Design improvement through dynamic and Structural pattern identification

Nadia Bouassida*, Hanene Ben-Abdallah**

Mir@cl Laboratory,

*Institut Supérieur d'Informatique et de Multimédia, Sfax University, Sfax, Tunisia

**Faculté des Sciences Economiques et de Gestion, Sfax University, Sfax, Tunisia

Nadia.Bouassida @isimsf.rnu.tn, Hanene.BenAbdallah@fsegs.rnu.tn

Despite the high design quality they can produce, design patterns remain difficult to reuse directly within a development process. To overcome this problem, several works proposed approaches and/or tools to identify design patterns for code improvement and/or re-engineering purposes. Overall, these propositions either rely mainly on the static aspect of the pattern, or require an exact match between the design pattern and its instantiation in the design/code. In this paper, we propose to improve designs through recognizing occurrences of patterns using both static and dynamic information. To propose improvements in a design, we adapt an XML document retrieval technique and, thus, tolerate design variability with respect to non essential parts of the pattern.

Keywords: design pattern identification; design pattern instantiation; XML document retrieval

1. INTRODUCTION

The quality of a design highly influences the quality of the corresponding code and eventually the performance of the produced system. The design-code tight coupling is ever increasing with the new tools for automatic code generation and the new development approaches based on automated model transformations, *e.g.*, creating graphical editors with the Eclipse Graphical Editor Modeling Framework (GMF). This coupling motivated several works trying either to improve the quality of a given design and/or to produce a high quality design. Independently of their objective, the proposed works agree on the importance of design patterns [8] reuse to ensure certain quality criteria, *e.g.*, minimal structure, conventional functional properties, etc.

However, one major problem behind benefiting from design patterns is the difficulties inherent to first understanding and then applying them to a particular application. In fact, even an experienced designer would spend a considerable amount of time understanding, identifying and instantiating/reusing design patterns pertinent to his/her applications. Hence, a promising way to benefit from design patterns is to assist an inexperienced designer to improve his/her design by finding design patterns that could better structure his/her design.

To offer such assistance, several approaches propose to determine the potential similarities of the structure and/or method calls between the design and a given pattern. These approaches differ mainly in the pattern concepts they consider (*i.e.*, only the structure vs. the structure and the methods) and the degree of structural discordance they tolerate (exact match [2] [11] vs. partial match [5], [14]).

Evidently, pattern identification should tolerate a certain degree of structural discordance between the design and a pattern; indeed, an instantiation of a pattern often adds details (specific classes, attributes, method invocations ...). In addition, while some elements can be deleted in a design approximating a pattern, others representing the essence of the pattern (or its core) should not; otherwise the pattern would be lost. In other words, the pattern elements should not be treated equally. However, all pattern identification approaches that tolerate structural discordance treat them equally. Furthermore, most of the existing pattern identification approaches focused on identifying the pattern static structure and neglected the pattern dynamic aspects; this can produce imprecise results for patterns with a similar structure. The few works that treated the methods dealt only with method calls [14] [5] which represent partially the dynamic aspect of a pattern: the method invocation order is often very important, essentially, in behavioral and creational design patterns.

In this paper, we propose to improve designs by relying on the results of our pattern identification technique [4]. 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 in which the query is searched. It relies on a context resemblance function [12] to compute the similarity potential between the design structure and behavior, and the pattern. One advantage of this approach is that it is applicable to account for both the structure and dynamic aspects of the pattern. A second advantage is that it accommodates design variability with respect to the pattern structure without losing the pattern essence.

The remainder of this paper is organized as follows. Section 2, overviews currently proposed approaches for pattern identification. Section 3 presents our approach for pattern identification in terms of its structure and dynamic aspects. Section 4 shows how the results of pattern identification can be used to propose improvements in a design. In Section 5, the identification and improvement approaches are illustrated through two examples: The first example illustrates the structural pattern identification, and the second illustrates all the approach through the observer design pattern and a fragment of the JHotDraw framework for graphical drawing editors [16]. Section 6 summarizes the paper and outlines our future work.

2. CURRENT PATTERN IDENTIFICATION APPROACHES

Table 1 presents an overview of current approaches for the identification of design patterns. As illustrated in this table illustrates, the proposed approaches differ in their objective: 1) software re-engineering where the main is to detect design patterns at the code level to assist in better understanding parts of the program; vs. 2) design quality improvement. As a result, the proposed approaches are applicable either at the source code, or the design (mainly UML class diagrams).

