Pattern and Spoiled Pattern Detection through an Information Retrieval Approach

Nadia Bouassida
Institut Supérieur d'Informatique et de Multimédia
Université de Sfax, Tunisie
Email: Nadia.Bouassida @isimsf.rnu.tn

Hanêne Ben-Abdallah
Faculté des Sciences Economiques et de Gestion
Université de Sfax, Tunisie
Email:Hanene.BenAbdallah@fsegs.rnu.tn

Abstract—**Design patterns provide for a higher software quality and a reduced development cost. However, to reach these benefits, designers are expected to have a good understanding and experience with design patterns, which is not evident to acquire. Another way to benefit from design patterns is by assisting designers in their detection/identification within a given design in order to improve it.**

Since the *exact* structural instantiation of a pattern is less frequent to find within a design, the identification process should account for variations of the design with respect to the pattern. It assists the designer by showing the pattern elements in terms of the design which can be validated with respect to the classes, attributes, methods and relations of the pattern: the designer can add/remove some elements from the design in order to ensure a good instantiation of the identified pattern. However, not all structural variations of a pattern are tolerated; in fact, some variations may result in non-optimal instantiations of the pattern, *a.k.a.* spoiled patterns. In this case, the identification process can assist the designer by proposing corrections for an acceptable pattern instantiation.

Within this design context, we propose a method that identifies design patterns and spoiled patterns through an XML document retrieval approach. This latter provides for the possibility of tolerating structural variations between the design and the searched pattern. In addition, our pattern identification method can be parameterized in order to delimit the degree of acceptable variations.

Index Terms—**Design pattern identification, pattern instantiation, XML document retrieval.**

I. Introduction

Design patterns [9] are generic solutions for often occurring problems. Being proven solutions, proposed by experts, they promise several reuse benefits, such as high quality software, faster and lower cost software development. However, to attain these benefits, a designer must overcome the difficulties inherent to first understanding and then applying design patterns. In fact, even an experienced designer would spend a considerable time understanding, identifying and instantiating/reusing design patterns pertinent to his/her applications.

A straight forward way to benefit fully from design patterns is to assist an inexperienced designer to improve his/her design by identifying, in the design, instantiations of design patterns. On the other hand, since exact instantiations of a design pattern is less frequent (and is less problematic), an exact pattern identification method is, therefore, of limited use. Instead, pattern identification should look for structures that "resemble" a design pattern. By resembling, we mean structures that vary from a design pattern by adding/removing some elements (classes, attributes, methods, relations). The pattern identification method can, in this case, assist the designer in restructuring his/her design in conformance with the pattern found.

However, while tolerating pattern instantiations with variations, the identification method should watch out for non-tolerated variations, and in particular *spoiled patterns*. A spoiled pattern is a structure that allows to instantiate inadequate solutions for a given problem, where requirements are respected but architecture is improvable [17]. As an example of a spoiled pattern, an observer pattern where observers are subclasses of the subject class, a composite pattern where the leaf classes do not inherit from the composite class but are connected to the composite by a composition relation.

To offer assistance through pattern identification, several approaches propose to determine the potential similarities of the structure, the class names and/or method names between the design and a given pattern. These approaches differ mainly in the pattern concepts they consider (i.e., only the structure, the structure and the methods) and the degree of structural discordance they tolerate: exact match [4] or partial match [6], [13]. All methods that tolerate structural discordance between the design and a pattern treat all pattern elements equally. However, while some elements can be deleted in a design resembling a pattern, others representing the essence of the pattern (its core) should not; otherwise the pattern would be lost and/or spoiled.

In this paper, we present a new pattern identification technique that can: 1) be used to identify the structure, class names and participant roles of the pattern, 2) identify the spoiled patterns, and 3) take into account the degree of variability of a pattern. In addition, once a similarity is found, the identified design fragment is presented with the pattern roles and variability. This presentation assists the designer in better understanding the pattern through his/her application and in validating its instantiation. For this, we propose to use the P-UML design language [3], a UML profile for patterns.

More specifically, our identification technique reuses an XML document retrieval approach where the pattern is seen as the XML query and the design as the XML document where the query is searched. It relies on the context resemblance function [11] to compute the similarity potential of the design structure and the pattern. One advantage of this approach is that it is applicable to account for both the structure and methods in the pattern. A second advantage is that it accommodates design variability with respect to the pattern structure without losing the pattern essence or spoiling it.

The remainder of this paper is organized as follows. Section 2 overviews currently proposed approaches and tools for pattern and spoiled pattern identification. Section 3 first summarizes the basic concepts of XML document retrieval in general and the P-UML design language. Section 4 presents our approach for pattern identification and illustrates it with the composite pattern and a spoiled composite pattern. Section 5 summarizes the paper and outlines our future work.

II. CURRENT PATTERN IDENTIFICATION APPROACHES

Several works have been interested in pattern identification but for different purposes. For reverse engineering purposes, several proposals addressed the problem of automating the identification of design patterns in source code, *cf.*, [10], [7], [15]. For instance, Lee *et al.*, [10] use a static analysis to collect the structural aspect of software and a dynamic analysis to elucidate dynamical aspects of the software during the program execution such as the message passing between objects.

For both reengineering and code improvement purposes, Albin-Amiot et al. [1] present a toolset to help OO software practitioners design, understand, and re-engineer a piece of software using design patterns. Their prototype tool uses a constraint satisfaction technique to detect patterns within a given source code. It has the advantage of taking into account refractoring aspects and identifying distorted versions of the pattern in a source code. In addition, it can transform the source code so that it complies with the detected design pattern.

