Design Pattern Detection by Greedy Algorithm Using Inexact Graph Matching

[1]Rajwant Singh Rao, [2]Manjari Gupta
[1]Guru Ghasidas Vishwavidyalaya, Bilaspur (C.G.), [2] Banaras Hindu University Varanasi

Abstract

Design Patterns are proven solution to common recurring design problems. Design Pattern Detection is most important activity that may support a lot to re-engineering process and thus gives significant information to the designer. Knowledge of design pattern exists in the system design improves the program understanding and software maintenance. Therefore, an automatic and reliable design pattern discovery is required. Graph theoretic approaches have been used for design pattern detection in past. Here we are applying an algorithm which decomposes the graph matching process into K phases, where the value of K ranges from 1 to the minimum of the numbers of nodes in the two graphs to be matched. The effectiveness of this algorithm results from the use of small values of K, and significantly reduces the search and space and producing very good matching between graphs. The same algorithm we are here using for design pattern detection from the system design.

Keywords—*design pattern, UML, matching, sub graph isomorphism, bijective, error matching.*

1. Introduction

Graph based approached have been used in many software engineering problems. Design Patterns are proven solutions for common recurring software design problems. The design patterns have been extensively used by software industry to reuse the design knowledge [1]. During maintenance of a software system the necessary tasks are to understand and modify it. It would be helpful to discover pattern instances in it, if any. Many algorithms have been proposed for design patterns detection like [2, 3, 4, 5]. Similar works on design pattern detection have been discussed in section 2.

This paper presents a design pattern detection technique by greedy algorithm using a graph matching algorithm. Here, the graphs are corresponding to the relationship graphs which exist in the UML diagrams of system design (model graph or system under study) as well as in UML diagrams of design patterns. In the classic concept of exact graph matching, the aim is to determine whether two graphs are the same or whether a subgraph of one exists in the other. In practical applications, objects are often affected by noise and distortion, so using exact graph matching often fails to find the appropriate solution. One way to handle with noisy data is to use inexact graph matching. The objective is to find a (sub) graph isomorphism that tolerates distortions; this is known as an error-correcting or error-tolerant (sub) graph isomorphism [7].

The algorithm is based on the greedy algorithm. A greedy algorithm always makes the choice that looks best at the moment. That is it makes a locally optimal choice in the hope that this choice will lead to go a globally optimal solution. In this way it gives the best matching between both of the model graph and design pattern graph. Here we decomposes the matching process into *K* phases, where the value of *K* ranges from 1 to the minimum of the numbers of nodes in the two graphs to be matched. The efficiency of the new algorithm results from the use of small values of *K*, which significantly reduces the search space while still producing very good matchings (most of them optimal) between graphs [7]. The outline of this paper is as follows. In section 2 related works are discussed. Section 3 explains the representation of model graph and design patterns in terms of relationship graphs is explained. The graph matching algorithm is described in section 4. In section 5 the design pattern detection is described using some examples. Lastly we concluded in section 6.

2. Related Work

The first attempt for automatically detecting design pattern was by Brown [8]. 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. [9] developed a technique to identify structural patterns in a system 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. However, the limitation of similarity algorithm is that it only calculates the

similarity between two vertices, not the similarity between two graphs. Jing Dong [3] gave another approach called template matching, which calculates the similarity between sub graphs of two graphs instead of vertices, to solve the above limitation. S. Wenzel [4] purposed a 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. Bergenti and Poggi [10] developed a method that examines UML diagrams and proposes the software architect modifications to the design that lead to design patterns. In our earlier work, we used several graph matching techniques to detect design patterns [12-].

3. Relationship Graphs Representation

The UML diagram of the system design as well as design pattern in converted into graphs. Classes of UML diagram are represented as nodes and relationships among classes by edges. Each node and edge is labelled. 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 we are considering only three attributes of a class. Each edge is corresponding to one of the relationships. We assign label 1 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. In this paper, we consider this graph as a model graph of system design.

