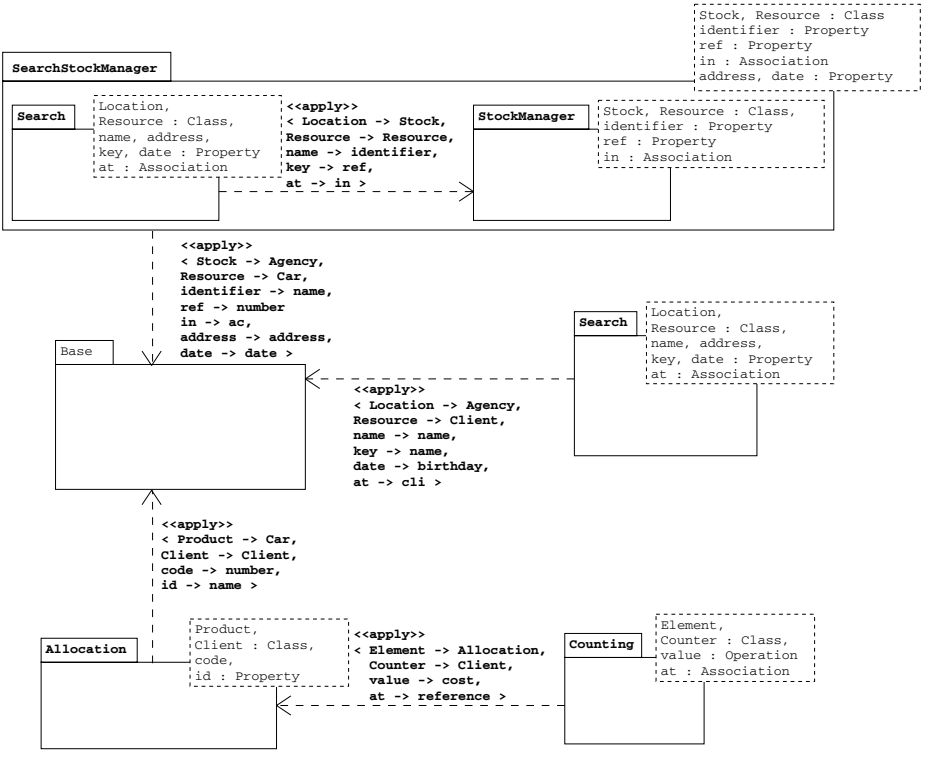

Fig. 9. Car Hiring System

3 Formalization of Parameterized Model Application

The *apply* operator allows to compute a model from the application of a parameterized source one to a target one.

The formalization precises the computation of the resulting model as a set of elements and a set of correspondence relationships between the source model and the target model. These correspondence relationships link elements that represent the same entity[2]. After some definitions, we will state properties of this operator which guarantee the correctness of the previous practices.

3.1 Definitions

Models are considered as sets of model elements that can be classes, attributes, operations or associations. We assume $\mathscr{E}$ to be the set of all these model elements. In case of parameterized models, a model holds a set of parameter elements. As mentioned above, *apply* will construct models that establish correspondence rela-

[2] As the UML2 <<trace>> dependency does.

tionships which are pairs of model elements. The following definition generalizes all these kinds of models.

Definition 1. *A model A is defined by a 3-tuple (E_A, P_A, V_A). E_A is a set of model elements. $P_A \subset E_A$ is a set of parameters. V_A is a set of correspondence relationships defined on $(E_A \times E_A)$.*

Note that in case of models which are not parameterized P_A is empty, and in case of base models V_A is empty.

Definition 2. *Two models are equal if and only if they contain the same set of elements, they own the same set of parameters and they have the same set of correspondence relationships.*

Let two models Z and R, $Z = R \Leftrightarrow \begin{cases} V_Z = V_R \\ E_Z = E_R \\ P_Z = P_R \end{cases}$

Based on these definitions we specify the *apply* operator as follows.