TABLE 1: The current pattern identification approaches

	Objective	Level	Pattern aspect	Technique	Limits/advantages
Lee <i>et al.</i> [11]	software re-engineering	Source code	static & dynamic		Only exact matching of patterns can be identified.
Albin - Amiot <i>et al.</i> [1]	software re-engineering	Source code	static & dynamic	constraint satisfaction	Refactoring, distorted, and partial versions of the pattern can be identified. Prototype tool (PTIDEJ) that can transform the source code so that it complies with the detected pattern.
Tsanalis [14]	software re-engineering	Design	static & dynamic	similarity scoring between graphs coded as matrices	The main drawback of this approach is the convergence time which can be huge.
Dong <i>et al.</i> [5]	software re-engineering	Design	static & dynamic	template matching between graphs coded as matrices	This approach encodes eight design features, such as generalization, association and method invocations.
Bergenti and Poggi [2]	Design improvement	Design	static & dynamic	Template matching between graphs coded as matrices	Only exact matching of patterns can be identified. Support tool (IDEA) to improve UML designs (class and collaboration diagrams) based on a knowledge base of patterns.
Ka-Yee <i>et al.</i> [10]	Software re-engineering	Source code	dynamic	dynamic analysis & constraint programming	Dynamic analysis to reverse engineer the UML sequence diagrams from Java source code. An important challenge using dynamic analysis to trace the behavior of a system is the large amount of data involved and thus the execution time to solve the huge CSP.

In summary, none of the proposed approaches combines the structural and dynamic aspects in their pattern identification. Except for Ka-Yee [10], 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.

