A TOOL FOR DESIGN PATTERN DETECTION

Imen Issaoui**, Nadia Bouassida*, Hanene Ben-Abdallah**
Mir@cl Laboratory,
*Institut Supérieur d’Informatique et de Multimédia
**Faculté des Sciences Economiques et de Gestion
Sfax University, Tunisia
imene.issaoui@gmail.com, Nadia.Bouassida@isimsf.rnu.tn, Hanene.BenAbdallah@fsegs.rnu.tn

ABSTRACT

The advantages of design patterns, as good quality generic solutions for reoccurring problems, are irrefutable. They can be gained either at the design or the maintenance phases of a system. They motivated the proposition of several methods and tools for design pattern detection. The presented tool is distinguished by its detection method which identifies design patterns through an XML document retrieval approach. This identification method tolerates structural differences between the examined design and the identified pattern. In addition, it accounts for the important concepts representing the essence of a design pattern. Furthermore, to ensure the applicability of the method on large scale designs, the presented tool augments it with a decomposition method that applies a set of heuristics to divide the design into manageable segments.

Keywords: Automatic design pattern detection, XML document retrieval

1. INTRODUCTION

The advantages of design patterns [7], as good quality generic solutions for reoccurring problems, are irrefutable. They can be gained either at the design or the maintenance phases of a system. In fact, reusing design patterns during the design phase accelerates the development process; in addition, it ensures the derivation of good quality designs and eventually influences the quality of the produced code.

On the other hand, detecting design patterns in a re-engineered design can give hints on the comprehension of the software and on what kind of problems have been addressed during the development of the software. In fact, the presence or design patterns can be considered as an indicator of the good design quality of the software. In addition, they are very important during the re-documentation process.

Motivated by these advantages, different tools for design pattern detection have been proposed in the literature [2] [4] [5] [8] [11]. However, the so far proposed tools have problems in detecting all the design patterns of the GoF [7]. In fact, the tools presented in [1] [8] and [9] recognize only a small subset of these patterns and the found results contain many false positive design pattern instances. Moreover, they do not scale-up well when trying to analyze medium/large systems.

The aim of this paper is to describe a new design pattern detection approach and its associated tool. Our approach [3], for design pattern detection, has the originality of reusing an XML document retrieval technique 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 [10] 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 the structure in the pattern. A second advantage is that it accommodates design variability with respect to the pattern structure without losing the pattern essence. A third important advantage of our detection method is its genericity: It can be applied to any list of design patterns.

Compared to our previous work [3], this paper has a two-fold motivation. First, it augments our detection method by a new design decomposition method; this latter relies on a set of heuristics in order to find a way around the combinatorial nature of the detection problem. In addition, it allows the detection of multiple instances of a given pattern in one or more sub designs. The second motivation behind this paper is to present a tool for our detection method.

The remainder of the paper is organized as follows: Section 2 overviews current and existing works and tools for design pattern detection. Section 3 presents our approach for pattern identification. Section 4 describes the overall architecture of the tool and illustrates its functionalities through the detection of the Prototype design pattern in a fragment of the JHotDraw framework for graphical drawing editors [6]. Section 5 summarizes the paper and outlines our future work.

2. RELATED WORK

For reverse engineering purposes, several proposals looked in automating the identification of design patterns.

Tsantalis [11] proposes a design pattern detection methodology based on similarity scoring between graphs coded as matrices. Besides the structural aspect of patterns, this work examines methods through a graph representation; two methods are considered similar either when they have the same signature, or
when explicitly stating the base class method (e.g., via the super identifier in Java). This type of method invocation is one particular aspect of behavioral information in design patterns; another aspect, important for behavioral and creation patterns, is the temporal information about method invocation, which is not treated by this approach. In addition, the main drawback of similarity scoring-based approaches is their convergence time which depends on the graph size of the design.

Gueheneuc[8] propose an approach to semi-automatically identify the microarchitectures that are similar to design motifs in source code and to ensure the traceability of these microarchitectures between implementation and design. DeMIMA consists of three layers: two layers to recover an abstract model of the source code, including binary class relationships, and a third layer to identify design patterns in the abstract model. This approach focuses only on the structural aspect of the pattern, while neglecting the behavioral aspect.

Brown [4] uses dynamic information, analyzing the flow of messages. His approach is restricted to detecting design patterns in Smalltalk, since he only regards flows in VisualWorks for Smalltalk. He therefore annotates the Smalltalk runtime environment. Another drawback of this work is that it only gathers type information at periodic events.

Carriere et al. [5] employ code instrumentation to extract dynamic information to analyze and transform architectures. The presented approach only identifies communication primitives, but no complex protocols.

Bergenti and Poggi [2] propose an approach that allows only an exact match with the pattern. It proposes a tool, called IDEA, to improve UML designs (class and collaboration diagrams). It relies on a knowledge base where each pattern is described in terms of a structure template and a collaboration template (described internally using Prolog rules). For example, to detect the Composite pattern, the system searches all triplet classes having a template structure identical to the composite. When IDEA finds a pattern instance, a set of design rules are verified to test if the design could be improved. Then, a set of critiques are proposed as possible design improvements. The critiques/proposed improvements are pattern specific and they require a high level of understanding of both the design and the pattern.