Definition 3. *We write $R = B \underset{s}{\rightarrow} A$ the application of a parameterized model B to a model A according to a substitution set s.*

We note FP_s the set of formal parameters and EP_s the set of effective parameters of s. The resulting model R is constructed according to the following definition rules: $R = B \underset{s}{\rightarrow} A \Rightarrow R = \begin{cases} V_R = V_B \cup s \cup V_A \\ E_R = E_B \cup E_A \\ P_R = (P_B \setminus FP_s) \cup P_A \end{cases}$

Source and target models of a parameterized application cannot share elements: $E_A \cap E_B = \varnothing$.
Formal parameters are elements of the source model: $FP_s \subseteq P_B$.
Effective parameters are elements of the target model: $EP_s \subseteq E_A$.

Note that according to these definitions, parameterized models can be applied to any kind of models, parameterized (see Figure 8) or not (see Figure 6). In the case of parameterized target models, the resulting model is itself parameterized. Recall that resulting parameters are those of the target model plus unsubstituted source model ones. The computation formula of the resulting parameter set (P_R) formalizes this.

3.2 Properties

Assuming these definitions, it is possible to demonstrate a set of properties that guarantees the correctness of application chains and their alternative ordering capacities.

Property 1. *When applying two models to a third one, the order of both applications does not influence the result.*

Let two substitutions set s, s' such as $EP_s \subseteq E_A$ and $EP_{s'} \subseteq E_A$. Then we have
$B \underset{s}{\rightarrow} (C \underset{s'}{\rightarrow} A) = C \underset{s'}{\rightarrow} (B \underset{s}{\rightarrow} A)$.

Thanks to this property, it is not necessary to express in which order parameterized models must be applied. This is shown Figure 9. Anyhow *Search* and *Allocation* are applied to the base, the resulting model is the same.

Proof. *Let* $Z = B \underset{s}{\rightarrow} (C \underset{s'}{\longrightarrow} A)$,

$$Z = B \underset{s}{\rightarrow} Z' \text{ with } Z' = (C \underset{s'}{\longrightarrow} A) \Rightarrow Z' = \begin{cases} V_{Z'} = V_C \cup s' \cup V_A \\ E_{Z'} = E_C \cup E_A \\ P_{Z'} = (P_C \setminus FP_{s'}) \cup P_A \end{cases}$$

$$\Rightarrow Z = \begin{cases} V_Z = V_B \cup s \cup V_{Z'} \\ E_Z = E_B \cup E_{Z'} \\ P_Z = (P_B \setminus FP_s) \cup P_{Z'} \end{cases}$$

$$\Rightarrow Z = \begin{cases} V_Z = V_B \cup s \cup V_C \cup s' \cup V_A \\ E_Z = E_B \cup E_C \cup E_A \\ P_Z = (P_B \setminus FP_s) \cup (P_C \setminus FP_{s'}) \cup P_A \end{cases}$$

Let $Y = C \underset{s'}{\longrightarrow} (B \underset{s}{\rightarrow} A)$,

$$Y = C \underset{s'}{\longrightarrow} Y' \text{ with } Y' = (B \underset{s}{\rightarrow} A) \Rightarrow Y' = \begin{cases} V_{Y'} = V_B \cup s \cup V_A \\ E_{Y'} = E_B \cup E_A \\ P_{Y'} = (P_B \setminus FP_s) \cup P_A \end{cases}$$

$$\Rightarrow Y = \begin{cases} V_Y = V_C \cup s' \cup V_{Y'} \\ E_Y = E_C \cup E_{Y'} \\ P_Y = (P_C \setminus FP_{s'}) \cup P_{Y'} \end{cases}$$

$$\Rightarrow Y = \begin{cases} V_Y = V_C \cup s' \cup V_B \cup s \cup V_A \\ E_Y = E_C \cup E_B \cup E_A \\ P_Y = (P_C \setminus FP_{s'}) \cup (P_B \setminus FP_s) \cup P_A \end{cases}$$

