A Technique for Design Patterns Detection

Manjari Gupta

Department of computer science
Institute of Science
Banaras Hindu University
Varansi-221005, India
manjari_gupta@rediffmail.com

Abstract—**Several recurring patterns of classes exist in many object oriented software as an experience of developers. Design Pattern Detection is an important part of many solutions to Software Reuse practices. Design pattern instances are highly important and useful for program understanding and software maintenance. Hence an automatic and reliable design pattern mining is required. In this paper inexact graph matching approach is applied for design pattern detection using genetic algorithm.**

Keywords- design pattern, UML, subgraph isomorphism, genetic algorithm

I. 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 and thus a software developed using design patterns have many desired properties. Design pattern detection is a part of reengineering process and thus gives important information to the designer. To understand a software system and to modify it, it is necessary to recover pattern instances. It would be useful for reengineers to have an automatic design pattern detection tool that can detect design pattern from the system without need of thorough/manual analysis of it. There are number of pattern detection techniques, some of them have been discussed in section 5. In this paper an inexact graph matching approach is used for design pattern identification by applying genetic algorithm. We are implementing this approach to get a design pattern detection tool so that reengineers need not to manually analyze the design of the system to identify used design patterns, if any, to understand the design of the system.

Two graphs are taken, one is corresponding to the system design (i.e. system under study) and other is corresponding to the design pattern graph. A particular example is shown in figure 1. A large number of random mappings (population) are generated between the system design and the design pattern graph. Different mappings are corresponding to different subgraphs of the system design. Genetic algorithm is applied over these random mappings to obtain the best mapping based on their fitness function value. The advantage of this approach is that it reduces time complexity of matching two graphs. Using this approach we can also detect instances of design patterns that is not possible by many other approaches proposed in the literature. In section 2 graph representations of the system design and design patterns are explained. The proposed genetic algorithm for design pattern detection by inexact graph matching is described in section 3. Section 4 describes the issues in design pattern detection using inexact graph matching. Related works are discussed in section 5. The paper is concluded in section 6.

II. GRAPH REPRESENTATION OF SYSTEM DESIGN AND DESIGN PATTERNS

UML diagrams of system design and design patterns are converted into graphs. Before converting a UML diagram into graphs first the UML diagram is modified in such a way so that variant of design patterns can also be detected. The reason for the design pattern variant problems is the fact that the inheritance and aggregation relationship have the property of transitiveness [2]. Thus if there is an inheritance (or aggregation) relationship between classes c1 and c2 and the same relationship between c2 and c3, we will introduce the same relationship between c1 and c3 also.

Classes of UML diagram are represented as nodes and relationships among classes by edges. Each node and edge is labeled. The label of each node is 3-tuple (t_1, t_2, t_3) where t_1 is number of super classes, t_2 is number of sub classes, and t_3 is number of collaborators of this node (class). It can be modified to include more other attributes of a class. In this initial effort only three attributes of a class are considered. Each edge is corresponding to one of the relationships. Label 1 is assigned for dependency, 2 for generalization, 3 for direct association, and 4 for aggregation. For the system design represented by the UML Diagram shown in fig. 1, the corresponding graph (MG) is extracted and shown in fig 2.

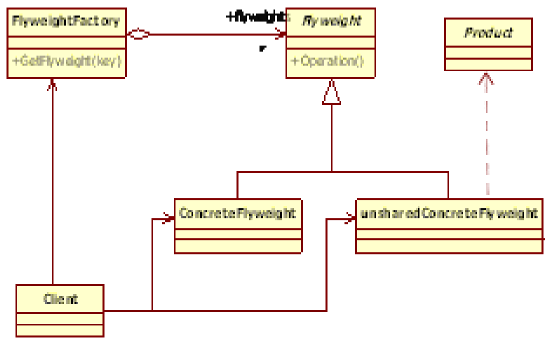

Fig. 1 UML Diagram of System Design

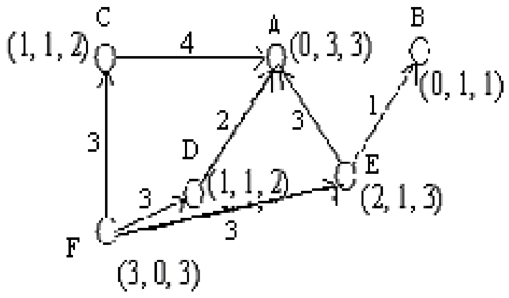

Fig. 2. Model graph (MG) corresponding to system design

The relationship graphs (DPG) for design patterns can be represented as a graph in similar manner as for system design. For example command design pattern and the corresponding graph are shown in fig3 and fig 4 respectively.