3. PATTERN IDENTIFICATION APPROACH

As mentioned in the introduction, in order to tolerate structural variations of a pattern, we adapt an XML document retrieval approach: we consider a design pattern as an XML query and the design as the target XML document where the pattern is searched. This adaptation is feasible since the transformation of UML diagrams into XML documents is straightforward and can be handled by all existing UML editors. Figure 1 illustrates an extract of the DTD commonly used to export a UML class diagram into an XML document.

```
<!ELEMENT classdef (name, inherit*, composition*,
association*,
dependency*,aggregation*, implements*, typedef*,
op*)>
<!ELEMENT inherit (type*, incompletetag?) >
<!ELEMENT composition (type*) >
<!ELEMENT aggregation(type*) >
<!ELEMENT dependency (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 1: DTD extract for the UML class diagram

In XML document retrieval, the document can be considered as an ordered, labelled 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 [12]:

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

where:

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

Note that the value of $C_R(c_q, c_d)$ is 1 if the two paths are identical. On the other hand, the more nodes separate the paths, the less similar they are considered, *i.e.*, the smaller their context resemblance value will be.

In the remainder of this section, we show how we adapted [refXXX] the above context resemblance function (1) to identify both, the static (structure and method declaration) and dynamic aspects of pattern in a design. Afterwards, in section 4, we illustrate how the results of similarity measures can be used to propose design improvements.

3.1 Static design pattern identification

The static pattern identification is composed of two parts: structure identification followed by method declaration identification. The first step relies entirely on an XML retrieval technique to identify the class and relationship part of the pattern. The second step relies on linguistic and typing information to identify the methods of the pattern. It confirms the first step's results and resolves any non deterministic identification when necessary.

Pattern structure identification

In XML document retrieval in general, the context resemblance function $C_R(1)$ 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 in the design are application domain dependent, while those in the design are generic. 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 C_R ; and 2) the importance of each relation in the pattern.

The structural resemblance between a pattern and a design starts by calculating the resemblance between each path of the pattern to all the paths in the design. In this calculation, 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.

Note that each tree in the XML documents representing a class diagram according to the DTD of Figure 1 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. Thus, in our resemblance calculation, we are examining one type of relationship among classes at a time.

To determine the structural resemblance between a pattern Q and a design fragment 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 design D
    - 3.1.1 If P_d and P_q have different types of relations
    - 3.1.2 then $CR(P_q, P_d) := 0$
    - 3.1.3 else //compare P_q with all sub-paths in P_d starting from different nodes
      - 3.1.3.1 For $s=1$ to $|P_d|-1$ // tolerate at most w additional nodes
        - 3.1.3.1.1 For $w=1$ to $\min(L+N, |P_d|-1)$
          - 3.1.3.1.1.1 $P'_d := P_d[s..s+w]$
          - 3.1.3.1.1.2 Compute $C_R(P_q, P'_d)$
4. Compute the *weighed* sum of all C_R scores for all the paths and store them in $CRMMatrix(Q,D)$;
5. Normalize $CRMMatrix(Q,D)$ by dividing each entry by the number of classes in D

In the 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 collect in $CRMMatrix$ the resemblance scores (*i.e.*, correspondences) between the classes of the design and the classes of the pattern; this matrix sums up the values of the context resemblance scores for each class in the design with respect to a class in the pattern. This weighted sum accounts for the importance of the relations in the pattern; for instance, in the Composite pattern, the aggregation relation is more important than the inheritance relation.

Finally, in step 5, these scores are normalized with respect to the total number of classes in the design; the final matching results are collected in $NormalizedCRMMatrix$ 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 $NormalizedCRMMatrix$. Note that there might be more than one such class. This non-deterministic correspondence could be resolved through the method correspondence step.

On the other hand, given two designs $D1$ and $D2$, 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 $D1$ and $D2$; the design with the maximum sum is the one that better instantiates the pattern.

Note that in a worst case 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.

Pattern method identification