As a result, we have $Z = Y \Rightarrow B \underset{s}{\rightarrow} (C \underset{s'}{\longrightarrow} A) = C \underset{s'}{\longrightarrow} (B \underset{s}{\rightarrow} A)$

Property 2. *For any sequence* $B \underset{s_1}{\longrightarrow} (C \underset{s'_1}{\longrightarrow} A)$*, there exists a sequence* $(B \underset{s_2}{\longrightarrow} C) \underset{s'_2}{\longrightarrow} A$*, that produces the same result, such as* $s_2 = s_1 \setminus ((E_A \times \mathscr{E}) \cap s_1)$ *and* $s'_2 = s'_1 \cup ((E_A \times \mathscr{E}) \cap s_1)$.

Figure 10 shows the application of the *StockManager* template to the *Base* then the application of *Search* to the result. This is an alternative construction chain of the application of *Search* to *StockManager* then to the *Base*. This latter construction is equivalent to applying the complex template *SearchStockManager* to the *Base*. We can verify that its parameter sets are conformant to this property.

Parameter elements of *Search* applied to elements of the *Base* in Figure 10 (*address* and *date*) are not concerned by the direct application (Figure 8) of *Search* to *StockManager*. As result, they are transfered to the application of *SearchStockManager* to the *Base* in Figure 9.

In order to prove this property we need the following intermediate property on parameter substitutions:

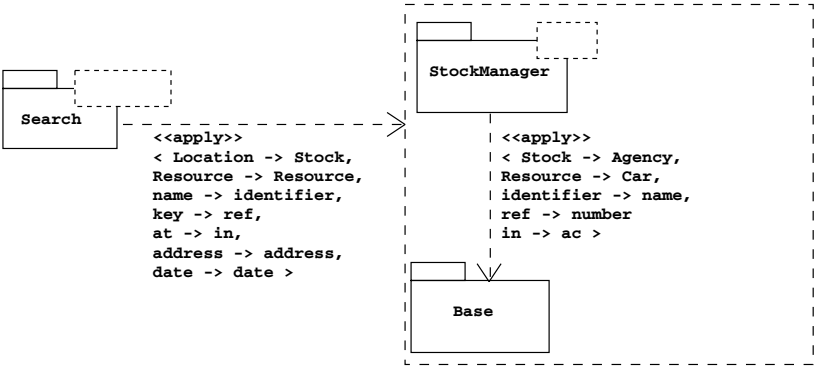

Fig. 10. An alternative to the application of *SearchStockManager*

Property 3. *A parameter can not be substituted more than one time. Once a parameter is substituted, it is no more a parameter in the resulting model.* $(B \underset{s_1}{\longrightarrow} C) \underset{s_2}{\longrightarrow} A \Rightarrow FP_{s_1} \cap FP_{s_2} = \varnothing$.

Proof. *Let* $(B \underset{s_1}{\longrightarrow} C) \underset{s_2}{\longrightarrow} A = R \underset{s_2}{\longrightarrow} A$ *with* $R = B \underset{s_1}{\longrightarrow} C$.
By definition, $P_R = (P_B \setminus FP_{s_1}) \cup P_C$ *or* $FP_{s_1} \subseteq P_B$ *and* $P_B \cap P_C = \varnothing$ *(*$E_B \cap E_A = \varnothing$, $P_B \subset E_B$ *and* $P_A \subset E_A$*) then* $FP_{s_1} \cap P_C = \varnothing$.

$$\begin{aligned} \Rightarrow P_R &= (P_B \setminus FP_{s_1}) \cup (P_C \setminus FP_{s_1}) \\ &= (P_B \cup P_C) \setminus FP_{s_1} \end{aligned}$$

$\Rightarrow P_R \cap FP_{s_1} = \varnothing$ *and by definition* $FP_{s_2} \subseteq P_R$ *so* $FP_{s_1} \cap FP_{s_2} = \varnothing$.