Besides the code, other works extract design patterns from a design. For example, the work of Tansalis [13] proposes a design pattern detection methodology based on similarity scoring between graph vertices. The graphs of the searched pattern and the examined design are encoded as matrices. These latter are then used to compute a similarity matrix. This matrix is calculated using the similarity scoring algorithm which has been proposed by Blondel et al. [2] The main drawback of the similarity scoring approach is the convergence time which depends on the graph size of the design.

Also within this matrix similarity-based approach, Dong *et al.* [5] use a template matching method to calculate the normalized cross correlation between the pattern matrix and the matrix representing a design segment. A normalized cross correlation shows the degree of similarity between the pattern and the design segment.

On the other hand, Florijin and Meijers [18] proposed a tool capable of detecting all pattern instantiations in an OMT design. The tool implements a graph matching technique. Once a pattern template is detected, the tool associates a set of roles to the classes composing the detected pattern instantiation. This information can assist the designer in validating the correct instantiation of a pattern. However, it does not tolerate any discordance between the design and the pattern.

A second purpose of pattern identification within a design is to improve the quality of the design. Within this context, Bergenti and Poggi [19] propose a tool, called IDEA, to improve UML designs (class and collaboration diagrams) using automatic pattern detection. Their method relies on a knowledge base where each pattern is described in terms of a structure template and a collaboration template (described as PROLOG rules). For example, to detect the Composite pattern, the system searches all triplet classes having the template structure *identical* to the composite. Thus, this method handles only an exact instantiation of the pattern. When IDEA finds a pattern instance, a set of design (PROLOG) rules are verified to test if the design could be improved. Then a set of critiques are proposed as possible design improvements. It is worth noting that the critiques/proposed improvements are pattern specific and they require a high level of understanding of both the design and the pattern.

The work of El Boussaidi and Mili [7] represents the design problem the pattern is meant to solve explicitly. It aims at recognizing occurrences of the modeled problem solved by a design pattern, which is then transformed according to the solution proposed by the design pattern. This work uses a meta model of the pattern problem to identify its instances in a given design. Once a problem is detected, it marks the appropriate entities and finally applies transformations to get the pattern solution. This work relies on graph modeling and transformation. One of its limits is that it focuses only on the pattern structure. However, among the essential constituents of a pattern (problem and/or solution) is the methods used.

Bouhours *et al.* [17] propose a detection approach for "bad smells" in a design that can be remodeled through the use of design patterns. A bad smell is any symptom that possibly indicates a design problem. The proposed approach can identify some spoiled patterns and their alternative model fragments. It uses a generator of OCL queries and a specific profile that encodes structural particularities of spoiled patterns. However, this work allows only exact matches with the spoiled patterns and considers only the structural information.

As summarized in table I, none of the proposed approaches combines the structural and dynamic aspects in their pattern identification. Except for Ka-Yee [21], none of the few works treating the dynamic aspect describes the behavior in terms of scenarios of ordered method invocations and tolerates behavioral variability. In fact, the dynamic aspect treated in the other approaches is limited to method calls between pairs of related classes, independently of the overall temporal behavior.

TABLE I. CURRENT PATTERN IDENTIFICATION APPROACHES

	Technique	Type	Tolerate variation	Aspect
[11]		Pattern, problem, ou spoiled	Yes/no	static & dynamic
[1]	constraint satisfaction	pattern	Yes	static
[13]	similarity scoring between graphs coded as matrices	pattern	Yes	static & partially dynamic (only method calls)
[5]	template matching between graphs coded as matrices	pattern	Yes	static & partially dynamic (only method calls
[7]	Meta-model of the problem and CSP	Pattern problem	Yes	static
[17]	OCL queries	Spoiled pattern	No	static
[21]	dynamic analysis & CSP	pattern	yes	dynamic

In addition, none of the proposed approaches for pattern detection can also detect spoiled patterns. Furthermore, none of these approaches instantiates a detected pattern within the examined design while highlighting the pattern variability and the role of each class in the design. Such information can assist the designer in understanding the pattern (or spoiled pattern) and validating its correct instantiation.

III. XML DOCUMENT RETRIEVAL AND PATTERN NOTATION

Our approach has a two-fold objective. The first objective is to identify correct and spoiled pattern instantiations within an application design, while tolerating certain variability. For this, we adapt an XML document retrieval technique that we overview in Subsection *A*. The second objective is to assist the designer in understanding and validating the instantiation of the identified pattern within the examined design. For this, we propose to use the P-UML notation which we briefly present in Subsection *B*.

A. XML document retrieval

XML document retrieval has been treated in the literature by several researchers. The most complete work has been proposed by Manning *et al.*, [11]. In this work, the authors adapt the vector space formalism for XML retrieval by considering an XML document as an ordered, labeled tree. Each node of the tree represents an *XML element*. The tree is analyzed as a set of paths starting from the root to a leaf. In addition, each query is examined as an *extended* query – that is, there can be an arbitrary number of intermediate nodes in the document for any parent-child node pair in the query. Documents that match the query structure closely by inserting fewer additional nodes are given more preference.

