Design Patterns Mining using Subgraph Isomorphism: Relational View

Manjari Gupta, Akshara Pande
DST- Centre for Interdisciplinary Mathematical Sciences,
Department of Computer Science,
Banaras Hindu University, Varanasi-221005, India.
manjari@bhu.ac.in, pandeakshara@gmail.com

Abstract

Design Pattern Detection is a part of many solutions to Software Engineering difficulties. The usage of design patterns leads to benefits for new and young developers by enabling them to reuse the knowledge of their experienced colleagues. Mining of design pattern instances is important for program understanding and software maintenance. Hence a reliable design pattern mining is required. Here we are using the relational view of subgraph isomorphism to detect design patterns in the source code.

Keywords: *design pattern, UML, subgraph isomorphism.*

1. Introduction

Design patterns [1] increasingly being applied in object oriented software design processes as a part of many solutions to Software Engineering difficulties and thus extensively used by software industries. Each Design pattern denotes a high level abstraction, and contains expert knowledge. It is a part of reengineering process and thus gives important information to the designer. A design pattern can be reused as a building block for better software implementation and their documentation in a software system can improve software maintenance and program understanding. With the help of these patterns specific design problem can be solved and object oriented design become more flexible and reusable. It is tough to trace out such design information. To understand the systems and to modifications in them it is necessary to recover pattern instances. There are number of pattern detection techniques, some of them have been discussed in section 4. In this paper we are using a relational view of subgraph isomorphism for design pattern detection. The advantage of this approach is that variants of each design pattern as well as any occurrence of a design pattern can be detected.

Here we are taking two graphs, one is corresponding to the input graph (i.e. system under study) and other is corresponding to the design pattern graph. Subgraph isomorphism is the problem of finding an isomorphic image of a given graph, called the pattern (design pattern in our case), in another graph, called the target (system under study in our case) [8]. Related works are discussed in section 2. In section 3 representation of relationship graphs are shown. Relational view of subgraph isomorphism and its application for design patterns detection are explained in section 4. Lastly we concluded in section 5.

2. Related Work

The first effort towards automatically detect design pattern was achieved by Brown [6]. In this work, Smalltalk code was reverse-engineered to facilitate the detection of four well-known patterns from the catalog by Gamma et al. [1]. Antoniol et al. [7] gave a technique to identify structural patterns in a system with the purpose to observe how useful a design pattern recovery tool could be in program understanding and maintenance. Nikolaos Tsantalis [2], proposed a methodology for design pattern detection using similarity scoring. But the limitation of similarity algorithm is that it only calculates the similarity between two vertices, not the similarity between two graphs. To solve the above limitation Jing Dong [3] gave another approach called template matching, which calculates the similarity between subgraphs of two graphs instead of vertices. S. Wenzel [4] gave the difference calculation method works on UML models. The advantage of difference calculation method on other design pattern detecting technique is that it detects the incomplete pattern instances also.

Our earlier work, we used klenberg approach and fuzzy graph algorithms for design pattern detection [9]. The drawback of these two methods is that they are only concerned about node similarity not the whole graph. We used sub graph isomorphism detection approach that overcomes this drawback [9]. We have used these and other approaches for design pattern detection in GIS application [10]. To reduce complexity of design pattern detecting algorithm we used the graph decomposition technique [11]. The order of complexity of this decomposition algorithm is O(n3), where n is the number of nodes present in the graph. This algorithm works for only those design patterns having similar relationships among at most three classes in its UML class diagram. However this condition may not hold for only few of the design patterns. Thus this approach can be applied for almost all of the design patterns. In another work we find out whether design pattern matches to any subgraph of system design by using decision tree [12]. A decision tree is developed with the help of row-column elements, and then it is traversed to identify patterns. By applying the decision tree approach, the complexity is reduced. We proposed a new approach 'DNIT' (Depth-Node-Input Table) [13]. It is based on the concept of depths from the randomly chosen initial node (also called root node which has depth zero) in directed graph. In another work we applied state space representation of graph matching algorithm to detect design patterns [14]. State space representation easily describes the graph matching process. The advantage of this method used for design pattern detection was that the memory requirement was quite lower than from other similar algorithms. Another advantage is that it detects variants as well as any occurrence of each design patterns. We proposed a design pattern detection technique using a greedy algorithm for multi-labeled directed graphs [15]. Using inexact graph matching design pattern detection is described in [16, 17].

3. Representation of System Design and Design Patterns

Before applying relational view of subgraph isomorphism, it is necessary to extract all the relationship graphs (which are directed graphs) from the UML diagrams of the system design (system under study) and design patterns. After extracting all the relationships, each relationship graph is represented by taking a node corresponding to a class involved in that relation and an edge corresponding to that relationship between two classes (nodes). Relationship graphs representation for system design is explained here.