We can now prove the property 2.

Proof. *Let* $R = B \underset{s_1}{\longrightarrow} (C \underset{s'_1}{\longrightarrow} A)$,

$$R = \begin{cases} V_R = V_B \cup s_1 \cup V_C \cup s'_1 \cup V_A \\ E_R = E_B \cup E_C \cup E_A \\ P_R = (P_B \setminus FP_{s_1}) \cup (P_C \setminus FP_{s'_1}) \cup P_A \end{cases}$$

According to property 3:
$P_B \cap P_C = \varnothing$ *and* $FP_{s_1} \cap FP_{s'_1} = \varnothing \Rightarrow P_R = (P_B \cup P_C \cup P_A) \setminus (FP_{s_1} \cup FP_{s'_1})$

Let $R' = (B \underset{s_2}{\longrightarrow} C) \underset{s'_2}{\longrightarrow} A$, $R' = \begin{cases} V_{R'} = V_B \cup s_2 \cup V_C \cup s'_2 \cup V_A \\ E_{R'} = E_B \cup E_C \cup E_A \\ P_{R'} = ((P_B \setminus FP_{s_2}) \cup P_C) \setminus FP_{s'_2} \cup P_A \end{cases}$

According to property 3:
$P_B \cap P_C = \varnothing$ *and* $FP_{s_2} \cap FP_{s'_2} = \varnothing \Rightarrow P_R = (P_B \cup P_C \cup P_A) \setminus (FP_{s_2} \cup FP_{s'_2})$.

Then we get $R = R'$ *if* $s_1 \cup s'_1 = s_2 \cup s'_2$
Since $s_2 = s_1 \setminus ((E_A \times \mathscr{E}) \cap s_1)$ *and* $s'_2 = s'_1 \cup ((E_A \times \mathscr{E}) \cap s_1)$,

$$\begin{aligned} s_2 \cup s'_2 &= s_1 \setminus ((E_A \times \mathscr{E}) \cap s_1) \cup s'_1 \cup ((E_A \times \mathscr{E}) \cap s_1) \\ &= s_1 \cup s'_1 \cup ((E_A \times \mathscr{E}) \cap s_1) \setminus ((E_A \times \mathscr{E}) \cap s_1) \\ &= s_1 \cup s'_1 \end{aligned}$$

A particular case of property 2 stands when, for each application in the chain, all parameters of the source model are substituted with elements of its target model. In this case, any parameterized model can be directly applied to the next model within the application chain. For such application chains, the evaluation order does not influence the result. This is formalized by the next property.

Property 4. *Let* $B \underset{s}{\rightarrow} C \underset{s'}{\longrightarrow} A$ *an application chain such as* $EP_s \subseteq E_C$*, it can be evaluated either as* $B \underset{s}{\rightarrow} (C \underset{s'}{\longrightarrow} A)$*, or as* $(B \underset{s}{\rightarrow} C) \underset{s'}{\longrightarrow} A$.

Proof. *According to property 2:*
let $B \underset{s}{\rightarrow} (C \underset{s'}{\longrightarrow} A) = (B \underset{s_2}{\longrightarrow} C) \underset{s'_2}{\longrightarrow} A$ *with* $\begin{cases} s_2 = s \setminus ((E_A \times \mathscr{E}) \cap s) \\ s'_2 = s' \cup ((E_A \times \mathscr{E}) \cap s) \end{cases}$

Since $EP_s \subseteq E_C$, $EP_s \cap E_A = \varnothing$ *(by definition* $E_C \cap E_A = \varnothing$*). From this, we deduce that* $(E_A \times \mathscr{E}) \cap s = \varnothing$ *(it can not exist* x *such as* $x \subseteq EP_s$ *and* $x \subseteq E_A$*)*
$\Rightarrow \begin{cases} s_2 = s \\ s'_2 = s' \end{cases}$
Which proves the property.