Once the static classes and relations of the pattern are identified within the design, the pattern identification continues with the identification of pattern methods within the design. This identification should examine both the method name and signature. However, note that, it is not possible to compare the method signatures since in the design, the methods are adapted to the application or domain; thus their parameters are different from those of the pattern.

For method names, the resemblance is based on a linguistic/semantic similarity determined through either a dictionary (e.g., Wordnet [17]), or a domain ontology when available. A method m is said to resemble another method m' , if the name of m is either a synonym or homonym of the name of m' .

To determine the correspondences among the design and pattern methods, we use a normalized matrix (called *NormMethodDecMatrix*) that for each design class (row), it gives the percentage of resembling methods it has with each pattern class (column).

Global static pattern identification

So far, we managed to derive resemblance information about the static features of the design pattern: the classes and their relationships on one hand, and their methods on another hand. To determine the overall static resemblance, we combine both types of information. The combination can be either a simple addition of the two normalized matrices, or a weighted sum to reflect the importance of the two types of collected information: classes and their relationships vs. classes and their method declarations. The combined information should reinforce the quality of the identification results.

3.2. Dynamic pattern identification

While the static information can be sufficient for structural design patterns, it may be insufficient for behavioral and creational patterns. In fact, for these latter, the ordered method invocations are as essential as the static (structural) information.

To determine the behavioral resemblance between a design D and a pattern P , we rely on their sequence diagrams. In addition, to compare two sequence diagrams, we will compare the ordered message exchanges for each pair of objects that were already identified as similar during the static identification phase. For each object O in a sequence diagram, its ordered message exchanges are represented through an XML path (cf., Figure 8). Each node of these paths represents the type of the message (sent/received) along with the message being exchanged; this information allows us to derive a path where the edges have the same meaning: temporal precedence.

To compute the resemblance function scores between message paths, we slightly modify the C_R function defined in (1) to tolerate as opposed to penalize the additional, intermediate nodes. In fact, for message exchanges (represented as nodes), the important factor is the presence of particular messages (those of the pattern) in a given order; that is, additional message exchanges to those of the pattern will not affect the behavior of the pattern instantiation. Thus, the new function to compare message paths is as follows:

$$CRM(m_q, m_d) = \begin{cases} 1 & \text{if } m_q \text{ matches } m_d \\ 0 & \text{if } m_q \text{ does not match } m_d \end{cases} \quad (2)$$

where: m_q is an XML path representing message exchanges of an object O_p in the pattern, and m_d is an XML path representing message exchanges of an object O_d in the design.

Let O_p be an instance of a class C_p in the pattern P ; O_d be an instance of a class C_d in a given design; and suppose that C_d was identified as resembling C_p in the first step. Then, we have O_p and O_d have a resembling behavior if and only if the sum of the CRM of the sub-paths in the XML message paths of O_p and O_d is at least equal to the number of sub-paths in the XML message paths of O_p . When this constraint is not satisfied, then O_d either lacks messages exchanged by O_p , or it does not respect the order of message exchanges. In this case, one advantage of our method is that the designer will receive feedback about the mismatch cause at the behavioral level of each object.

4. DESIGN IMPROVEMENT PROPOSITIONS

Our pattern identification approach accepts the instantiation of a pattern P in a design D in the following cases:

- D preserves the core classes and relations of P ;
- D adds a “tolerated” number of classes between the classes of P ;
- the methods of the classes in P have their (semantically) matching methods in the corresponding classes of D ; and
- D preserves the ordered method invocations of P while possibly adding intermediate method invocations (in a tolerated number).

If any of the above cases of a pattern identification does not hold, our approach can detect the reason(s) behind it. More specifically, during the structural identification step, if D misses one relation of the pattern P , then all the scores $C_R(P_q, P'_d)$ computed during step 3.1.3 of our algorithm (see Section XXX) will be equal to zero. (Recall that each path represents one type of relation between the classes.) In this case, the missed relation can be indicated to the designer to correct his/her design. The designer is assisted since he/she is given the type of the relation to add and the classes (in terms of the application) where the relation must be added.