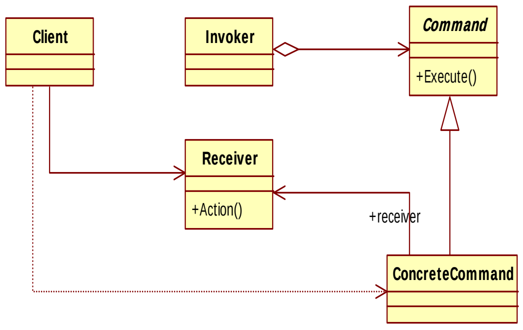

Fig. 3 UML Diagram of Command Design Pattern

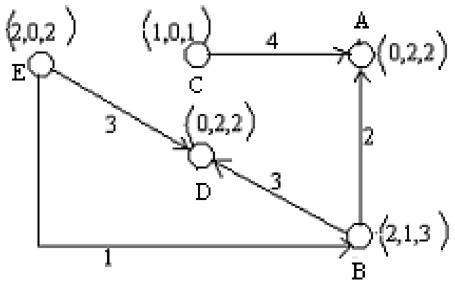

Fig. 4 Graph (DPG) for command design pattern

III. Proposed Genetic Algorithm for Design Pattern Detection by inexact graph matching

A chromosome represents one of the node to node mappings of design pattern to system design. The proposed chromosome structure C is n x m matrix where n is the no. of nodes in design pattern graph and m is no. of nodes in system design graph. Each row and each column of C contain at most one '1' and rest of the elements are '0'. A chromosome is legal only if it represents a bijective mapping. In this the main objective is to find out chromosomes that represent sub-graph isomorphism between design pattern and system design, if any. This objective is incorporated into the fitness function of the GA. The fitness function measures the performance of the chromosomes i.e. how much the mapping represented by it is close to the sub graph isomorphism between the design pattern and the system design graph. Fitness function (F) is defined as followed.

$F = F_{nc} + F_{ec}$, where F_{nc} is the cost of node mapping and F_{ec} is the cost of edge mapping. F_{nc} and F_{ec} are defined as follows:

The matching error generated by nodes (F_{nc}) that is dissimilarity between matched nodes i.e. $|MG_{t1}-DPG_{t1}|+|MG_{t2}-DPG_{t2}|+|MG_{t3}-DPG_{t3}|$, where MG_{ti} is the ith component of the label of node in model graph and DPG_{ti} is the ith component of the label of matched node in the design pattern graph.

The error generated by the corresponding edges (F_{ec}) is calculated as follows. The difference between labels of matched edges is calculated. If it is 0 then F_{ec} is 0 since both edges are corresponding to same relationship, otherwise F_{ec} is set to q where q is a very large positive number to show that edges are not corresponding to the same relationship.

After generating the feasible chromosomes, using this objective function, one chromosome may be compared with another chromosome. Fundamental genetic operators: crossover, mutation and selection, for this problem, are proposed as follows. 2-point crossover is used where crossover site will divide the rows or columns of matrix C. 1st part of (some rows or columns) one matrix is combined with 2nd part of the other matrix and we will get one new offspring. Similarly we get the other offspring by combining remaining parts of these two matrixes. Mutation randomly interchanges place of '1' (column) present in one row to the place of '1' present in other row. After applying crossover or mutation the generated offsprings may be illegal. Thus a repair function is needed to convert this illegal chromosome into a legal chromosome (that represents a bijective mapping). This repair function will check if any row or any column of a chromosome matrix contain more than one '1s', it inverts any '1' to '0' and any '0' to '1' such that each row and each column contains a single '1'.

Any selection method for e.g. ranking or proportional selections can be used. The rationale is that the chromosome with lower cost function value will have a higher probability of surviving into next generation. Terminating condition may be chosen as no. of generations. After a large number of generations it will generate a mapping that will be corresponding to sub-isomorphism between design pattern and the system design graph.

IV. issues in Design pattern detection using inexact graph-matching

There are 23 GoF (Gang of Four) [1] design patterns. UML diagrams can be drawn for each of the corresponding design patterns. After checking sub isomorphism between the relationship graphs of a design pattern and the model graph, there may be three cases [14]:

i) Relationship graph of a design pattern is (sub) isomorphic to the model graph.
ii) Relationship graph of a design pattern is partially (sub) isomorphic to the model graph.
iii) Relationship graph of a design pattern is not sub isomorphic to the model graph.