This last property allows, in Figure 9, to apply *Counting* to *Allocation* to *Base*, without specifying any evaluation order.

4 Related Work

Approaches allowing the decomposition of a system following its functional or technical dimensions aim to simplify the design of information systems. However to form the global system all the dimensions must be assembled. Various approaches exist to express this assembly.

Examples of such decomposition techniques are the Subject-Oriented Design (SOD) approach [8] or the Catalysis approach [10]. The SOD approach proposes the design of an independent model for each concern of the system. These models are called *Subjects* and take the form of a standard UML package. A new type of relation (*CompositionRelationship*) is proposed to compose subjects and express the composition of their elements. A criterion of correspondence can be attached with this relation to indicate whether elements of the same name constituting the composite represents the same entity (default) or not.

The Catalysis approach proposes to decompose the design of systems in horizontal and vertical slices. Vertical slices correspond to a functional decomposition of the system from the points of view of various categories of users. Horizontal

slices give a decomposition according to the technical concerns of the system. In this approach, packages are also used to represent the various slices of the system.

These structuring techniques only support the assembly of models designed by analysis of a particular system. They are not suitable for the building of new systems from reusable models [15]. In the following, some approaches have addressed this issue by introducing parameterization techniques.

In [14] we have proposed to consider views as a decomposition technique of systems and their models. A view captures some coherent functional aspect that can be added to a system. Each view is represented as a UML package which contains a model of the functional aspect. The application of such an aspect model to a base model is done by a connection mechanism, which allows to target view model elements to the base ones. The work presented in this paper generalizes this mechanism through a parameterized application process.

The Catalysis approach proposes specific constructions in order to design reusable packages: model frameworks. Those are represented using template packages that are abstract packages containing some elements that must be substituted for being concretized and used. This set of substitutions is defined by a set of element pairs (required element/system element). The Theme approach [9,18] proposes an analysis method of the system (Theme/Doc) which helps in the identification of relations among various functionalities, and a notation (Theme/UML) which allows the formulation of these various functionalities as template packages called *Themes*. A relation (named *bind*) is used to express the parameterized composition of two Themes. In this paper, we focus on parameterized composition chains and their ordering properties. We have focused on structure diagrams. It will be useful to evaluate this formalization on dynamic model elements such as sequence diagrams of Theme or parameterized collaboration diagrams [20].

France *et al.* describe an Aspect Oriented Modelling technique in which aspect and primary models are expressed using UML [17,19]. Aspects are specified by parameterized models. Like SOD, elements of same type and same name are merged to form a single one in the composed model. In order to allow composition of generic aspects (in which elements are named differently as primary model elements) they defined a set of composition directives that can be used to modify the default composition procedure. They also provide directives to state the order of composition between aspects and the primary model. In this paper, we study how far composition orderings are equivalent. These properties are particularly useful in the case of complex composition processes where parameterized models could be composed through alternative composition chains.

Though, our definitions do not impose any targeting strategies, particularly for the structuring of the resulting modelling packages and the realisation of the parameters substitution process. It would be possible, for example, to apply fusion or integration strategies. In this way, the semantic of our apply operator differs from the UML2 merge relationship that defines new elements. It also differs from the MOF combine relationship and the package extension techniques

[7] that impose fusion semantic. Moreover, these relationships are not defined to compose parameterized models.

5 Tool

In the context of MDA approach, we have developed a modelling tool based on the Eclipse Modelling Framework and the UML2 Eclipse plug-in. The Eclipse Modelling Framework is a Java meta-modelling framework that allow to create models in a programmatic way or by a basic (non-graphical) editor. The UML2 Eclipse plug-in is defined by EMF and provides a set of Java classes to handle UML2 models.