A simple measure of the similarity of a path c_q in a query *Q* and a path c_d in a document *D* is the following *context resemblance* function [11]:

$$C_R(c_q,c_d)=\begin{cases}\dfrac{1+|c_q|}{1+|cd|} & \text{if } c_q \text{ matches } c_d\\ 0 & \text{if } c_q \text{ does not match } c_d\end{cases}$$

Where:

- $|cq|$ and $|cd|$ are the number of nodes in the query path and document path, respectively, and
- c_q *matches cd* if and only if we can transform *cq* into *cd* by inserting additional nodes.

Note that the value of CR(c_q, c_d) is 1 if the paths c_q and c_d in *Q* and *D* are identical. On the other hand, the more nodes separating the paths of *Q* and *D*, the less similar they are considered, *i.e.*, the smaller their context resemblance value will be.

B . P-UML: a design pattern notation

Several UML-based formalisms for pattern representation (*cf.*, [8] , [14])) were proposed. To account for the variability of a pattern, some proposed languages are able to distinguish the "regular" methods from the *hook* and *template* methods in a pattern, *cf.*, [5] [12]. These two types of methods encapsulate the pattern extensibility: template methods define abstract and generic behavior, while hook methods provide their implementation. However, none of the proposed languages both shows the variability and guides potential instantiations of the pattern and identifies the elements, the structure and the role played by the elements of the pattern.

In response to the above shortages, we have proposed a notation for patterns, called P-UML [3], that is an extension of the UML class diagram. The extensions outline the roles played by both the classes as well as the methods within a design pattern. In addition, they link the pattern elements, and therefore help in visually distinguishing between different patterns used in a system design. Moreover, the extensions identify the pattern hot-spots and meta-patterns (template and hook methods). To illustrate the main concepts of the P-UML language, we will use the Composite pattern (Figure 1). The reader is referred to [3] for a detailed description of this language.

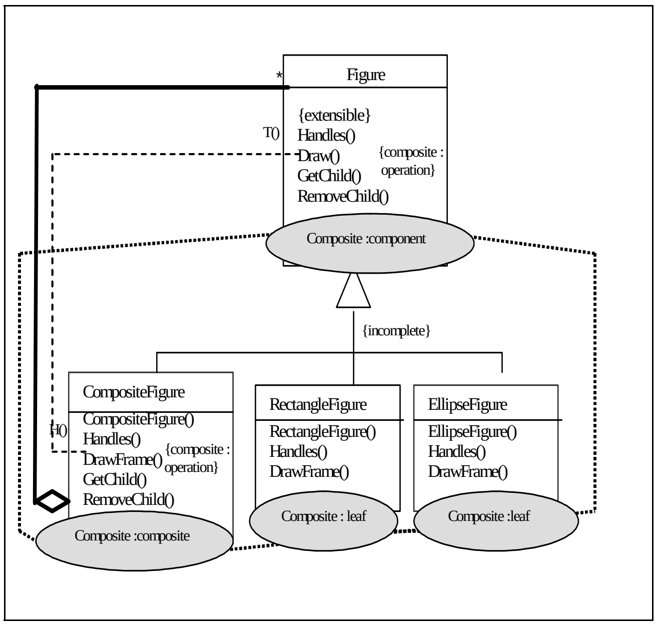

Figure 1. The *composite* pattern in the P-UML notation

Figure 1 shows an instantiation of the Composite pattern for an application in the graphical editor domain. The pattern participant roles (the ellipses) and their relationships are indicated in the instantiation. In addition, P-UML identifies the methods that play essential roles in the pattern: the Draw() method has been identified as a fundamental method in the composite pattern. A dashed line joins the hook Draw() and the template method DrawFrame(). The aggregation relation which is fundamental in the pattern is drawn with a highlight. Finally, P-UML delimits the pattern boundaries, which eliminates any confusion when multiple patterns are composed.

IV. Pattern detection technique

To detect patterns within a design, we take into account that a given pattern may be represented in various forms that differ from the basic structure without loosing the essence of the pattern. Thus, an exact pattern matching approach is insufficient.

On the other hand, the problem of finding an XML document (query) within a larger document while tolerating structural variations has been treated within the information retrieval domain. Several solutions were proposed to handle the structural differences that may exist between the query and a retrieved document. These solutions motivated us to convert design pattern detection into an XML document retrieval problem. More specifically, we consider a design pattern as an XML query and the design as the target XML document where the pattern is searched. In fact, since we consider the pattern and the examined design as two class diagrams, their transformation into XML documents is straightforward and can be handled by most existing UML editors. Furthermore, by transforming the pattern detection problem into an XML document retrieval problem, our approach can benefit from existing search engines.

```
<!ELEMENT classdef (name, inherit*,  composition*,
          association*,aggregation*, implements*, typdef*, op*)>
<!ELEMENT inherit (type*, incompletetag?) >
<!ELEMENT composition (type*) >
<!ELEMENT aggregation(type*) >
<!ELEMENT  typedef    (name+, type+) >
<!ELEMENT  predcalc    (predcalc*, type *) >
<!ELEMENT op (name, returntype+,param*)>
<!ELEMENT  param    (name+, type +) >
<!ELEMENT  name   (#PCDATA) >
<!ELEMENT type (#PCDATA)   >
<!ELEMENT incompletetag (#PCDATA)   >
```

Figure 2. DTD extract for the UML class diagram

To illustrate our method, we will focus on the structural features of the corresponding class diagram: the classes, generalizations, aggregations, compositions, etc. For this, we will use the DTD shown in Figure 2 to transform the pattern and the design into XML documents. Note that each tree in these XML documents is composed of class nodes interconnected by relation nodes (generalization, association, etc). In addition, each path in a tree contains relation nodes from the same type.

