Structural Variants Detection for Design Pattern Instantiation

Cédric Bouhours, Hervé Leblanc, Christian Percebois

IRIT – équipe MACAO (Modèles, Aspects et Composants pour des Architectures à Objets)
Université Paul Sabatier
118 Route de Narbonne
F-31062 TOULOUSE CEDEX 9
{bouhours,leblanc,perceboi}@irit.fr

ABSTRACT. *In this paper, we suggest to directly take into account the know-how of experts during activities of a process development. Such a work imposes to be able to analyze and to transform models, in particular in order to inject design patterns. Our approach considers models produced by the process as potential alternative models which can be replaced by design patterns. We introduce the definition of remarkable features associated to each alternative model for a design pattern which summarizes its characteristics such as association, features, generalization… This approach has been validated on GOF structural design patterns using an OCL*[*] *backtrack algorithm which automatically identifies classes may be able to play roles defined in the pattern.*

1. Context

Our team works on model-driven process development. This new generation of processes has been proposed by the emergent community MDE which aims at giving a productive character on models. More precisely, we propose to tool some approaches based on UML models thanks to the NEPTUNE project (Nice Environment with a Process and Tools using Norms - UML, XML and XMI - and Example) (NEPTUNE, 1995) (Leblanc and al, 2005). However, model driven development processes should be able to reuse the know-how of experts generally expressed in terms of analysis, design or architectural patterns approved by the expert community.

To help designers, it seems useful to propose a method allowing to determine fragments of models which may be substitutable with a pattern. This method will have to suggest a set of patterns to the designers who could integrate them into their model. The integration will be dealt by automatic transformations. In this first study, we have tried to determine the condition of applicability of each structural pattern proposed by (Gamma and al, 1995). Up to date, in spite of the efforts to improve design pattern classification, and assistance tools for patterns integration in models, we have not found a tool inspecting models, and urging to use patterns, in the most automatic possible way. To do that, we had to find a set of structural characteristics which, once located in a model, would target the fragment of model to be transformed by the injection of a pattern.

According to the taxonomy proposed by (Chikofsky and al, 1990), the implemented technique could be connected to a redocumentation technique so as to permit to do restructuring model. So we are in a reengineering stage dedicated to UML design models and patterns. In a first part, we will present the concept of alternative models and the technique used to identify them in design model. Next, we will present some alternative models dedicated to one design pattern and in a third part, we will summarize the results found with the other GOF patterns.

[*] *Object Constraint Language*

2. Alternative models and structural features

To define these applicability conditions, we started by seeking the alternative models of each pattern. An alternative model is a model which solves the same problem as the pattern, but with a more complex or different structure than the pattern. Therefore, in agreement with hypotheses on design patterns and class design defects (Guéhéneuc and al, 2001), it is a candidate model to substitution with a pattern. After obtaining these alternatives, we have searched for remarkable features allowing to detect them in every model. Our method is connected with the exact pattern matching of alternative models on UML models. We have chosen to detect a set of alternative models to a pattern, rather than an approximate pattern matching detection based on similarity research, as in computational intelligence (Arcelli F. and al, 2004). Then, we have automated this detection using OCL rules supported by the NEPTUNE platform. Lastly, we have tested these rules on industrial models and standard meta-models.

There are several possibilities to produce alternative models:

The first one is to analyze the structure of patterns and to make transformations on their structure (classes and relationships between classes) in order to denature them (move responsibilities between two roles, replace or invert some relationships …). If this solution should permit to list exhaustively the set of alternative models, it may cause a combinative explosion of possible cases, among which a lot of them would have a limited interest. Indeed, they would not be found in usual models because they would be overly artificial or too different from a standard design.

The second one is to collect a set of models which don't use pattern but solve a problem solvable with a pattern. Next, it consists in extracting the models which may be considered as an alternative model.

For this study, we have chosen the last one. We have organized an experimentation which consists in designing the seven standard problems solvable with the seven structural patterns, in UML notation. We have decided to use examples presented in the "motivations" section of the GOF catalogue, when they were relevant. Each problem admitted a solution using a pattern, but experimenters have solved these problems without knowledge on patterns. From three hundred models obtained, we have selected eleven of them which presented significant structural variants with related patterns, the others were either an incorrect or duplicated design. We have considered these models valid because they permitted to solve the problem and they respected pattern constraints on architecture imposed by the intention. For example, the *Composite* pattern constraints are a unique access point for the client and a recursive composition of objects available. Each model obtained (between two and five for each pattern), constituted a plausible alternative to one pattern. To detect these alternatives in a model, we have analyzed their structural features, as we would have done it if we had wanted to redocument design models in detecting patterns.