Our tool[3] adds a graphical representation and allows defining generic models as UML2 template packages at the PIM level. It provides the functionality of composing these generic models and applying them in order to build a complete system. Strategies to transform composition of parameterized models into platform specific models are offered (see Figure 11).

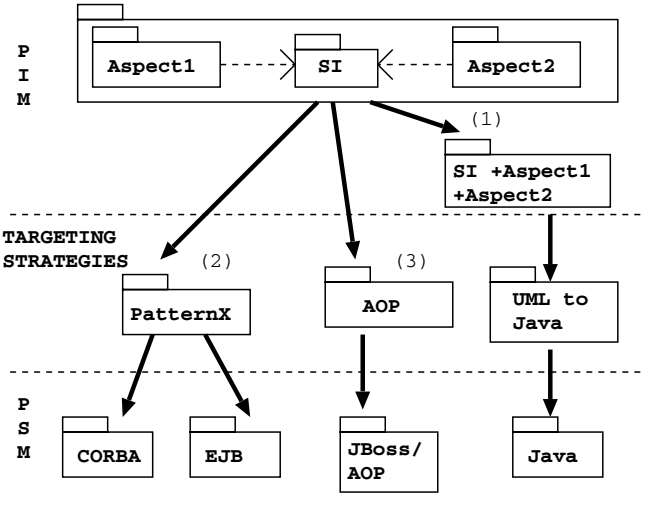

Fig. 11

The simple strategy (1) is to merge all generic models into a single one. Note that this strategy can produce name clashes in the merged model which must be resolved by the user. The resulting model is a standard object-oriented model that can be translated in any object-oriented language (Java in our case). In order to preserve genericity and traceability of our templates down to platform specific models, we have explored two targeting strategies. For the first one (2), we have defined some design patterns allowing to implement generic splited entities and

[3] Available at http://www.lifl.fr/~mullera/cocoamodeler

experiment them on different platforms [16,6,3]. Our tool can generate IDL and Java code for the CORBA platform according to these patterns. We also explored an AOP targeting strategy (3). We have extended the JBoss/AOP framework to support aspect entities and generate an XML descriptor for their weaving. It is possible to select a targeting strategy specific to each application. The MDA process described in [4] based on marked intermediate PIM is being integrated.

6 Conclusion

Several model driven approaches recognize templates, particularly in the UML sphere, as a powerful technique to specify parameterized models and their usage in the construction of whole system models. We have focused here on parameterized model application which allows to obtain an extended model from an existing one. We have justified concretely and formally some properties which guarantee the correctness of application chains and their alternative ordering capacities.

The formalization is deliberately independent from any specific usage or existing methodologies. It would help in supporting model driven processes and tools dedicated to systems construction by the application and composition of prefabricated generic models. It also allows to combine generic models in order to obtain richer ones.

All this work is integrated into a design tool based on the Eclipse Modeling Framework and the Eclipse UML2 plug-in, from modeling to implementation-level languages. It will give the ability to manage libraries (design, composition, import,...) of parameterized models as standard UML2 templates. This will help in building systems by applying parameterized models.

References

1. OMG Model-Driven Architecture Home Page, http://www.omg.org/mda.
2. Auxiliary Constructs Templates, http://www.omg.org/docs/ptc/03-08-02.pdf, pages 541-568. UML 2.0 Superstructure Specification, 2003.
3. O. Barais, A. Muller, and N. Pessemier. Extension de Fractal pour le support des vues au sein d'une architecture logicielle. In *Objets Composants et Modèles dans l'ingénierie des SI (OCM-SI 04)*, Biarritz, France, jun 2004. http://inforsid2004.univ-pau.fr/AtelierOCMv1.htm.
4. X Blanc, O Caron, A Georgin, and A Muller. Transformation de modèles : d'un modèle abstrait aux modèles ccm et ejb. In *Langages, Modèles, Objets (LMO'04)*, Lille, France, Mars 2004. Hermès Sciences.
5. O. Caron, B. Carré, A. Muller, and G. Vanwormhoudt. Formulation of UML 2 Template Binding in OCL. In Thomas Baar, Alfred Strohmeier, Ana Moreira, and Stephen J. Mellor, editors, *UML 2004 - The Unified Modeling Language. Model Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004, Proceedings*, volume 3273 of *LNCS*, pages 27–40. Springer, October 2004.