In XML document retrieval in general, the context resemblance function (CR) is calculated based on an exact match between the names of the nodes in the query and the document paths. However, for pattern detection, the nodes representing the classes are often different in the pattern from those in the design. Thus, we first need to calculate the resemblance values for the various matches between the class nodes in the query (pattern) and those in the design. Secondly, we need to take into account: 1) the number of times a given match between two class nodes is used to calculate CR; and 2) the importance of each relation in the pattern.

A. *Resemblance determination*

The resemblance between a pattern and a design starts by computing the resemblance between each path of the pattern to all the paths in the design. In this computation, we assume that the structural variability should be limited between the pattern and a potential instantiation in the design. That is, we assume that a design path may differ from a pattern path by adding at most N nodes compared to the longest path of the pattern. The larger the N, the more scattered the pattern instantiation would be in the design, which might loose the pattern essence.

To determine the resemblance between a pattern Q and a document D, we proceed as follows:

```
1.          L := the number of class nodes in the longest path in Q;
2.          N := the maximum number of intermediate/additional nodes in the design path;
3.          For each path $P_q$ in the pattern Q
3.1           For each path $P_d$ in the document D
3.1.1           If $P_d$ and $P_q$ have different types of relations
3.1.2           then CR($P_q$ , $P_d$) := 0
                else
                  //compare $P_q$ with all sub-paths in $P_d$ starting
                  // from different nodes
3.1.3             For s=1 to | $P_d$|-1
                    // tolerate at most w additional nodes
3.1.3.1             For w=1 to min(L+N, | $P_d$|-1)
3.1.3.1.1             $P'_d$ := $P_d$ [s .. s+w]
3.1.3.1.2             Compute CR($P_q$ , $P'_d$)
4.          Compute the weighed sum of all CR scores for all the paths and store them in CRMatrix(Q,D);
5.          Normalize CRMatrix(Q,D) by dividing each entry by the number of classes in D
```

In step 3.1.3, we consider that the match between the pattern path and the design path may not necessarily start at the root node; for this we need to consider all possible sub-paths of the design. These sub-paths start at different class nodes in P_d. In addition, since the structural difference between the pattern path and the design path is limited, then each sub-path can cover at most $L+N$ class nodes; thus the number of sub-paths to be considered is reduced. This in turn limits the temporal complexity of the algorithm. The tolerated maximal intermediate nodes N can be fixed by the designer.

In step 4, we sum up in *CRMatrix* the resemblance scores (*i.e.*, correspondences) between the classes of the design and the classes of the pattern. This weighted sum accounts for the importance of the relations in the pattern. Finally, in step 5, these scores are normalized with respect the total number of classes in the design; the final matching results are collected in *NormalizedCRMatrix* whose columns are the classes in the pattern and whose rows are the classes of the design. Now given this matrix, we can decide upon which correspondence better represents the pattern instantiation: For each pattern class, its corresponding design class is the one with the maximum resemblance score in *NormalizedCRMatrix*.

On the other hand, given two designs $D_1$and D_2, to decide upon which design better instantiates a pattern P, we first compute their normalized resemblance matrices. Secondly, we compute the sum of the normalized resemblance scores for all the matched pattern classes in $D_1$and D_2; the design with the maximum sum is the one that better instantiates the pattern.

Note that in the worst instantiation, each pattern class must be matched to at least one class in the design; thus, on average, the sum of the normalized resemblance scores of the matched classes should not be less than the number of classes in the pattern divided by the number of classes in the design.

B. Example: detection of the composite pattern

Graphics applications like drawing editors let users build complex diagrams out of simple components. The user can group components to form larger components. For instance, we can define classes for Text and Lines and classes that act as containers for these classes. However, when using these classes, the primitive classes such as Text and Line have to be treated differently from the container class: This is the composite pattern problem instantiated in the graphical editor application shown in Figure 1.

The composite pattern (Figure 3.b) applies to compose the objects in a tree structure where individual objects as well as the composed objects behave uniformly. Composed objects delegate the requests to the individual leaf objects.

To detect the composite pattern, we have to identify its structure. Let us try to identify it in the design of Figure 3.a. The XML corresponding paths are illustrated in Figure 4.

Table II shows a sample of the resemblance function scores corresponding to the design and pattern paths of Figure 4. Recall that some concepts are more essential in a pattern than others. In the composite pattern, let us consider that the aggregation relation is twice as important as the inheritance relation. Thus, when collecting the CR scores in the resemblance matrix, the score of the aggregation match is multiplied by two.

The normalized CR matrix identifies the composite design pattern correctly and indicates that the class A matches the *component* class, the class D matches the *composite* and consequently the class C matches the *Leaf* class. Note that the match score of the class C to *Leaf* is equal to the match score of C to *composite* (0.39); however, since D has been identified as *composite* with a greater matching score (0.725), then C is identified as *Leaf*.