In addition, if D misses one class of P , then the normalized matrix *NormalizedCRMMatrix* computed in step 5 of our algorithm will have one column filled with zeros; this column

corresponds to the unmapped class. The designer can use the missing class name and context within the pattern to adjust his/her design.

Furthermore, if D adds too many classes between any two related classes in the pattern P , then the essence of the pattern might be lost. In this case, our identification approach will not detect the pattern. The designer can increase the number of tolerated intermediate classes (the parameter w in the identification algorithm) and try the identification again. Once the structural identification succeeds for a particular w (which could be too large), the designer can exploit the the class correspondences to restructure his/her design: he/she can eliminate intermediate classes, for instance by regrouping them, in order to have a design that is more faithful to the essence of the pattern.

The final part of the static pattern identification deals with the method declarations. In this case, our approach examines the matched classes pair-wise; any unmatched pattern method is automatically detected and can be signalled to the designer to intervene. He/she is assisted by the context where the missing method is defined to adjust his/her design.

As for the dynamic aspect of pattern identification, similar to class identification, any missed method invocation can be detected through all context resemblance scores being zeros.

5. EXAMPLES

In this section, we first use a simple example to illustrate how our structural identification approach can be used both to identify and propose design improvements. Then, we use a real design example to illustrate our approach: a fragment of the JHotDraw framework for graphical editors [16].

5.1. Illustration of structural pattern identification

To illustrate the steps of pattern resemblance determination using the structural information, let us consider the two design fragments illustrated in Figure 2.a and 2.b. To determine which one of the two design fragments (1 and 2) is the most similar to the illustrated pattern (Figure 2.c), we first convert them into XML documents; these latter are illustrated in a graphical format in Figure 3 and 4.

FIGURE 2: Sample design fragments to compare to a pattern

FIGURE 3: XML document trees for fragment 1 and the pattern

The context resemblance scores of the paths of the design fragment 1 with the pattern paths (Figure 3) are: $CR(cq1, cd1) = 1$ if the node A matches 1 and B matches 2 for the path $q1$, and $CR(cq2, cd1) = 0$ for the path $q2$ where $cq1$, $cq2$ and $cd1$ are the relevant paths from the root to a leaf.

Recall that some concepts are more essential in a pattern than others. In this pattern example, let us consider that the aggregation relation is twice as important as the inheritance relation. Thus, when collecting the CR score in the resemblance matrix, the score of the aggregation match is multiplied by two.

Based on these context resemblance scores, we obtain the following similarity matrix between the design fragment 1 and the pattern:

$$CRMatrix(\text{pattern}, \text{Fragment1}) \begin{matrix} \mathbf{A} \\ \mathbf{B} \end{matrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The normalized matrix is obtained by dividing the above similarity matrix by two (the number of classes in the design):

$$NormalizedCRMatrix(\text{Pattern}, \text{Fragment1}) \begin{matrix} \mathbf{A} \\ \mathbf{B} \end{matrix} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

On the other hand, the context resemblance function scores corresponding to the design fragment 2, illustrated in Figure 4, are shown in Table 2.

FIGURE 4: XML trees for the design fragment 2 and the pattern

TABLE 2: The context similarity function scores

	inherits 1 → 2	aggregates 1 → 2
inherits A → B	$CR(c_{q1}, c_{d1}) = 1$ if A=1 and B=2	0
inherits B → C	$CR(c_{q1}, c_{d1}) = 1$ if B=1 and C=2	0
inherits inherits A → B → C	$CR(c_{q1}, c_{d1}) = 0.75$ if A=1 and C=2	0
aggregates A → C	0	$CR(c_{q2}, c_{d2}) = 1$ if A=1 and C=2

Once the above context resemblance scores are computed, we compose the normalized similarity matrix which sums up the values of the context resemblance scores for each class in the design with respect to a class in the pattern and divides it by the number of classes in the design. In this example, the aggregation relation is, also, twice as important as the inheritance relation. Thus, when collecting the CR score in the resemblance matrix, the score of the aggregation match is multiplied by two. The resulting normalized CR matrix is therefore as follows:

$$\text{NormalizedCRMMatrix}(\text{pattern}, \text{Fragment2}) = \frac{A \begin{bmatrix} 3.75 & 0 \\ 1 & 1 \\ 0 & 3.75 \end{bmatrix}}{3} = \frac{A \begin{bmatrix} 1.25 & 0 \\ 0.33 & 0.33 \\ 0 & 1.25 \end{bmatrix}}{3}$$