6. Olivier Caron, Bernard Carré, Alexis Muller, and Gilles Vanwormhoudt. Mise en oeuvre d'aspects fonctionnels réutilisables par adaptation. In *Première journée Francophone sur le Développement de Logiciels par Aspects, JFDLPA 2004*, Paris, France, September 2004.
7. A Clark, A Evans, and S Kent. A Metamodel for Package Extension with Renaming. In H Hussmann J-M Jezequel and S Cook, editors, *The Unified Modeling Language 5th International Conference*, Proceedings LNCS 2460, pages 305–320, Dresden, Germany, September 2002.
8. S. Clarke. Extending standard UML with model composition semantics. In *Science of Computer Programming, Elsevier Science*, volume 44, pages 71–100, 2002.
9. Siobhán Clarke and Robert J. Walker. Generic aspect-oriented design with Theme/UML. In Robert E. Filman, Tzilla Elrad, Siobhán Clarke, and Mehmet Akşit, editors, *Aspect-Oriented Software Development*, pages 425–458. Addison-Wesley, Boston, 2005.
10. Desmond D'Souza and Alan Wills. *Objects, Components and Frameworks With UML: The Catalysis Approach*. Addison-Wesley, 1999.
11. David S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.
12. Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley, 2004.
13. S Kent. Model Driven Engineering. In *Proceedings of IFM 2002*, LNCS 2335, pages 286–298. Springer-Verlag, 2002.
14. A. Muller, O. Caron, B. Carré, and G. Vanwormhoudt. Réutilisation d'aspects fonctionnels : des vues aux composants. In *Langages et Modèles à Objets (LMO'03)*, pages 241–255, Vannes, France, January 2003. Hermès Sciences.
15. Alexis Muller. Reusing Functional Aspects : From Composition to Parameterization. In *Aspect-Oriented Modeling Workshop, AOM 2004*, Lisbon, Portugal, October 2004.
16. Olivier Caron, Bernard Carré, Alexis Muller, and Gilles Vanwormhoudt. A Framework for Supporting Views in Component Oriented Information Systems. In *Proceedings of International Conference on Object Oriented Information Systems (OOIS'03)*, volume 2817 of *LNCS*, pages 164–178. Springer, September 2003.
17. Robert France and Geri Georg and Indrakshi Ray. Supporting Multi-Dimensional Separation of Design Concerns. In *AOSD Workshop on AOM: Aspect-Oriented Modeling with UML*, march 2003.
18. Siobhán Clarke and Robert J. Walker. Composition Patterns: An Approach to Designing Reusable Aspects. In *23rd International Conference on Software Engineering (ICSE)*, May 2001.
19. Greg Straw, Geri Georg, Eunjee Song, Sudipto Ghosh, Robert France, and James M. Bieman. Model composition directives. In Thomas Baar, Alfred Strohmeier, Ana Moreira, and Stephen J. Mellor, editors, *UML 2004 - The Unified Modeling Language. Model Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004, Proceedings*, volume 3273 of *LNCS*, pages 84–97. Springer, 2004.
20. Gerson Sunyé, Alain Le Guennec, and Jean-Marc Jézéquel. Design patterns application in UML. In E. Bertino, editor, *Proceedings of ECOOP 2000*, volume 1850 of *LNCS*, pages 44–62. Springer, 2000.
21. Alan Wills. Frameworks and component-based development. In *Proceedings of International Conference on Object Oriented Information Systems (OOIS'96)*, pages 413–431, 1996.