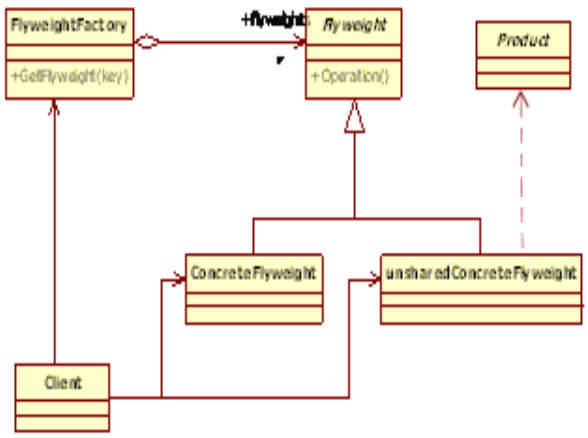

Fig. 1. UML Diagram of system design [11]

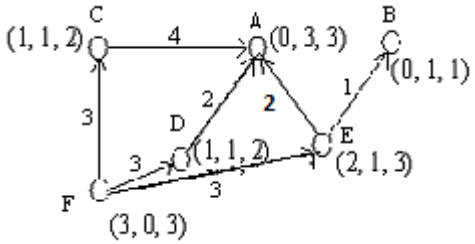

Fig. 2. Model graph corresponding to system design

4. Graph Matching Algorithm

The graph matching algorithm [6, 7] determines whether two graphs are the same or whether a graph is a subgraph of the other or whether a subgraph of one exists in the other. Given two graphs, the aim is to find the matching between their nodes and edges that leads to the minimum matching error between two graphs. This error is defined as the dissimilarity between each pair of matched nodes, plus the dissimilarity between corresponding edges. It can be viewed as the distance between the two graphs. In this algorithm the first part of matching error can be found by summing the matching errors of the node mapping. The second part of matching error is error in edge mapping. The basic idea of the new algorithm is to iteratively explore the possible node mappings and to select the best mapping at each iteration phase. The essential assumption behind this algorithm is that in order to obtain a good (or optimal) matching between two graphs, one should match similar nodes and corresponding relationships in the two graphs.

A. Algorithm description

Given two graphs, design pattern graph DPG = (V_1, E_1, u_1, v_1) and model graph MG = (V_2, E_2, u_2, v_2), where V_1, V_2 are set of nodes, E_1, E_2, are set of edges, u_1, u_2 are functions assigning labels to nodes and v_1, v_2 are functions assigning labels to edges in DPG and MG respectively.

To extract the most promising mapping, an n×m matrix P = (p_{ij}) is introduced, where n and m are the numbers of nodes in the DPG and MG, respectively. Each element p_{ij} in P denotes the dissimilarity between node i in G_1 and node j in G_2. In order to extract promising mappings, we use a second n×m matrix B = (b_{ij}) whose elements represent promising node-to-node mappings. The algorithm first initializes matrix P by setting p_{ij} = d

(m_1 (v_i), m_2 (v_j)), where p_{ij} is distance between v_i and v_j. In the first phase of iteration, the algorithm initializes matrix *B* by setting most elements to zero (b_{ij}= 0) and the others to 1, depending on their corresponding values in matrix P. Specifically, for each row in matrix B, the elements corresponding to the minimum elements in the same row of matrix P will be set to 1. Then, for each possible mapping extracted from B, the algorithm computes the error generated by nodes and the error generated by edges. The mapping that gives rise to the smallest matching error will be recorded. In the second phase, the algorithm will reset some elements in each row of matrix B to 1, specifically, those elements that correspond to the second-smallest elements in each row of matrix P. The algorithm will extract those isomorphisms from matrix B that contain at least one node-to-node matching added to matrix B at this phase. Of these isomorphisms and those obtained in the first phase, those with the smallest cost are retained. The algorithm then proceeds to the next phase and so on [6].

A matrix B' is introduced to keep a copy of all possible node-to-node matchings that have been considered by the algorithm so far. B is used as a 'temporary' matrix. At each phase (except the first), each of the n rows of B is examined successively. For each row i of B, all of the previous rows of B will contain all of the possible node-to-node matchings examined so far. Row i contains only the possible node-to-node matchings in the present phase. Finally, all of the following rows of B will contain only the possible node-to-node matchings examined in the previous phases. Such a matrix B guarantees that the isomorphisms extracted as the algorithm progresses will never be the same and that all of the isomorphisms that need to be extracted at each phase will indeed be extracted [7].