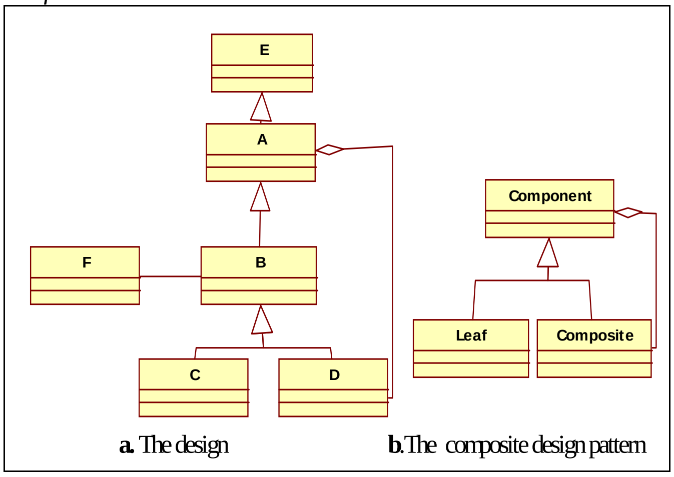

Figure 3. A sample design (a) and the composite pattern (b)

Furthermore, the sum of the maximum normalized CR for the nodes of the pattern (2.365) is greater then the threshold which is equal to 3/6; thus this identification is acceptable.

Given the above matching, we can represent the Composite pattern within the design through the P-UML language as shown in Figure 5. Through this representation, the designer can better understand the roles of his/her design classes as indicated in the identified pattern.

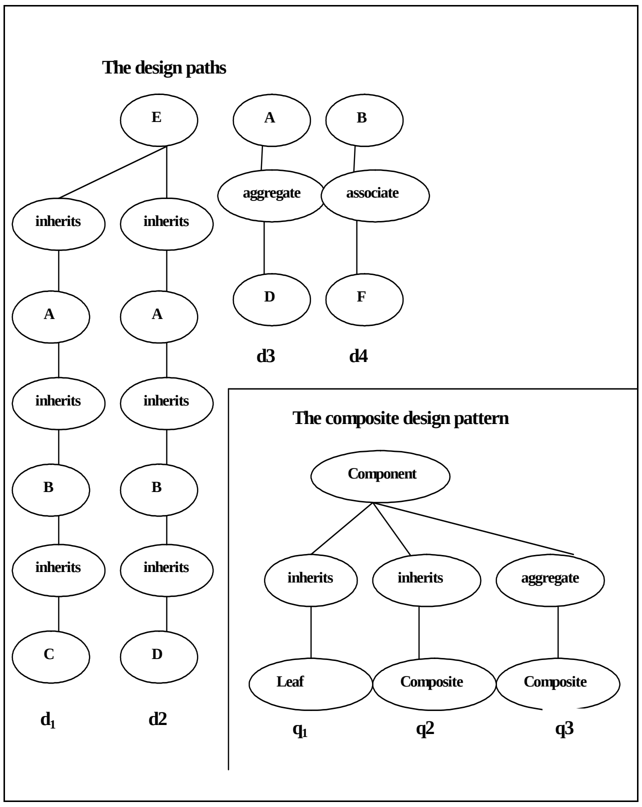

Figure 4. XML document trees for the example of Fig 3.

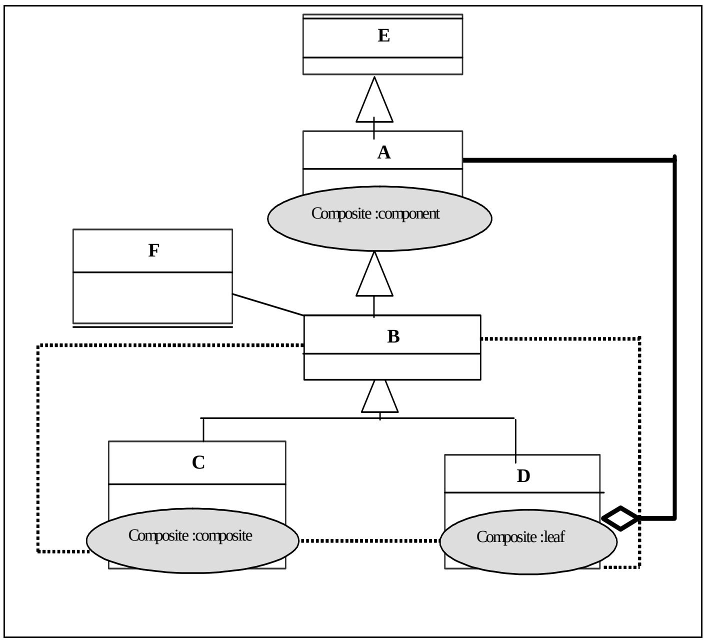

Figure 5. The identified composite pattern with P-UML

TABLE II. SAMPLE CONTEXT RESEMBLANCE SCORES AND NORMALIZED MATRIX

	component $\xrightarrow{\text{inherits}}$ leaf	**component $\xrightarrow{\text{inherits}}$ composite**	**component $\xrightarrow{\text{aggregates}}$ composite**
E $\xrightarrow{\text{inherits}}$ A	**CR(*cq*1 , *cd*1) =1** **If Component=E Leaf=A**	**CR(*cq*1 , *cd*1) =1** **if Component=E Composite=A**	**0**
A $\xrightarrow{\text{inherits}}$ B	**CR(*cq*1 , *cd*1) =1** **If Component=A Leaf=B**	**CR(*cq*1 , *cd*1) =1** **if Component=A Composite=B**	**0**
E $\xrightarrow{\text{inherits}}$ A $\xrightarrow{\text{inherits}}$ B	**CR(*cq*1 , *cd*1) =0.75** **If Component=E Leaf=B**	**CR(*cq*1 , *cd*1) =0.75** **if Component=E Composite=B**	**0**
A $\xrightarrow{\text{aggregates}}$ D	**0**	**0**	**CR(*cq*2, *cd*2) =2** **if Component=A Composite=D**
F $\xrightarrow{\text{associates}}$ B	**0**	**0**	**0**