Each alternative model is characterized like a pattern. A set of structural features is associated with each alternative model role. For the moment, these features concern inter-class relations only: i.e. associations, generalizations and aggregation/composition links, but neither interfaces nor classes semantics. We have deduced these features both associating corresponding pattern roles to each class of alternative models, and studying their structure.

To detect these features in a model, we have established a backtrack method which locates a class able to play a reference role in the pattern to identify. This role is the root of backtrack stage, chosen because from a class playing this role in a model. It permits to access directly to all the other classes of the model. Then, the method tries to assign the other roles of the pattern to the other classes of the model. For instance with the *Composite* pattern, we have searched for classes being able to play the « Component » role. To finish, we have tried to assign the « Composite » and « Leaf » roles to the other related classes and whose properties correspond to the remarkable features we wished to locate. This method has enabled us to deduce the generic search algorithm below.

```
In: UML model AND set of remarkable features associated to each role
of alternative model.

Find classes satisfying remarkable features of reference role
For each candidate class do
      Find classes associated in the model
      For each class associated do
            If it satisfies remarkable features of an other role
            Then
                  Affect the role to this class
            Fi
      Done
Done

Out: set of classes may have each role of alternative model
```

So, for each UML model, the algorithm finds all the occurrences of one alternative model. Each time one applies this algorithm for each alternative model associated to a given pattern, one retrieves all substitutable model fragments by the pattern itself. Therefore, this method is determinist and the result is complete with the set of alternative models dedicated to the pattern.

3. Structural variants of a Composite pattern

We would like to exemplify here a problem solvable with the *Composite* design pattern. Then, we will present four alternative models for this pattern, and we will try to explain how to obtain them with a perturbation composition. Lastly, for each alternative, we will summarize their remarkable features in a table. Each alternative model presented here is taken from our experimentation.

The problem "*Design a system enabling to draw a graphic: a graphic is composed of lines, rectangles, texts and images. An image may be composed of other images, lines, rectangles and texts.*" may be solvable with the *Composite* design pattern. The diagram represents its instantiation. If we perturb this model, we will obtain alternative models. To precisely target these transformations, it is necessary to use strong points of patterns and to withdraw them so that they become weak points. In the *Composite* pattern, the strong points are the maximal factorization of the aggregation relationships and a unique protocol for all compositions of instances.

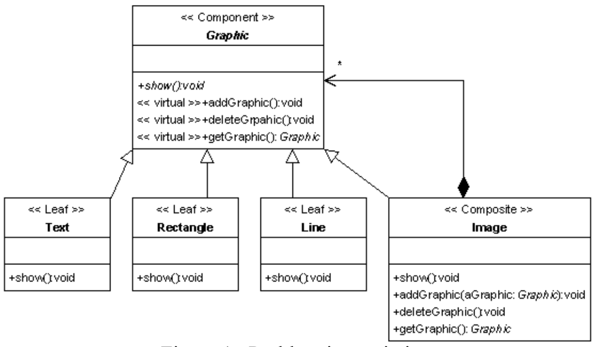

Figure 1 : Problem instantiation

First, if we use the instantiated pattern, and if we replace the inheritance link by aggregation links and aggregation links by inheritance links, we obtain a first alternative model. We have named it: **development of the composition on « Component »**. In this case, the composition is expressed by using directly the aggregation

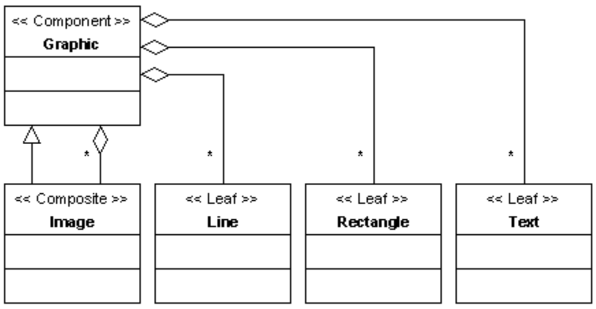

Figure 2 : Alternative model 1

concept. This solution is valid, since the recursive composition is possible, even if with coding, the "*Graphic*" class will be forced to store all the composition information. This will cause the multiplication of iterators on every element of the hierarchy. The use of the pattern would have avoided the redundancy of aggregation links, simplifying the structure and coding.