The algorithm is given below

Input: DPG and MG.

Output: matching between nodes in DPG and MG,

1. Initialize P as follows:
   For each p_{ij}, set pij= d (m_1 (v_i), m_2 (v_j)).
2. Initialize B as follows:
   For each b_{ij} , i =1,..., n and j = 1,...,m , set b_{ij} = 0 .
3. While Current _ Phase < K
   If Current _ Phase = 1,
   Then for all *i* = 1,..., n
   select the element with the smallest value in
   P that is not marked 1in B and set it to 1 in B;
   call Matching_Nodes(B).
   Else for all *i* = 1,..., *n*
   set B' = B
   for all j =1,...,m set b_{ij} = 0
   select the element with the smallest
   value in P that is not marked 1 in B'
   and set it to 1 in B and B';
   call Matching_Nodes(B);
   set B = B'.
   If all elements in B are marked 1,
   Then set Current _ Phase = K
   Else add 1 to Current_Phase.

End

Procedure: Matching_Evaluation(B)

For each valid mapping in B

1.Compute the matching error generated by nodes that is difference between matched nodes i.e. $|MG_{t1}-DPG_{t1}|+ |MG_{t2}-DPG_{t2}|+ |MG_{t3}-DPG_{t3}|$, where MG_{ti} is the i^{th} component of the label of node in model graph and DPG_{ti} is the i^{th} component of the label of matched node in the design pattern graph.

2. Add the error generated by the corresponding edges to the matching error. We take difference between labels of matched edges. If it is 0 then this part of matching error is 0 since both edges are corresponding to same relationship, otherwise we will set edge matching error to q where q is a very large positive number to show that edges are not corresponding to same relationship.

3. Save the actual matching if the matching error is minimal.

5. Design Pattern Detection Using Graph Matching algorithm

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

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. In the case ii) design pattern partially exists in the model graph. In the case iii) design pattern does not exist in the model graph. All these cases are described in detail by using examples.

A. Design Pattern Detection as Strategy Design Pattern: Exact Matching

Firstly, we are considering strategy design pattern, the UML diagram and corresponding labelled graph (DPG) is shown in Fig. 3 and Fig. 4 respectively. In this case we find at least one minimum error (without having q) bijective matching such that for all matched nodes there corresponding edges are same.

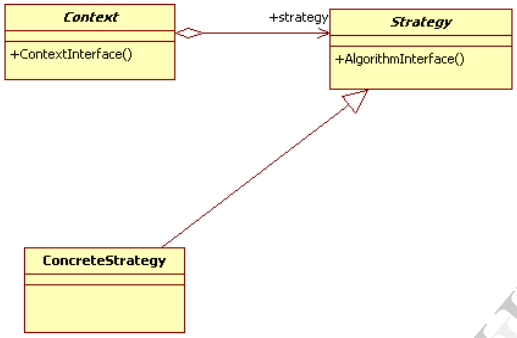

Fig. 3. Strategy Design Pattern [11]

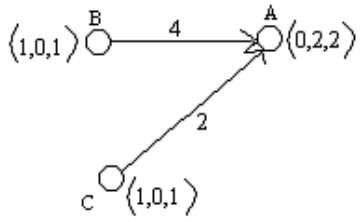

Fig. 4. DPG for strategy design pattern

Considering the DPG (Fig. 4) and MG (Fig. 2) the matrix P, in form of matching errors of nodes, shown in following Table 1.

Table 1. Matrix P

0,1,1	0,1,1	1,1,0	1,1,0	2,1,1	3,2,1
1,3,2	1,1,0	0,1,1	0,1,1	1,1,2	2,0,2
1,3,2	1,1,0	0,1,1	0,1,1	1,1,2	2,0,2

After adding the matching errors of each cell, the final matrices P and B for phase 1 are shown in table 2 and table 3 respectively.