$$\text{NormalizedCRMatrix(Design, pattern)} = \begin{array}{c} \\ E \\ A \\ B \\ C \\ D \\ F \end{array} \begin{array}{ccc} \text{component} & \text{composite} & \text{leaf} \\ \end{array} \left(\begin{array}{ccc} 5.9 & 0 & 0 \\ 7.5 & 1 & 1 \\ 4 & 1.75 & 1.75 \\ 0 & 2.35 & 2.35 \\ 0 & 4.35 & 2.35 \\ 0 & 0 & 0 \end{array} \right) / 6 = \begin{array}{c} E \\ A \\ B \\ C \\ D \\ F \end{array} \left(\begin{array}{ccc} 0.983 & 0 & 0 \\ 1.25 & 0.16 & 0.16 \\ 0.66 & 0.29 & 0.29 \\ 0 & 0.39 & 0.39 \\ 0 & 0.725 & 0.39 \\ 0 & 0 & 0 \end{array} \right)$$

C. *Detection of the spoiled composite pattern*

As we are able to detect design patterns, we are also able to detect spoiled patterns. An example of a spoiled composite pattern is illustrated in Figure 6. It shows a typical composite object structure of recursively composed graphic objects. The Figure class is composed of other Figures which could be composed of lines, texts and rectangles. In this spoiled pattern the pattern quality rules or the decoupling and extensibility properties of the pattern are not respected. Moreover, the fact that *Line*, *Text* and *Rectangle* do not inherit from *Graphic* will cause some duplication of code with excessive use of delegation [17].

A correction of this spoiled pattern is brought by the composite pattern which defines an abstract class that represents both primitive and container classes: the class *Graphic* shown in Figure 8.

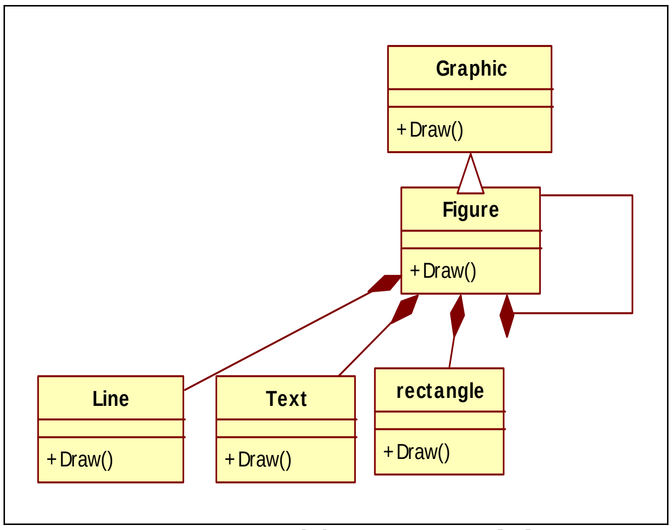

Figure 6. A spoiled composite pattern [17]

To detect the spoiled Composite, an abstraction of the spoiled pattern is necessary. The abstraction is shown in Figure 7. Now, let us consider the design fragment illustrated in Figure 9 and determine if it is similar to the spoiled composite abstraction illustrated in Figure 7.

Similar to pattern detection, the design is converted into XML trees as illustrated in a graphical format in Figure 10. Some of the context similarity function scores are illustrated in table III and the resulting Normalized matrix is shown below.

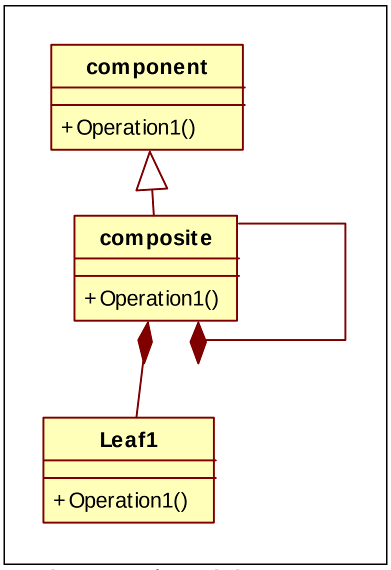

Figure 7: An abstraction of a spoiled composite pattern

The normalized CR matrix identifies the spoiled composite design pattern correctly. Thus, the class *Graphic* is identified as the *component* class, the class *Figure* is identified as the *composite* class. The classes *TextFigure*, *TriangleFigure* and *EllipseFigure* match the *Leaf* class. Note that the match scores of these classes to *Leaf* is equal to their match scores to *composite* (1/6); however, since the *composite* has been identified with a greater matching score (10/6), then they are identified as *Leaf*.

Given the above matching, we can substitute the spoiled Composite pattern with a correct instantiation of a composite pattern.