The class diagram of the system under study or the system for which we have the source code is drawn in figure 1. After that the relationship graphs (that exists in UML diagram) is extracted. There are two relationships (i.e. generalization and direct association), the corresponding relationship graphs (i.e. directed graph) are shown in Figure 2 and 3 respectively.

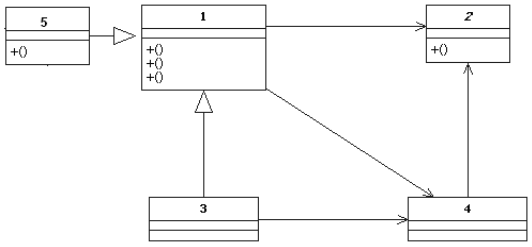

Fig. 1. UML Diagram of System Design [18]

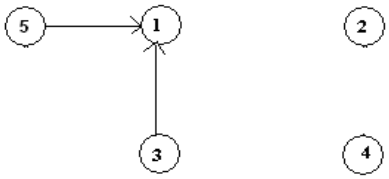

Fig.2. Generalization Relationship Graph of UML Diagram of System Design

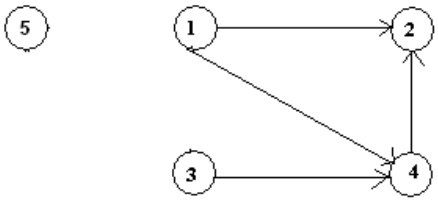

Fig.3. Direct Association Relationship Graph of UML Diagram of System Design

The relationship graphs for design patterns can be extracted in similar manner. For some of the design patterns it is shown in section 4.

4. Relational View of Subgraph Isomorphism

Subgraph isomorphism is an important and very general form of exact pattern matching. Theoretically, subgraph isomorphism is a common generalization of many important graph problems including finding Hamiltonian paths, cliques, matching, girth and shortest paths.

Subgraph isomorphism is the problem of finding an isomorphic image of a given graph, called the pattern, in another graph, called the target. The name of this problem comes from the image of the pattern being then a subgraph of the target [8]. Here, we take pattern and target as that of relationship graph of UML diagram of design pattern and system design respectively.

A relation Φ € V1 × V, is an isomorphism of a graph G1 = (V1, E1) to a subgraph of a graph G = (V, E) if the following conditions hold [8].

$$\phi^T \phi \subseteq I \quad (1)$$

$$\phi^T \phi = I \quad (2)$$

$$E = \phi E1 \phi^T \quad (3)$$

and it is also written as Φ: G1→G.

The conditions 1 and 2 assure that Φ is an injective function from V1 to V, the condition 3 guarantees that Φ preserves the structure of G1, that is, it maps all arcs of G1 to arcs of G. In other words, an injective function Φ: V1→ V is a subgraph isomorphism of G1 = (V1, E1) into G = (V, E) if it satisfies the following condition: (u1, v1) € E1 implies (Φ (u1), Φ (v1)) € E for all u1, v1€ V1.

The set of all subgraph isomorphisms of a graph G1 = (V1,E1) into a graph G = (V,E) can be represented by an n-ary relation I $\subseteq$ Vn, where $|V1| = n$. The subgraph isomorphism relation I is defined as follows:

$(v_1,.....v_n) \in I \Leftrightarrow \exists$ a subgraph isomorphism Φ: G1→ G such that $\phi(v_1^1) = v_1, \phi(v_2^1) = v_2 ..... \phi(v_n^1) = v_n, \quad v_i^1 \in G1, v_i \in G$ for all i = 1,2,…n.

Given two graphs G1 = (V1,E1) and G = (V,E) with $|V1| = n$, let $E_{ij} = V^n$ denote the n-ary relation on V containing exactly those (pairwise disjoint) vertices of V which are joined by some arc in G, that is,

$$E_{i,j} = \{(x_1, x_2, ...x_n) \in V^n | (x_i, x_j) \in E, x_r \neq x_s \text{ for all } r \neq s\}$$

The subgraph isomorphism relation I can be computed as follows [8].