Table 2. Matrix P

2	2	2	2	4	6
6	2	2	2	4	4
6	2	2	2	4	4

Table 3. Matrix B / Phase 1

1	0	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

From the matrix in Table 3, the matching {(DPG_A, MG_A), (DPG_B, MG_B), (DPG_C, MG_B)} will be extracted but this is not bijective, so it will not be considered for the further exploration. In phase 2, matrices *B* and *B'* will be as follows at step 1:

Table 4. Matrix B Step 1 /Phase 2 And Matrix B' Step 1 /Phase 2

0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

1	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

Here the matching {(DPG_A, MG_B), (DPG_B, MG_B), (DPG_C, MG_B)} is extracted which is not bijective, so it will also not be considered for exploration.

Table 5. Matrix B Step 2 /Phase 2 And Matrix B' Step 2 /Phase 2

1	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

1	1	0	0	0	0
0	1	1	0	0	0
0	1	0	0	0	0

In step 2 matchings {(DPG_A, MG_A), (DPG_B, MG_C), (DPG_C, MG_B)} and {(DPG_A, MG_B), (DPG_B, MG_C), (DPG_C, MG_B)} are extracted but only the first matching is bijective and its matching error is (6+q).

Table 6. Matrix B Step 3 /Phase 2 And Matrix B' Step 3 /Phase 2

1	1	0	0	0	0
0	1	1	0	0	0
0	1	0	0	0	0

1	1	0	0	0	0
0	1	1	0	0	0
0	1	1	0	0	0

In step 3 the only one matching {(DPG_A, MG_A), (DPG_B, MG_B), (DPG_C, MG_C)} is bijective and the matching error is (6+q).

At step 1 of phase 3, matrices *B* and *B'* are as follows:

Table 7. Matrix B Step 1 /Phase 3 And Matrix B' Step 1 /Phase 3.

0	0	1	0	0	0
0	1	1	0	0	0
0	1	1	0	0	0

1	1	1	0	0	0
0	1	1	0	0	0
0	1	1	0	0	0

From table 7 no bijective matching will be extracted.

Table 8. Matrix B Step 2 /Phase 3 And Matrix B' Step 2 /Phase 3

1	1	1	0	0	0
0	0	0	1	0	0
0	1	1	0	0	0

1	1	1	0	0	0
0	1	1	1	0	0
0	1	1	0	0	0

In step 2 of phase 3, the bijective matchings are {(DPG_A, MG_A), (DPG_B,MG_B), (DPG_C, MG_B)}, {(DPG_A, MG_A), (DPG_B, MG_D), (DPG_C, MG_C)}, {(DPG_A, MG_B), (DPG_B, MG_D), (DPG_C, MG_C)} and {(DPG_A, MG_C), (DPG_B, MG_D), (DPG_C, MG_B)}, and the matching error of these matchings are (6+q), (6+q), (6+q) and (6+q) respectively.

Table 9. Matrix B Step 3 /Phase 3 And Matrix B' Step 3 /Phase 3.

1	1	1	0	0	0
0	1	1	1	0	0
0	0	0	1	0	0

1	1	1	0	0	0
0	1	1	1	0	0
0	1	1	1	0	0

In step 3 of pase 3, the bijective matchings are {(DPG_A, MG_A), (DPG_B,MG_B), (DPG_C, MG_D)}, {(DPG_A, MG_A), (DPG_B, MG_C), (DPG_C, MG_D)}, {(DPG_A, MG_B), (DPG_B, MG_C), (DPG_C, MG_D)} and {(DPG_A, MG_C), (DPG_B, MG_B), (DPG_C, MG_D)}, and the matching error of these matchings are (6+q), 6, (6+q) and (6+q) respectively. Thus here according to greedy algorithm we can see, the matching {(DPG_A, MG_A), (DPG_B, MG_C), (DPG_C, MG_D)} has the minimum error and also all the matching nodes and their corresponding edges are same in DPG and MG, so the strategy design pattern completely exist in the model graph.