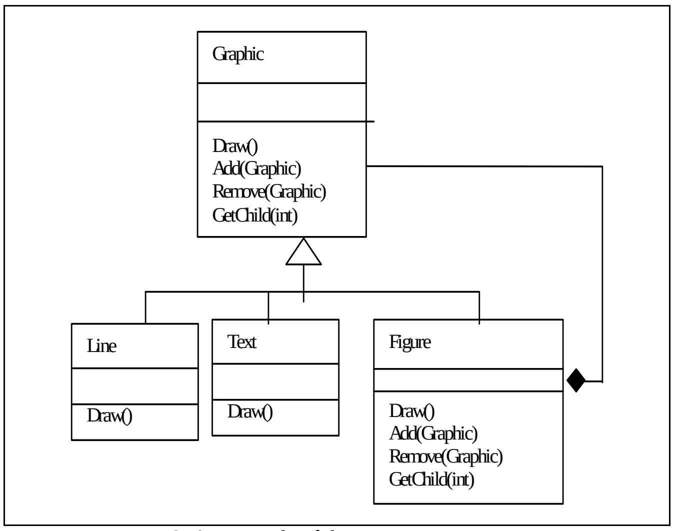

Figure 8: An example of the composite pattern

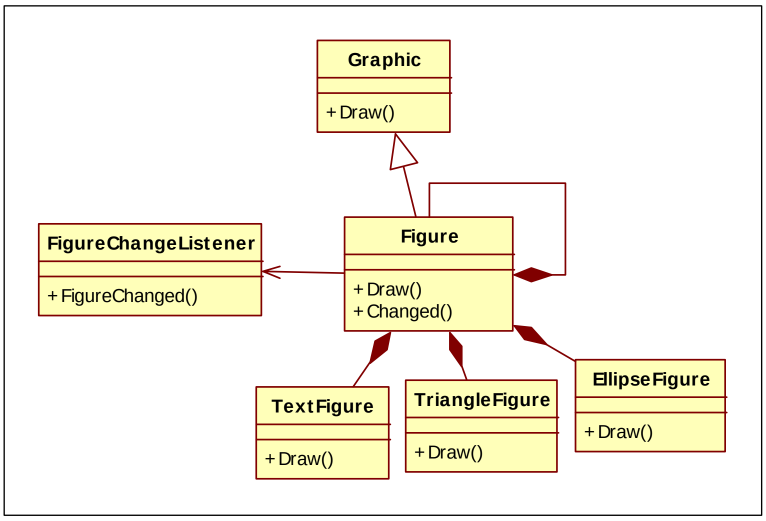

Figure 9: a fragment of a design

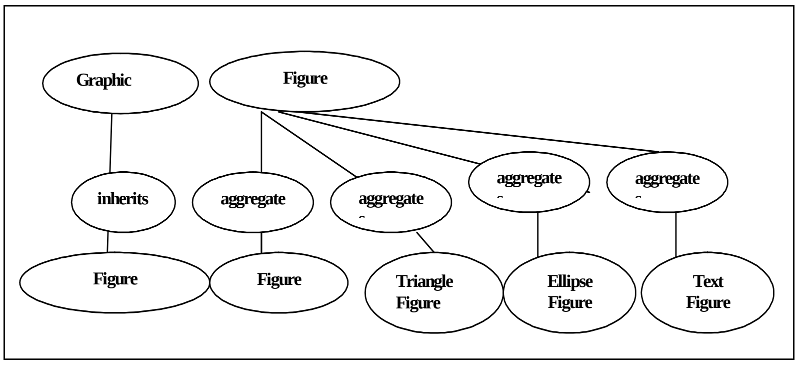

Figure 10: XML document trees for the spoiled composite pattern

TABLE III. CONTEXT RESEMBLANCE SCORES FOR SPOILED COMPOSITE PATTERN DETECTION

	Component $\xrightarrow{\text{inherits}}$ *Composite*	*Composite* $\xrightarrow{\text{aggregates}}$ *Composite*	*Composite* $\xrightarrow{\text{aggregates}}$ *Le af*
Graphic $\xrightarrow{\text{inherits}}$ *Figure*	$CR(c_q , c_d) = 1$ *if Component=Graphic* *Composite=Figure*	*0*	*0*
Figure $\xrightarrow{\text{aggregates}}$ *Figure*	*0*	$CR(c_q , c_d) = 1$ *if Composite=Figure* *Composite=Figure*	$CR(c_q , c_d)) = 1$ *if Composite=Figure* *Leaf=Figure*
Figure $\xrightarrow{\text{aggregates}}$ *EllipseFigure*	*0*	$CR(c_q , c_d) = 1$ *if Composite=Figure* *Composite= EllipseFigure*	$CR(c_q , c_d) = 1$ *if Composite=Figure* *Leaf=EllipseFigure*
Figure $\xrightarrow{\text{aggregates}}$ *TextFigure*	*0*	$CR(c_q , c_d) = 1$ *if Composite=Figure* *Composite=TextFigure*	$CR(c_q , c_d) = 1$ *if composite=Figure* *Leaf=TextFigure*
Figure $\xrightarrow{\text{aggregates}}$ *TriangleFigure*	*0*	$CR(c_q , c_d) = 1$ *if composite=Figure* *composite=TriangleFigure*	$CR(c_q , c_d) = 1$ *if composite=Figure* *Leaf= TriangleFigure*
Figure $\xrightarrow{\text{asociates}}$ *FigureChangeListener*	*0*	*0*	*0*

$$\mathbf{NormalizedCRMatrix(PatternPb, Design)} = \begin{array}{l} \\ \textit{Graphic} \\ \textit{Figure} \\ \textit{EllipseFigure} \\ \textit{TextFigure} \\ \textit{TriangleFigure} \\ \textit{FigureChangeListener} \end{array} \begin{array}{c} \begin{array}{ccc} \textit{Component} & \textit{Composite} & \textit{Leaf} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array} /6$$

V. Conclusion

Design patterns ensure an improvement of design quality, traceability and a better documentation [9]. However, the difficulty of their detection and instantiation reinforces the need for a technique that automates these tasks. This paper proposes a new approach for pattern and spoiled pattern detection and instantiation.