In the case i) design pattern exist in model graph and we should find at least one minimum error (without having q) bijective matching such that for all matched nodes there corresponding edges are same.

In the case ii) design pattern partially exists in the model graph. In this case we will not find any minimum error (without having q) bijective matching such that for all the matched nodes, there corresponding edges are same. Objective function value in this case would always have q in its expression since for at least one relationship between two matched nodes of DPG will not match the relationship between corresponding nodes in MG.

In the case iii) design pattern does not exist in the model graph. In this case we will not find any minimum error (with or without having q) bijective matching such that for atleast two matched nodes the corresponding edges are same.

V. Related Work

Brown [7] proposed a method for automatically detection of design patterns. In his work Smalltalk code was reverse-engineered to facilitate the detection of four well-known patterns from the catalog by Gamma et al. [1].

Many genetic-based search approaches are used to solve the problem of design pattern detection. Error correcting graph isomorphism problem that has been utilized in design pattern detection is also solved by genetic algorithm [6].

Nikolaos Tsantalis [3], proposed a methodology for design pattern detection using similarity scoring between graph vertices. It can detect variants of design patterns also. But the limitation of similarity algorithm is that it only calculates the similarity between two vertices, not the similarity between two graphs. To solve this Jing Dong [4] gave another approach called template matching, which calculates the similarity between subgraphs of two graphs instead of vertices. They detected design patterns from software by using normalized cross correlation.

Stencel and Wegrzynowicz [5] proposed a method for automatic design pattern detection that is able to detect many nonstandard implementation variants of design pattern. Their method was customizable because a new pattern retrieval query can be introduced along with modifying an existing one and then repeat the detection using the results of earlier source code analysis stored in a relational database. Drawback was that the method was not general enough to identify all design patterns. Further the translation of first order logic formulae as SQL queries is very laborious and error-prone.

In our earlier work, we used klenberg approach and fuzzy graph algorithms for design pattern detection [8]. 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 [8]. 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.

VI. Conclusions

In this paper the system design and a design pattern is taken and it is to find out whether design pattern matches (fully or partially) to any subgraph of system design by using genetic algorithm. The inexact graph matching technique is used. The encoding scheme and genetic operators: crossover, mutation and selection are discussed for solving this problem. The fitness function can be modified to improve the performance of the algorithm. We are developing a prototype that allows the implementation of the approach discussed.

References

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

[2] Zhi-Xiang Xhang, Qing-Hua Li and Ke-Rong Ben: A New Method for Design Pattern Mining, IEEE Explore, 2004.

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

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

[5] Stencel K. and Wegrzynowicz P.: Detection of Diverse Design Pattern Variants, 15th Asia-Pacific Software Engineering Conference, IEEE Computer Society, (2008).

[6] 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)

[7] 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)

[8] A. Pande, 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 (2010)

[9] A. Pande, M. Gupta: A New Approach for Design Pattern Detection Using Subgraph Isomorphism. In Proc. Of National Conference on Mathematical Techniques: Emerging Paradigm for Electronics and IT Industries(MATEIT-2010) (2010)

[10] A. Pande, M. Gupta, A.K. Tripathi. Design Pattern Mining for GIS Application using Graph Matching Techniques. 3rd IEEE International Conference on Computer Science and Information Technology . 09-11 July, 2010, Chengdu, China (2010)

[11] A. Pande, M. Gupta, 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. 09-11 August, 2010, Noida, India (2010)

[12] A. Pande, M. Gupta, A.K. Tripathi. A Decision Tree Approach for Design Patterns Detection by Subgraph Isomorphism, International Conference on Advances in Information and Communication Technologies, ICT 2010, Kochi, Kerala, published by Springer.

[13] A. Pande, M. Gupta, A.K. Tripathi. DNIT – A New Approach for Design Pattern Detection, International Conference on Computer and Communication Technology (ICCCT-2010), proceedings to be published by the IEEE (Accepted).

[14] Gupta M., Singh R.R., Pande A., Tripathi A.K.: Design pattern Mining Using State Space Representation of Graph Matching, 1st International Conference on Computer Science and Information Technology, Banglore, 2011, to be published by LNCS, Springer, (Accepted)