B. Design Pattern Detection as Command Design Pattern: Partial Matching

In some cases it is also possible that a particular design pattern partially exist in the system design pattern (case ii discussed in section 4). For example consider the command design pattern, the UML diagram and corresponding labelled graph (DPG) is shown in Fig. 5 and Fig. 6 respectively. In this case we will not find any minimum error (without having q) bijective matching such that for all matched nodes there corresponding edges are same. Minimum matching error 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.

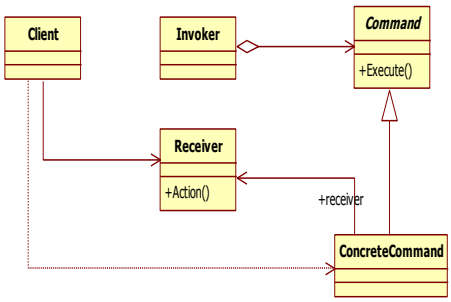

Fig. 5. Command Design Pattern [11]

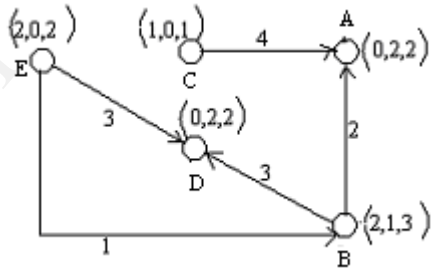

Fig. 6. DPG of command design pattern.

By considering DPG (Fig. 6) and MG (Fig. 2), the matrix P in form of matching errors of nodes is shown in following Table 1.

Table 10. Matrix P

0,1,1	0,1,1	1,1,0	1,1,0	2,1,1	3,2,1
2,2,0	2,0,2	1,0,1	1,0,1	0,0,0	1,1,0
1,3,2	1,1,0	0,1,1	0,1,1	1,1,2	2,0,2
0,1,1	0,2,2	1,1,0	1,1,0	2,1,1	3,2,1
2,3,1	2,1,1	1,1,0	1,1,0	0,1,1	1,0,1

After adding the matching errors of each cell the final matrices P and B for phase 1 are shown in tables 11 and table 12 respectively.

Table 11. Matrix P

2	2	2	2	4	6
4	4	2	2	0	2
6	2	2	2	4	4
2	4	2	2	4	6
6	4	2	2	2	2

Table 12. Matrix B / phase 1

1	0	0	0	0	0
0	0	0	0	1	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	1	0	0	0

From the matrix in Table 12, we are not getting any bijective matching.

In phase 2, matrices B and B' will be as follows at step 1:

Table 13. Matrix B Step 1 /Phase 2 And Matrix B' Step 1 /Phase 2.

0	1	0	0	0	0
0	0	0	0	1	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	1	0	0	0

1	1	0	0	0	0
0	0	0	0	1	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	1	0	0	0

In this step no bijective matching is found. Similarly for step 2/phase 2 there will not be any bijective matching.

Table 14. Matrix B Step 3 /Phase 2 And Matrix B' Step 3 /Phase 2.

0	1	0	0	0	0
0	0	1	0	1	0
0	0	1	0	0	0
1	0	0	0	0	0
0	0	1	0	0	0

1	1	0	0	0	0
0	0	1	0	1	0
0	1	1	0	0	0
1	0	0	0	0	0
0	0	1	0	0	0

In step 3 the bijective matchings are {(DPG_A, MG_B), (DPG_B, MG_E), (DPG_B, MG_C), (DPG_D, MG_A), (DPG_E, MG_C)} and their matching errors are same i.e. (8+q).

Similarly, if we proceed further, we will find that for all the bijective matchings that we will get, the matching error is (8+q). That is in those mappings, some of the matching nodes have same corresponding edges while for some of the matching nodes corresponding edges are not same in MG and DPG. Thus in this case command design pattern partially exists in the model graph.

C. Particular design pattern may not exist

Above we have seen the examples of design pattern existence (complete or partially) but it can be possible that a particular 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 even for few numbers of matched nodes there corresponding edges are same.

For example if we take singleton design pattern (Fig. 7), there is only one relationship: direct association on itself node. Corresponding DPG is shown in Fig. 8.