To decide upon which fragment resembles more the pattern, we compute the sum of the maximum normalized CR for the nodes of the pattern. We get for fragment 1, the value (0.5+0.5) which is less than (1.25+1.25) for fragment 2; hence fragment 2 resembles more the pattern.

Fragment 2 was identified as resembling the pattern and each class in the design was identified as playing a pattern role. In this case, the designer is assisted by examining the pattern in terms of his/her application names. On the other hand, for fragment 1 which was not identified as a pattern, the designer receives a justification for the mismatch: the aggregation relation is missing in the design. Such a justification is important to restructure the original design and improve its quality.

5.2. Pattern identification in a fragment of the JHotDraw framework

To illustrate the steps of our approach, let us consider a fragment of the JHotDraw framework for graphical drawing editors [16]. This mature software architecture was used to derive several applications and it reuses several design patterns combined. Due to its complexity (about 250 classes and 200 relations), it is essential to have certain guides in identifying a particular design pattern, its relationships to others, and its variability points.

Figure 5 shows the JHotDraw design fragment we will analyze; note that, for an easier representation and comprehension of the patterns involved in the design, this class diagram shows the roles played by each class in ellipses. Figure 9 illustrates one of JHotDraw's sequence diagrams. Figures 6 and 10 show the class diagram and one sequence diagram for the Observer pattern which we will identify in the JHotDraw fragment.

Pattern structure identification

This first step computes the resemblance function scores. Table 3 shows a sample of these scores for the JHotDraw design and the Observer pattern paths of Figures 7 and 8.

FIGURE 5: A design fragment from the Jhotdraw framework

FIGURE 6: The observer design pattern [8]

FIGURE 7: An XML tree for the observer design pattern

FIGURE 8: XML trees for the Jhotdraw design fragment

FIGURE 9: A sequence diagram for a scenario of Jhotdraw

FIGURE 10: A sequence diagram for the Observer pattern

TABLE 3: Context resemblance scores

	$\xrightarrow{\text{inherits}}$ <i>Subject</i> $\rightarrow$ <i>ConcretS</i>	$\xrightarrow{\text{depends}}$ <i>Subject</i> $\rightarrow$ <i>Observer</i>	$\xrightarrow{\text{inherits}}$ <i>Observer</i> $\rightarrow$ <i>ConcretO</i>	$\xrightarrow{\text{depends}}$ <i>ConcretO</i> $\rightarrow$ <i>ConcreteS</i>
$\xrightarrow{\text{inherits}}$ <i>Figure</i> $\rightarrow$ <i>AbstractFigure</i>	$CR(c_q, c_d) = 1$ if <i>Figure</i> = <i>subject</i> <i>AbstractFigure</i> = <i>ConcretS</i>	0	$CR(c_q, c_d) = 1$ if <i>Figure</i> = <i>Observer</i> <i>AbstractFigure</i> = <i>ConcretO</i>	0
$\xrightarrow{\text{inherits}}$ <i>AbstractFigure</i> $\rightarrow$ <i>AttributeFigure</i>	$CR(c_q, c_d) = 1$ if <i>AbstractFigure</i> = <i>Subject</i> <i>AttributeFigure</i> = <i>ConcretS</i>	0	$CR(c_q, c_d) = 1$ if <i>AbstractFigure</i> = <i>Observer</i> <i>AttributeFigure</i> = <i>ConcretO</i>	0
$\xrightarrow{\text{depends}}$ <i>Figure</i> $\rightarrow$ <i>FigureChangeListener</i>	0	$CR(c_q, c_d) = 2$ if <i>Figure</i> = <i>Subject</i> <i>FigureChangeListener</i> = <i>Observer</i>	0	$CR(c_q, c_d) = 2$ if <i>Figure</i> = <i>ConcretO</i> <i>FigureChangeListener</i> = <i>ConcreteS</i>
$\xrightarrow{\text{depends}}$ <i>StdDrawing</i> $\rightarrow$ <i>AbstractFigure</i>	0	$CR(c_q, c_d) = 2$ if <i>StandardDrwing</i> = <i>Subject</i> <i>AbstractFigure</i> = <i>Observer</i>	0	$CR(c_q, c_d) = 2$ if <i>StandardDrwing</i> = <i>ConcretO</i> <i>AbstractFigure</i> = <i>ConcreteS</i>

Once these context resemblance scores are computed, we compose the normalized similarity matrix which sums up the values of the context resemblance scores for each class in the design with respect to a class in the pattern and divide it by the number of classes in the design. In this example, the dependency relation is, also, twice as important as the inheritance relation. Thus, when collecting the CR score in the resemblance matrix, the score of the dependency match is multiplied by two. The resulting normalized CR matrix is as follows:

$$\begin{aligned}
 \text{NormalizedCRMMatrix}(\text{Design}, \text{pattern}) &= \begin{matrix} & \begin{matrix} \text{Subject} & \text{Observer} & \text{ConcreteS} & \text{ConcreteO} \end{matrix} \\ \begin{matrix} \text{Figure} \\ \text{AbstractFigure} \\ \text{AttributeFigure} \\ \text{CompositeFigure} \\ \text{FigureChangeListener} \\ \text{StandardDrawing} \end{matrix} & \begin{bmatrix} 4.5 & 2.5 & 0 & 2 \\ 2 & 4 & 3 & 1 \\ 0 & 0 & 1.75 & 1.75 \\ 0 & 0 & 1.75 & 1.75 \\ 1 & 3 & 2 & 0 \\ 2 & 0 & 1 & 3 \end{bmatrix} \end{matrix} \\
 &= \begin{matrix} & \begin{matrix} \text{Subject} & \text{Observer} & \text{ConcreteS} & \text{ConcreteO} \end{matrix} \\ \begin{matrix} \text{Figure} \\ \text{AbstractFigure} \\ \text{AttributeFigure} \\ \text{CompositeFigure} \\ \text{FigureChangeListener} \\ \text{StandardDrawing} \end{matrix} & \begin{bmatrix} 0.75 & 0.41 & 0 & 0.33 \\ 0.33 & 0.66 & 0.5 & 0.16 \\ 0 & 0 & 0.29 & 0.29 \\ 0 & 0 & 0.29 & 0.29 \\ 0.16 & 0.5 & 0.33 & 0 \\ 0.33 & 0 & 0.16 & 0.5 \end{bmatrix} \end{matrix} \quad /6