$$I = \bigcap_{(v_i^1, v_j^1) \in E1} E_{i,j} \quad (4)$$

We will apply this procedure to detect whether a relationship graph of the design pattern exist in the same relationship graph of system design. The limitation of this process is that we cannot find the overall mapping of the design pattern into system design. This procedure detects sub isomorphism for each relationship separately. Thus, to find out whether a design pattern exists (complete/partially) or not we need to combine result of separate relationships exists in the design pattern. Here we are introducing a new algorithm (based on our algorithm proposed earlier [13]) that will identify complete sub isomorphism between design pattern graph and model graph (corresponding to system design) by combining the outputs (subgraph isomorphisms) of above procedure applied separately for all the relationship exists in a design pattern. Firstly, assign unique natural numbers i to each of the relationships of UML and let I_i be the corresponding isomorphism obtained by the above algorithm for the same relationship. We assign 1 to generalization relationship, 2 to direct association, 3 to aggregation relationship. In the system design and design patterns under study we are having only these three relationships. Similarly, other relationships can also be numbered.

```
PROCEDURE Complete_Detection
FOREACH set of Isomorphisms in I1
    Fi= ith isomorphism in I1                ( i= 1 to number of relationships)
    FOREACH isomorphism in Ii    ( i≠1)
```

I_{ij}= j^{th} isomorphism in I_i (j= 1 to number of isomorphism detected for i^{th} relationship)

IF any pair of F_i is equal to any pair of I_{ij} and there are corresponding edges between all the nodes

$F_i = F_i \cup I_{ij}$

END IF

END FOREACH

END FOREACH

END PROCEDURE

The above algorithm combines the isomorphisms detected for all relationship graphs of a design pattern to the corresponding relationship graphs of model graph, and gives the final isomorphism (F) of design pattern to model graph. If for any relationship, more than one instances exist in model graph corresponding I_i contains more than one elements showing detection of more than one subgraph isomorphism. In that case more than one instances of design pattern (F_i) or part of the design pattern may exist in the model graph. If F_i is empty, design pattern does not exist in the model graph. If F_i covers each node of the design pattern, design pattern exists in the model graph. Otherwise, design pattern partially exists in the model graph.

4.1. Design Pattern Detection as Façade Design Pattern (Full Match)

We are considering Façade design pattern, the UML diagram corresponding to it is shown in figure 4. There is only one relationship, i.e. direct association relationship. The corresponding graph is shown in Figure 5.

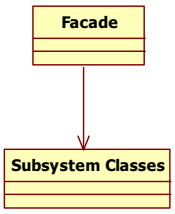

Fig.4. Façade Design Pattern [18]

Fig. 5. Direct Association Relationship Graph of UML Diagram of Façade Design Pattern

For all the relationships present in the façade design pattern, we will calculate the subgraph isomorphism I. There is only direct association relationship in this pattern.

Let us take graph corresponding to this relationship (figure 4) as G_1 and graph corresponding to same relationship in system design (figure 3) as G. Here in graph G1,

there is only one edge i.e. edge from node 1' to node 2'. Thus the n-ary relation ($E_{i,j}$) contains only one element $E_{1,2}$ = { (1, 2), (1, 4), (3, 4), (4, 2) }. Now by equation (4), I_2 = {(1,2),(1,4),(4,2),(3,4)}. Thus, I_{21} = {(1,1) (2,2)}, I_{22} = {(1,1), (2,4)}, I_{23} = {(1,4), (2,2)}, and I_{24} = {(1,3), (2,4)}.

Since there is no other relationship in this pattern, so we will apply the Complete_Detection algorithm only on I2. For each I_{2i} (i =1 to 4), we get one F_i, thus F_1 = {(1,1) (2,2)}, F_2 = {(1,1), (2,4)}, F_3 = {(1,4), (2,2)}, and F_4 = {(1,3), (2,4)}. Thus four instances of facade design pattern completely exist in the system design.

4.2. Design Pattern Detection as Composite Design Pattern (Partial Match)

Consider composite design pattern. Its UML diagram is shown in figure 5. The corresponding generalization, direct association and aggregation relationships have been shown in figure 7, 8 and 9 respectively.

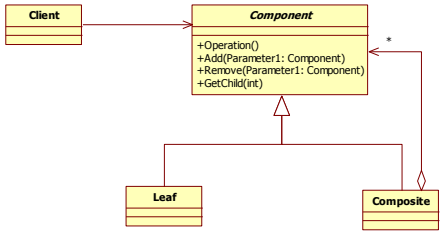

Fig. 6. Composite Design Pattern [18]

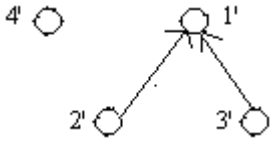

Fig. 7. Generalization Relationship Graph of UML Diagram of Composite Design Pattern

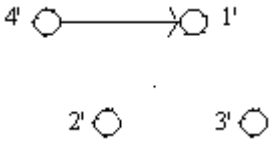

Fig. 8. Direct Association Relationship Graph of UML Diagram of Composite Design Pattern

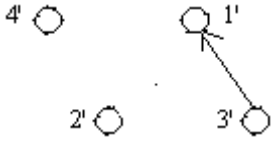

Fig. 9. Aggregation Relationship Graph of UML Diagram of Composite Design Pattern

Let us take first the generalization relationship graphs of design pattern (figure 7) and system design (figure 2) as G1 and G. There are two edges, one from 2' to 1' and another from 3' to 1'. Thus the n-ary relation ($E_{i,j}$) contains two element $E_{2,1}$ and $E_{3,1}$.