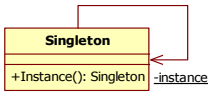

Fig. 7. Singleton Design Pattern [11]

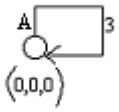

Fig. 8. DPG of Singleton design pattern

Consider DPG (Fig. 8) and MG (Fig 2) the matrix P, in form of matching errors of nodes are shown in Table 15.

Table 15. Matrix P

0,3,3	0,1,1	1,1,2	1,1,2	2,1,3	3,0,3

After adding the matching errors of cell, the final Matrix P and B for phase 1 are shown in tables 16 and 17, respectively.

Table 16. Matrix P

6	2	4	4	6	6

Table 17. Matrix B / Phase 1

0	1	0	0	0	0

From the matrix in Table 17, there will be bijective matching {(DPG_A, MG_B)} and its matching error is (2+q). Since for the matched node, no corresponding edge is found so singleton design pattern does not exist in the model graph.

6. Conclusions

This paper presents an approach for design pattern detection using error correcting representation of graph matching. We took the relationship graphs of the model graph (MG) and a design pattern (DPG), after that the graph matching algorithm is applied on both of the graphs and tried

to find out the bijective matching. If for this bijective matching, the matched nodes have the corresponding edges, we say that the design pattern exist in the model graph. If for the matched node no corresponding edges are found, design pattern does not exist in the model graph, and if for the matched nodes some of the corresponding edges are found and some are not found, we say that design pattern partially exists in the model graph. The advantage of this approach [6, 7] is that it reduces the search space and produce very good matchings (most of them optimal) between graphs. In general, the (worst) complexity of the algorithm depends on the number of phases (value of K) used by the algorithm. For a given value K ($0 \le K \le m$), each row in matrix B has K elements marked 1 and there are K^n node-to-node mappings to be extracted. To check the edge-to-edge mappings, the algorithm needs $O(n^2)$ steps for each mapping. Thus, the complexity of the algorithm is $O(n^2 K^n)$ [6, 7]. We are trying to develop a prototype that allows the implementation of the algorithm discussed.

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] Jing Dong, Yongtao Sun, Yajing 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.

[4] S. Wenzel and 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, October 2006.

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

[6] Adel Hlaoui, Shengrui Wang, "A New Algorithm for Inexact Graph Matching," *Pattern Recognition*, 16th International Conference on Pattern Recognition (ICPR'02) - Volume 4, 2002.

[7] Adel Hlaoui and Shengrui Wang, "A Node-Mapping-Based Algorithm for Graph Matching", Sciences/DI, Université de Sherbrooke, Sherbrooke, Québec, Canada J1K 2R1

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

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

[10] F. Bergenti and A. Poggi, "Improving UML Designs Using Automatic Design Pattern Detection," Proc. 12th Int'l Conf. Software Eng. and Knowledge Eng. (SEKE '00), July 2000.

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

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

[13] Pande A. & Gupta M., "Design Pattern Mining for GIS Application using Graph Matching Techniques", 3rd IEEE International Conference on Computer Science and Information Technology. pp. 09-11, Chengdu, China, 2010.

[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.

[15] Gupta M. Singh R.R., Tripathi A.K., "Design Pattern Detection using Inexact Graph Matching", International Conference on Communication and Computational Intelligence, Tamilnadu, Dec 2010, to be published by IEEE Explore.

[16] Gupta M., "Inexact Graph Matching for Design Pattern Detection using Genetic Algorithm", International Conference on Computer Engineering and Technology, Nov 2010, Jodhpur, to be published by IEEE Explore.

[17] Manjari Gupta, Akshara Pande, Rajwant Singh Rao, A.K. Tripathi, Design Pattern Detection by Normalized Cross Correlation, International Conference on Methods and Models in Computer Sciences (ICM2CS-2010), December 13-14, 2010, JNU, to be published by IEEE Explore.

[18] Manjari Gupta (2011), Design Pattern Mining Using Greedy Algorithm for Multilabeled Graphs, International Joint Conference on Information and Communication Technology, 8-9 Jan, 2011, Bhubaneshwar, to be published by IPM Pvt. Ltd, India.