In the first alternative model, if we replace inheritance links by its aggregation equivalence, we obtain a second model that we have named: **development of the composition on « Component » and « Composite »**. This model reproduces the *Composite* pattern structure, by forgetting any factorization concept. It clearly appears that "*Graphic*" contains all the other classes, and "*Image*" too, by containing itself. This solution is valid, even if the factorization lack will produce a "bad smelling" code.

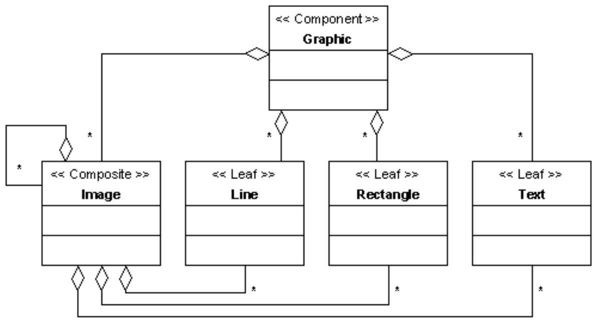

Figure 3 : Alternative model 2

From the first alternative model, by replacing the inheritance link by aggregation, we obtain a model called **recursive aggregation**. This model looks like the second one, but "*Image*" is not composed of itself but of the "*Graphic*" class. So, the code will be less "smelly" than the one obtained by the Model 2, but, the generalization lack will result in a code duplication.

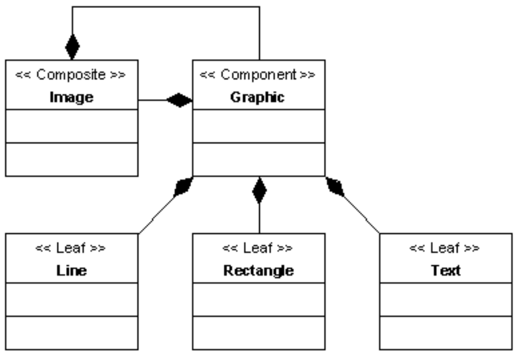

Figure 4 : Alternative model 3

From the original model, by developing aggregation from « Composite » to « Component » on every sub-class of « Component », we have obtained a new alternative model. We have named it: **development of the composition on « Composite »**. The composition is expressed such as it is described in the statement. We have located in this model that an image is composed of other images which could be composed of lines, rectangles or texts. The "*Image*" generalization is only there to be used as an access point to the client. However, the fact that "*Line*", "*Rectangle*" and "*Text*" do not inherit from "*Graphic*" will cause code duplications with excessive use of delegation.

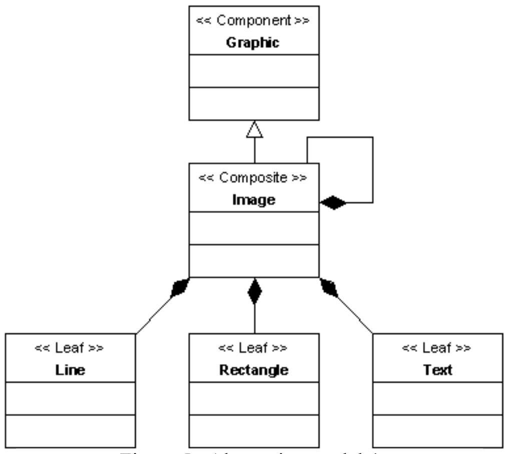

Figure 5 : Alternative model 4

In this experiment, the *Composite* pattern has a lot of alternative models. There are some interesting ideas to compose objects, the recurring characteristics remaining logically aggregations. The most frequent errors are a lack of aggregation relations factorization, and the loss of « Component » common interface. Inheritance has not been sufficiently used. The diversity of alternative models may cause the multiplication of detections of the same instance of the *Composite* pattern. However, for now, our detection system is limited by the lack of semantic interpretation. Indeed, a set of aggregations in a model does not have the same semantics as the ones of a corresponding alternative model.

To detect these alternative models, we have used these remarkable features:

Alternative model	Reference role	« Component » features	« Composite » features	« Leaf » features
Development of the composition on « Component »	« Component »	Classes having at least two aggregations	Classes both sub-class and aggregated with « Component »	Classes not sub-class and aggregated with « Component »
Development of the composition on « Component » and « Composite »	« Component »	Classes with at least two aggregations without any reflexive one	Classes aggregated with « Component » and aggregation of other classes, including itself	Classes aggregates with « Component » and without any reflexive aggregation
Recursive aggregation	« Component »	Classes with at least two aggregations	Classes both aggregated and aggregation of « Component »	Classes aggregated to « Component » but no aggregation of « Component »
Development of the composition on « Composite »	« Composite »	« Composite » super-class (detected after « Composite »)	Classes with a super-class and with at least two aggregations (with at least one reflexive one)	Classes aggregated to « Composite »