$E_{2,1}$ = { (1, 5, *), (1, 3, *)}, similarly, $E_{3,1}$ = { (1, *, 5), (1, *, 3)}. Now I_1 can be calculated by using equation 4, as follows:

$I_1 = E_{2,1} \cap E_{3,1}$ = { (1,5,3), (1,3,5)}

Thus I_{11}= {(1,1), (2,5), (3,3)} and I_{12}= {(1,1), (2,3), (3,5)}. This shows two instances of the generalization relationship graph of the composite design pattern exist in the model graph. Now we will consider direct association graph of the design pattern (figure 8) as G1 and the corresponding graph (figure 3,) of system design as G. Here in graph G1, there is only one edge i.e. edge from node 4' to node 1'. Thus the n-ary relation contains only one element $E_{4,1.}$ By the definition of E_{ij}, $E_{4,1}$= {(1, 2), (1, 4), (3, 4), (4, 2)}. Now by equation (4), I_2= {(1,2),(1,4),(4,2),(3,4)}. I_{21}= {(4,1), (1,2)}, I_{22}= {(4,1), (1,4)} , I_{23} = {(4,3), (1,4)}, I_{24}= {(4,4), (1,2)}. Thus there are four occurrences of the direct association relationship graph of composite design pattern in the corresponding graph of system design.

In composite design pattern there is aggregation relationship (corresponding graph is shown in figure 9) but it is not present in system design, thus $I_3 = \phi$. Since I_1 and I_2 are not empty we can find out the partial matching. We can say the variant of the composite design pattern is used in the system design. Now, we will apply Complete_Detection algorithm on I_1 and I_2. Since there is no common element in any pair of I_{1i}'s (i=1 to2) and I_{2j}'s (j=1 to 4), F_1= {(1,1), (2,5), (3,3)} and F_2= {(1,1), (2,3), (3,5)}. These mappings do not cover all the nodes of the composite design pattern and thus composite design pattern partially exist in the system design

4.3 Particular Design Pattern may or may not exist

Now, consider composite design pattern. There is single relationship (direct association) and a single node in this pattern.

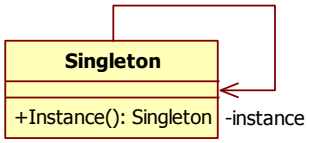

Fig.10. Façade Design Pattern [18]

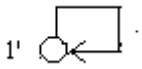

Fig. 11. Aggregation Relationship Graph of UML Diagram of Composite Design Pattern

Let us take direct association relationship graph of design pattern (figure 10) and of system design (figure 3) as G1 and G. Since there is no self loop we cannot calculate E_{ij} and thus, each Ii is empty in this case. It shows the singleton design pattern does not exist in the system design.

9. Conclusion

In this paper we took the relationships present in system design and design patterns, and tried to find out whether design pattern exists or not in system design. Firstly, we extract both. There are 23 GoF (Fang of Four) [1] design patterns. Then we calculated subgraph isomorphism relation I for all the relationships present in design pattern taking it as graph G1. On the basis of relationships presence and number of nodes covered by the subgraph isomorphism we categorize matching in three parts, fully match, partial match and no match. By using the relational view of subgraph isomorphism one can detect sub isomorphism for each relationship present in the design pattern separately. It is necessary to check the overall matching of the design pattern. Thus, we proposed an algorithm (based on algorithm we proposed earlier [14]) that combines the result of all the relationships and can identify this overall matching. If the result (subgraph isomorphism) of this algorithm covers all the nodes of the design pattern graph, we say that the design pattern exist in the model graph. If it is empty, design pattern does not exist in the system design. Otherwise, design pattern partially exists in the system design. If more than one subgraph isomorphisms exist, more than one instances of the design pattern or its part exist in system design.

References

[1] E. Gamma, R.Helm, R.Johnson, J.Vlissides, "Design Patterns Elements of Reusable Object-Oriented Software", Addison- Wesley, 1995.

[2] N.Tsantalis, A.Chatzigeorgiou, G.Stephanides, and S.Halkidis, "Design Pattern Detection Using Similarity Scoring", IEEE transaction on software engineering, 32(11) , 2006.