 \end{aligned}$$

This normalized CR matrix identifies the Observer design pattern and indicates that the class Figure matches Subject, the class StandardDrawing is ConcreteObserver. However, note that the class AbstractFigure has been identified as Observer and ConcreteSubject at the same time and note that the resemblance score of the class FigureChangeListener with the Observer class is equal to (3) while the resemblance score of AbstractFigure with Observer is (4); thus, in this case, the method declaration aspect will let us decide upon which match is better taken and will give more accuracy to the results. Furthermore, the sum of the maximum normalized CR for the nodes of the pattern (2.41) is greater than the threshold which is equal to 4/6; thus this overall identification is acceptable.

To summarize, the structural aspect identifies the pattern, thus the designer may reconsider and restructure the design in order to be better conforming to the pattern.

Pattern method identification

Once a mapping between the classes of the design and those of the pattern is obtained, we continue with the verification of the method declarations. For this, we compute the *MethodDecMatrix* whose columns are the classes of the pattern and whose rows are the classes of the design.

Each entry in the matrix contains the percentage of methods of the pattern class that have been identified as similar to methods in a class belonging to design. The similarity between methods is based on name and signature resemblance. For example Changed() and Notify() methods are similar. In addition, in the JHotDraw design fragment, the class AbstractFigure contains the methods GetAttributes() and SetAttributes() which have been identified as similar to the methods GetState() and SetState() of the ConcreteSubject. The class Figure contains the method Changed() which is similar to the notify() method of the Subject and the class StandardDrawing contains the method FigureChanged() which is similar to the Update() method of the ConcreteObserver. The resulting normalized Method matrix is therefore as follows:

$$NormMethodDeclMatrix = \begin{matrix} & \begin{matrix} \text{Subject} & \text{Observer} & \text{ConcreteS} & \text{ConcreteO} \end{matrix} \\ \begin{matrix} \text{Figure} \\ \text{AbstractFigure} \\ \text{AttributeFigure} \\ \text{CompositeFigure} \\ \text{FigureChangeListener} \\ \text{StandardDrawing} \end{matrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix} /6$$

After summing up the normalized method matrix and the normalized CRMatrix, we obtain the following Matrix:

	Subject	Observer	ConcreteS	ConcreteO
Figure	0.91	0.41	0	0.33
AbstractFigure	0.33	0.66	0.83	0.16
AttributeFigure	0	0	0.29	0.29
CompositeFigure	0	0	0.29	0.29
FigureChangeListener	0.16	0.66	0.33	0
StandardDrawing	0.33	0	0.16	0.66

The use of the CRMatrix combined with the MethodDecMatrix eliminates the confusion and allows a precise and better identification of the pattern. As a result, the match score of the class AbstractFigure to Observer is equal to the match score of FigureChangeListener to Observer (0.66); however, since AbstractFigure has been identified as ConcreteSubject with a greater matching score (0.83), then FigureChangeListener is identified as Observer.

A second design improvement can be proposed based on the number of identified methods common between the design and the pattern. For example, the class Figure is identified as playing the role of the Observer and knowing that in the Observer class has two methods while only one found in common with Figure. Thus, we can suggest to the designer to improve the design by adding a method similar to the missing method.

Behavioral Resemblance determination

The final step of our identification approach exploits the behavioral information contained in the sequence diagrams. In the example of the Observer pattern identification, let us assume that the ConcreteSubject is the AbstractFigure and the ConcreteObserver is StandardDrawing. Then we determine the resemblance between the messages sent and received by the object AbstractFigure with the message sent and received by the ConcreteSubject and those of the object StandardDrawing with the ConcreteObserver.

The XML path corresponding to the sequence diagrams of the observer pattern is illustrated in Figure 11. Table 4 shows a sample of the resemblance function scores comparing the JHotDraw and the Observer sequence diagrams.

Note that, in Table 4, in order to optimize the resemblance score calculation, we treated the pattern XML path by considering two messages at a time; this covers the total order of message

exchanges for each object. On the other hand, for the design we need to consider XML paths with lengths 2 or longer; this allows us to tolerate additional, intermediate message exchanges.