3. DESIGN PATTERN DETECTION APPROACH:
The method presented in this paper is meant to strongly motivate structural differences between the examined design and the identified pattern. In addition, it accounts for the important concepts representing the essence of a design pattern. Furthermore, it can be used to identify both the structure and the methods of the pattern.

In the following sub-sections, we present, first, the XML document retrieval technique used in our approach. Then, we present our pattern detection.

3.1 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., [10]. In this work, the authors adopt 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 [10]:

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

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 $C_{q}(c_q, c_d)$ 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.

3.2. 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 losing 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.
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 \( P_q \) in the document \( D \\
3.1.1 If \( P_d \) and \( P_q \) have different types of relations \( 3.1.2 \) then \( C_R(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 \( C_R(P_q, P'_d) \\
4. Compute the weighted sum of all \( C_R \) scores for all the paths and store them in \( C_R(\text{Matrix}(Q, D)) \\
5. Normalize \( C_R(\text{Matrix}(Q, D)) \) 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 sum up in \( C_R \) Matrix 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 Normalized \( C_R \) Matrix 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 Normalized \( C_R \) Matrix.

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.

4. TOOL SUPPORT

In this section, we first present the main functionalities of the tool. Secondly, we illustrate it through the Prototype pattern.
4.1. FUNCTIONAL ARCHITECTURE

An overview of the principal activities performed by our tool is depicted in Figure 2 which shows: the general process of XML document retrieval, design pattern detection and consequent results visualization.

The first step consists in either re-engineering the source code using the Rational Rose tool or drawing the new design. The second step first transforms the XMI file corresponding to the design into an XML file valid according to the DTD of Figure 1; secondly, it decomposes the entire design obtained into sub-designs. The third step is the application of the similarity algorithm for the sub-designs and a selected pattern; that is, this step seeks patterns in each sub-design separately. The last step extracts and identifies the pattern in each sub-design.

The decomposition method divides the input design into sub designs in order to improve the efficiency of the tool in practice. A sub-design is defined as a portion of the entire design consisting of classes belonging to one or more inheritance hierarchies. Decomposing a design into inheritance hierarchies creates groups of classes that can possibly be part of a design pattern instance. The inheritance relation is chosen as a decomposition criterion because the vast majority of GoF design patterns [7] employ polymorphism which is achieved by means of abstractions. Note, also, that since the association relationship exists, in general, only between two classes, then as a result, using associations as a decomposition criterion would result in a significantly larger number of class pairs: this is due to the fact that association relationships are definitely more common than inheritance relationships. Furthermore, the number of resulting class pairs will be significantly larger than the total number of classes in the design. On the other hand, decomposition based on inheritance hierarchies results in a number of class groups which is smaller than the total number of classes in the design. Obviously, such issues affect the performance of a design pattern detection.

4.2. IDENTIFICATION OF THE PROTOTYPE PATTERN IN JHOTDRAW:

To illustrate the functionalities of our tool, let us consider a fragment of the JHotDraw framework for graphical drawing editors [6] and let us identify the Prototype design pattern [7].

Note that, the Prototype design pattern (Figure 3) aims at specifying the kind of objects to create using a prototypical instance. It creates new objects by copying this prototype and it consists in a Prototype class that declares an interface for cloning itself, a ConcretePrototype class that implements an operation for cloning itself and one Client class that creates a new object by asking a prototype to clone itself. Figure 4 shows the JHotDraw design fragment we will analyze. Figure 6 and 7 show two sub designs. The design decomposition allows as to obtain an XML document corresponding to the sub-designs as shown in Figure 5. Then, the first step towards the detection of the design pattern exploits the structural/static information. This relies on the computation of resemblance function scores. Figure 8 shows a screen shot illustrating the resemblance function scores comparing the first sub design (shown in figure 6) and the Prototype pattern, of Figure 3.

Once the context resemblance scores are computed, the normalized similarity matrix is shown to the user. It 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.

The normalized CR matrix (shown in Figure 9) identifies the prototype design pattern and indicates that the class Figure matches the Prototype class, the class AbstractFigure is the ConcretePrototype and the class Client is identified as CreationTool.

Furthermore, the sum of the maximum normalized CR for the nodes of the pattern (1) is greater then the threshold which is equal to 4/5; thus this overall identification is acceptable.
Figure 3: The Prototype pattern

Figure 4: The JHotDraw design fragment
Figure 5: The decomposition

Figure 6: The first sub design
Figure 7: The second sub design

Figure 8: A screen shot showing the resemblance function scores
5. CONCLUSION

In this paper, we have presented a tool for design pattern detection which facilitates the understanding of a software system by detecting design patterns. We use an XML document retrieval technique. More specifically, we consider a design pattern as an XML query to be found in an XML document representing a design. We adapt a context similarity function [10] to determine the most probable correspondences between the classes of the design and those in the pattern. Our approach has the advantage of tolerating certain structural differences of 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 determining both the structure and the roles of the pattern elements. We are currently examining how to evaluate the proposed approach on open-source systems and compare the results of our experiments with other approaches.

REFERENCES