[3] J.Dong, Y.Sun, Y.Zhao, "Design Pattern Detection By Template Matching", the Proceedings of the 23rd Annual ACM, Symposium on Applied Computing (SAC),Ceará, Brazil,March, 2008, pp. 765-769.

[4] S.Wenzel, U. Kelter, "Model-driven design pattern detection using difference calculation" In *Proc*. of the 1st International Workshop on Pattern Detection For Reverse Engineering (DPD4RE), Benevento, Italy, 2006.

[5] B.T. Messmer, H. Bunke, "Subgraph isomorphism detection in polynomial time on preprocessed model graphs", Second Asian Conference on Computer Vision, pp. 151–155, 1995.

[6] K.Brown, "Design Reverse-Engineering and Automated Design Pattern Detection in Smalltalk", Technical Report TR-96-07, Dept. of Computer Science, North Carolina State Univ., 1996.

[7] G.Antoniol, G. Casazza, M. D. Penta, R.Fiutem, "Object- Oriented Design Patterns Recovery", J. Systems and Software, vol. 59, no. 2, 2001, pp. 181-196.

[8] J.Cortadella, and G.Valiente, "A relational view of subgraph isomorphism", In Proc. 5th Int. Seminar on Relational Methods in Computer Science, Qu´ebec, Canada, 2000, pp. 45–54.

[9] A.Pande, and M.Gupta, "Design Pattern Detection Using Graph Matching", International Journal of Comuter Engineering and Information Technology*(*IJCEIT), Vol 15, No 20, Special Edition 2010, pp 59-64.

[10]A.Pande, M. Gupta, and A.K.Tripathi, "Design Pattern Mining for GIS Application using Graph Matching Techniques", 3rd IEEE International Conference on Computer Science and Information Technology, 2010, Chengdu, China.

[11]A.Pande, M.Gupta, and A K Tripathi, "A New Approach for Detecting Design Patterns by Graph Decomposition and Graph Isomorphism", In Proc. Of Third International Conference on Contemporary Computing(IC3), published by Springer, 2010, Noida, India.

[12] A.Pande, M.Gupta, and A K Tripathi, "A Decision Tree Approach for Design Patterns Detection by Subgraph Isomorphism", In Proc. Of International Conference on Advances in Information and Communication Technologies (ICT 2010) published by Springer, 2010, Kochi, Kerela, India.

[13] A.Pande, M.Gupta, and A K Tripathi, "DNIT – A New Approach for Design Pattern Detection", In Proc. Of International Conference on Computer and Communication Technology, IEEExplore, 2010, MNNIT, Allahabad.

[14] M.Gupta., R.R.Singh, A. Pande, and A.K.Tripathi, "Design Pattern Mining Using State Space Representation of graph matching", CCSIT, published by Springer, 2011, Banglore, India.

[15] M.Gupta., "Design Pattern Mining Using Greedy Algorithm for Multilabeled Graphs", International Joint Conference on Information and Communication Technology, 2011, Bhubaneshwar, IPM Pvt. Ltd, India.

[16] M.Gupta., R.R.Singh, A.K.Tripathi, "Design Pattern Detection using Inexact Graph Matching", International Conference on Communication and Computational Intelligence, 2010, Tamil nadu.

[17] M.Gupta, "Inexact Graph Matching for Design Pattern Detection using Genetic Algorithm", International Conference on Computer Engineering and Technology, Nov 2010, Jodhpur.

[18] StarUML, The Open Source UML/MDA Platform. http://staruml.sourceforge.net/en/.

Acknowledgement

This work was partially supported by project "Design and Development of Next Generation e-Content for Software Reuse by Design Patterns and Frameworks" under the centrally sponsored Plan Scheme of National Mission on Education through Information & Communication technology (NMEICT) of M.H.R.D., GOI, New Delhi.

Authors

Dr. Manjari Gupta is serving as Assistant Professor with the Department of Computer Science of Banaras Hindu University, Varanasi, India has her MSc degree in Computer Science from J.K. Institute of Applied Physics and Technology, Allahabad University, Allahabad and the PhD in Computer Engineering from Banaras Hindu University, Varanasi, India. She is engaged in teaching and research for the last 8 years and the areas of his research interest include Software Reuse and Design pattern Detection.

Ms. Akshara Pande is Junior Research Fellow with the DST-Centre for Interdisciplinary Mathematical Sciences,of Banaras Hindu University, Varanasi, India. She has done MSc in Computer Science from J.K. Institute of Applied Physics and Technology, Allahabad University, Allahabad. Her research area is Design pattern Detection.