4. Alternative models for other GOF patterns

Our experiment has concerned all structural GOF patterns. *Bridge* and *Decorator* patterns are detectable thanks to the same method. For the others, we have been forced to adapt the method. To detect *Adapter*, we have needed to add new visibility stereotypes in UML, for example: « Black box » or « Glass box ». It will be possible to impose a set of rules defining a particular notation to a designer who wants to use our system of detection. The remarkable features of *Facade* pattern integrate metrics: the coupling between packages, the multiplicities between classes, the number of associations for a class… It seems to be a solution with all the alternative models which do not have any structural characteristics, but quantitative characteristics. However, *Flyweight* and *Proxy* patterns cannot be detected with our algorithm. These patterns increase the structural complexity of the models, which does not allow the detection of remarkable structural properties. They solve design problems, but with a non functional system orientation. Details for this work are available in (Bouhours*,* 2006).

We have studied creational patterns too, while trying to find structural characteristics starting from a fixed problem. Succeeding in applying all the creational patterns to the same problem permits us to predict that it will be difficult to obtain consequent alternative models by making the same type of experiment. Indeed, they are already each other's alternative models, since they substitute themselves for the same problem. The differentiation of these patterns is related to the object creation, so problems should contain non-functional requirements, which will be difficult to integrate in remarkable characteristics. Moreover, one fine dynamic system analysis will have to be planned to locate the adequacy between the instances of creative classes and the user classes of these same instances.

As for behavioral patterns, with the same type of experiment, it should be possible to deduce the remarkable characteristics of dynamic alternative models. Once done, it should be possible to deduce the remaining patterns, using the relations suggested by the GOF. The existing relations between patterns make us suppose that it is possible to deduce the other patterns by successive iterations. For example, if a *Composite* pattern is detected and validated, it will be able to deduce that the use of a *Builder* is judicious, by analyzing the new model obtained.

5. Conclusion

In order to validate our approach, we have applied OCL rules on XMI models. So, it has been necessary to implement detection of each alternative model by several rules. When a rule is not validated in the model, the NEPTUNE platform returns the context of the error, which is the model fragment substitutable by a pattern. To do so, we have written OCL rules so that their return value is always false when an alternative model has been identified. In a first stage, we have applied these rules on industrial models and then on OMG meta-models (Bouhours*,* 2006).

For each studied pattern, a considerable set of alternative models will correspond, multiplying the number of OCL rules to be written. By meta-modeling alternative models, it would be possible to generate automatically OCL rules, on the basis of remarkable characteristics associated to each alternative model.

In addition, before pattern integration, we reckon that it will be necessary that the models respect a minimal set of object oriented properties which will improve pattern detection. Indeed, if design lacks are present in the model, the detection method may malfunction. A set of inciting rules will have to be defined to assist the designer to make his models more relevant. These rules may be defined thanks to fundamental properties common to all patterns and thanks to the GRASP patterns.

As our approach remains at the design level, we would like to use dynamic models which may serve to identify behavioral, creational, and even structural patterns. In industrial models, we are currently limited by the lack of dynamic model views. Therefore, reverse engineering techniques may help us to rebuild dynamic views on subject system.

6. References

Arcelli Fontana F., Raibulet C., Tisato F., “Design Pattern Recognition”, IASSE 2004: 290-295

Bouhours C., Détection de particularités structurelles de modèles pour l'injection de patrons de conception[†], Master de recherche, Université Paul Sabatier, Toulouse, 2006.

Chikofsky E. J. and Cross J. H., “Reverse engineering and design recovery: A taxonomy”. IEEE Software, 7(1) : page 13 to 17, Jan. 1990.

Leblanc H., Millan T., Ober I., “Démarche de développement orienté modèles : de la vérification de modèles à l’outillage de la démarche”[‡], in *Ingénierie Dirigée par les Modèles, Paris*, page 125 to 139 ,30 June to 1 July 2005.

Gamma E., Helm R., Johnson R., Vlissides J., “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley Professional, 1995.

Guéhéneuc Y. G. and Albin-Amiot. H., “Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects”. in *proceedings conference on TOOLS*, pages 296 to 305, July 2001.

Neptune, Nice Environment with a Process and Tools using Norms - UML, XML and XMI - and Example, [w] http://neptune.irit.fr, 2003.

[†] M*odel structural features detection for design pattern injection*
[‡] *Model driven process : from static model checking to process tooling*