The proposed approach adapts an XML document retrieval technique. That is, it considers a design pattern (or spoiled pattern) as an XML query to be found in an XML document representing a design. It uses the context similarity function [11] to determine the most probable correspondences between the classes of the design and those in the pattern (or spoiled pattern). It has the advantage of tolerating certain structural differences in the design compared to the (spoiled) pattern; the designer can fix a threshold below which the differences are un-tolerated. In addition, our approach can be applied for both structure and method correspondences. Furthermore, once a (spoiled) pattern instantiation is detected, the correspondence information produced by our approach can be exploited to represent the design fragment with the (spoiled) pattern elements. This representation assists the designer in understanding the found (spoiled) pattern within the context of his/her application. Moreover, it allows him/her to validate (correct) the instantiation of the (spoiled) pattern.

Our future works deal with three axes. In the first, we are examining how to add more intelligence in our assistance for the recognition of pattern problems inside a design: how to alleviate the search task by adding priorities. In the second axe, we are seeking to exploit the information of the collaboration diagrams to best handle the (spoiled) pattern detection. In the third axe, we are looking into the formalization of design patterns. This will provide us with two benefits: 1) precise definition of patterns, and 2) analysis facilities to validate a pattern instantiation.

References

[1] H. Albin Amiot, P. Cointe, Y. G. Guéhéneuc, "Un meta-modele pour coupler application et détection des design patterns", *L'objet*, N° 8, 2002, pp1-18.

[2] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren, "A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching", *SIAM Review*, Vol. 46, N°. 4, 2004, pp. 647-666.

[3] N. Bouassida and H. Ben-Abdallah, "Extending UML to guide design pattern reuse", *Sixth Arab International Conference On Computer Science Applications*, March 2006.

[4] K. Brown, "Design reverse-engineering and automated design pattern detection in Smalltalk". Technical Report TR-96-07, University of Illinois at Urbana-Champaign, 1996.

[5] J. Dong, "UML extensions for design pattern compositions", *Journal of object technology*, Vol. 1, N° 5, 2005, pp 149-161.

[6] J. Dong , Y. Sun and Y. Zhao, "Design pattern detection by template matching". SAC'08, Ceara, Brazil, March 16-20, 2008 .

[7] G. El Boussaidi and H.Mili, "Detecting patterns of poor design solutions by using constraint propagation", In MODELS, *Proceedings of the 11th international conference on model driven engineering languages and systems*, 2008.

[8] M.F. Fontoura, W. Pree and B. Rumpe, "Extending UML to improve the representation of design patterns", *JOOP*, Vol. 13, N°11, 2001, pp. 12-19.

[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides, 1995, *Design patterns: Elements of reusable Object Oriented Software*, Addisson-Wesley, Reading, MA, 1995.

[10] H. Lee, H. Youn, E. Lee, "A design pattern detection technique that aids reverse engineering". In the *International Journal of security and applications*. Vol 2, N°1 January, 2008.

[11] C.D. Manning, P. Raghavan and H. Schütze, *An introduction to information retrieval*, Cambridge University Press, England, 2008.

[12] W. Pree, "Meta-patterns: a means for capturing the essentials of object-oriented designs", *Proceedings of the 8th European Conference on Object Oriented Programming*, Bologna, Italy, 1994.

[13] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides and S. T. Halkidis, "Design pattern detection using similarity scoring" In IEEE transactions on software engineering, vol 32, N°11, san franciso,USA,2006.

[14] Y. Sanada and R. Adams, "Representing Design Patterns and Frameworks in UML-Towards a Comprehensive Approach", *Journal of Object Technology*, Vol. 1, N°2, July-August, 2002.

[15] N. Shi and R.A. Olsson. "Reverse Engineering of Design Patterns from Java Source Code", P*roceedings 21st IEEE International Conference on Automated Software Engineering*, 2006.

[16] XML Metadata Interchange: OMG Document ad/98-07-03, July 6, 1998

[17] C. Bouhours, H. Leblanc and C. Percebois, "Bad smells in design and design patterns". In *Journal of Object Technology*, Vol 8, N°3, May-June, 2009.

[18] G. Florijin, M. Meijers and P. Van Winsen, "Tool support for object oriented patterns", In *Proceedings of the European conference on object oriented programming: ECOOP*, 1997.

[19] F. Bergenti and A. Poggi, "Improving UML design pattern detection", In p*roceedings of the 12th international conference on software engineering and knowledge engineering SEKE*, 2000.

[20] N. Bouassida and H. Ben-Abdallah, "Structural and behavioral detection of design patterns". In: *International Conference on Advanced Software Engineering & Its Applications (ASEA)*, December 10-12, Jeju Island, Korea, LNCS Proceedings, Springer, 2009.

[21] J. Ka-Yee Ng and Y. G. Gueheneuc, "Identification of behavioural and creational design patterns through dynamic analysis", *Proceedings of the 3rd International Workshop on Program Comprehension through Dynamic Analysis (PCODA)*, October 2007, pp 34-42.