For a design to cover all the message exchanges of the pattern, we need to find, in each column, at least one entry in this table that has the value of 1. Otherwise, the column that is missing a one indicates that the corresponding message exchange is missing in the design. This omission can be signaled to the designer in order to update the design. On the other hand, our method tolerates additional, intermediate message exchanges in the design. In our example, we tolerated at most one additional message; it is represented through the node representing the message Draw.

At the end of this dynamic aspect identification step, the designer has more confidence in the results of the structural and method identification steps.

FIGURE 11:. XML path for the sequence diagrams of the Observer pattern and JHotDraw

TABLE 4: Context resemblance scores for methods (C_{RM})

C_{RM}	$Recv: SetState()$ precedes → $Send: Notify$	$Send: Notify$ precedes → $Recv: Notify$	$Recv: Notify$ precedes → $Send: Update$	$Send: Update$ → precedes $Recv: GetState$
$Recv: SetAttribut$ precedes → $Recv : Draw$	0	0	0	0
$Recv: SetAttribut$ precedes → $Recv : Draw$ precedes → $send: changed$	1	0	0	0
$Recv: Draw$ precedes → $send : changed$	0	0	0	0
$Recv: Draw$ precedes → $send : changed$ precedes → $Recv: Changed$	0	0	0	0
$Changed$ precedes → $Changed$	0	1	0	0
$Changed$ precedes → $FigureChanged$	0	0	1	0
$FigureChanged$ precedes → $GetAttribute$	0	0	0	1

6. CONCLUSION

This paper illustrated how our proposed approach for pattern identification can improve a design. Our pattern identification considers a design pattern as an XML query to be found in an XML document representing the design. It uses an adapted context similarity function [12] to determine the most probable correspondences between the classes of the design and those in the pattern. It has the advantage of tolerating certain structural differences in the design compared to the pattern; the designer can fix a threshold below which the differences are un-

tolerated. A second advantage of our approach is that it can be applied for both structural and behavioural correspondences.

The pattern identification produces explanations in the case of unfaithful pattern instantiations: when pattern classes, relations and/or methods are missing, when the pattern classes are instantiated in a scattered way, and when the pattern method invocations are not respected in their order and number. These explanations can be exploited by a designer to rectify his/her original design and thus produce a better quality design.

Our future works include two axes. In the first, we are examining how to add more intelligence in our assistance for the recognition of pattern problems inside a design. This will be conducted by alleviating the search task by adding priorities in the computation of resemblance scores. In the second axis, 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, , Cointe P., Guéhéneuc Y. G., "Un meta-modele pour coupler application et detection des design patterns", L'objet N°8, 2002, pp1-18.
- [2] F. Bergenti, Poggi A., "Improving UML design pattern detection", Proceedings of the 12th international conference on software engineering and knowledge *engineering*, SEKE, 2000.
- [3] 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 Rev., vol. 46, no. 4, 2004, pp. 647-666.
- [4] N. Bouassida, H. Ben-Abdallah, "pattern detection through an information retrieval approach", International Conference on Information and Communication Systems, ICICS, December, 2009.
- [5] J. Dong, Y. Sun, Y. Zhao, "Design pattern detection by template matching", SAC'08, March 2008, Ceara, Brazil.
- [6] G. El Boussaidi, H. Mili, "Detecting patterns of poor design solutions by using constraint propagation", MODELS, proceedings of the 11th international conference on model driven engineering languages and systems, 2008.
- [7] G. M. Florijn, P. Meijers, Van Winsen, "Tool support for object oriented patterns", Proceedings of the European conference on object oriented programming, ECOOP, 1997.
- [8] E. Gamma, R. Helm, R. Johnson and J. Vlissides, "Design patterns: Elements of reusable Object Oriented Software", Addison-Wesley, Reading, MA, 1995.
- [9] Y. G. Gueheneuc, G. Antoniol, "DeMIMA: A Multilayered Approach for Design Pattern Identification", IEEE Transactions on Software Engineering, VOL. 34, NO. 5, 2008.
- [10] J. Ka-Yee Ng, 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.
- [11] H. Lee., Youn H., Lee E.. A design pattern detection technique that aids reverse engineering. "the International Journal of security and applications", Vol 2, N°1, January 2008.
- [12] C. D. Manning, P. Raghavan, H. Schütze. "An introduction to information retrieval", Cambridge University Press, England, 2008.
- [13] B. U. Pagel, M. Winter, "Towards pattern-based tools". Proceedings of EuropLop, 1996.
- [14] N. Tsantalis, Chatzigeorgiou A., Stephanides G. Halkidis S.T. "Design pattern detection using similarity scoring". IEEE transactions on software engineering, vol 32, N°11, san francisco,USA, 2006.
- [15] XML Metadata Interchange: OMG Document ad/98-07-03, July 6, 1998.
- [16] Gamma E., T. Eggenschwiler (2005), <http://www.jhotdraw.org>
- [17] WordNet, <http://wordnet.princeton.edu/>