Article can be accessed online at http://www.publishingindia.com

An Idea Towards Improving Design Pattern Detection

Arti Chaturvedi*, Manjari Gupta, Sanjay Kumar Gupta*****

Abstract

Design Pattern Detection is a part of re-engineering process and thus gives significant information to the designer. Detection of design patterns is helpful for improving the software characteristics. Therefore, a reliable design pattern discovery is required. The problem of finding an isomorphic subgraph is used to solve design pattern detection in past. It is noticed that ordering of vertices of the design pattern saves the time of process. In this paper we are doing ordering of vertices for few design patterns proposed by Gamma, Helm, Johnson, and Vlissides (1995) using an algorithm "GreatestConstraintFirst" proposed by Bonnici, Giugno, Pulvirenti, Shasha, and Ferro (2013). After getting this ordering, we use a matching algorithm that uses subgraph isomorphism conditions to check whether a particular design pattern exists in the system design or not (Bonnici et al., 2013). We redefine subgraph isomorphism conditions in the context of the problem of mining design patterns from the system design.

Keywords: Design Pattern, UML, Subgraph Isomorphism, Ordering of Vertices

Introduction

Design patterns are the detailed information of classes that are useful when developing a software or program. The design patterns are solutions of many problems that come in software development process. They are frequently used by software companies. The availability of pattern related information is generally lost when they are implemented in a system. Therefore, it is tough to trace out such design information. To understand and modify a software system, it is necessary to discover pattern instances in it, if any. Many algorithms have been proposed for design patterns detection like Gueheneuc, Sahraoui, and Zaidi (2004), Wenzel and Kelter (2006), Tsantalis, Chatzigeorgiou, Stephanides, and Halkidis (2006). This is a well-known NP-hard problem. Thus evolutionary approaches are being used for this problem.

The problem of design pattern detection is solved by mapping it to finding sub-graph isomorphism. The idea is before the subgraph isomorphism process starts, the ordering of vertices of design pattern graph is done to maximise the chance that a partial match will be pruned away as early as possible. In this paper, system design and design pattern are expressed in graph format. The outline of this paper is as follows. Second section shows the graph format of system design and design pattern. The vertex ordering algorithm is described in third section. How to use this ordering for getting all occurrences of design patterns in the system design is discussed in fourth section. Related works are discussed in fifth section. Lastly we give the conclusion in sixth section.

Relationship Graphs Representation

In this section our designs are shown in graph format. In graph, classes are denoted by node and relationship denoted by edge. Each node and edge are labelled. The label of each node is abstract or concrete or concrete sub class. Thus we represent it by three tuple (t1, t2, t3) where

* Research Scholar, SOS in Computer Science and Application, Jiwaji University, Gwalior, Madhya Pradesh, India. Email: arti.2408@gmail.com

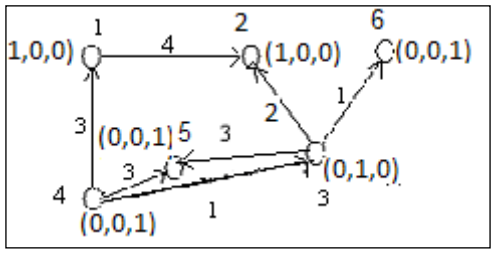
** Assistant Professor, Department of Computer Science, Faculty of Science, Banaras Hindu University, Varanasi, Uttar Pradesh, India. Email: manjari_gupta@rediffmail.com

*** Professor, SOS in Computer Science and Application, Jiwaji University, Gwalior, Madhya Pradesh, India. Email: Sanjaygupta9170@gmail.com

t1=1 if it is an abstract class, t2=1 if it is a concrete subclass, and t3=1 if it is a concrete class otherwise all tuples are 0. It can be modified to include more other attributes of a class. In this initial effort we are considering only these labels of a class. Further, each edge is corresponding to one of the relationships. We assign label 1 for dependency, 2 for generalisation, 3 for direct association, and 4 for aggregation.

The graph shown in Fig. 1 is system design graph.

Fig. 1: Model Graph Corresponding to System Design



Similarly the class diagram and corresponding graph for strategy and command design patterns are shown in Fig. 2 and 3 respectively.

Fig. 2: Strategy Design Pattern and its Corresponding Graph

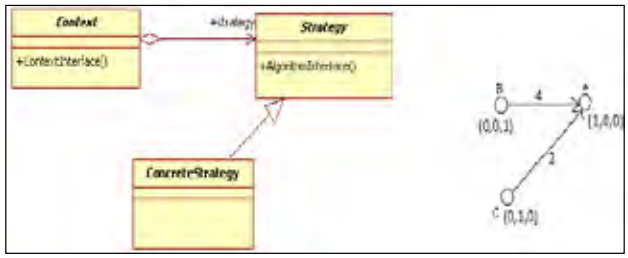
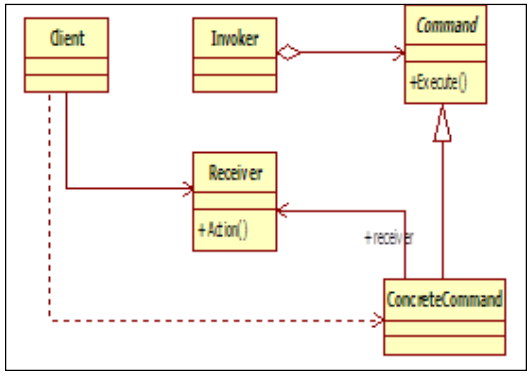
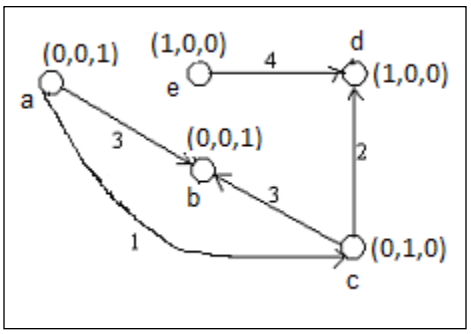


Fig. 3: Command Design Pattern and its Corresponding Graph





Ordering of Vertices of Design Patterns

In this section we will describe how the vertices of design patterns are ordered so that the process of matching vertices of graph for design patterns with vertices of system design graph can be speed up (Bonnici *et al.*, 2013).

Given the design pattern graph G(V,E), let n = |V|, the idea is to find the suitable sequence of vertices $\mu = (u_0, u_1,\ldots,u_n)$ of V, at each step i, the vertex $u_i \in V$ chosen is the one that maximises the size of the set $Bi = \{<ui, uj> \in E: uj \in \mu, 0<j\leq i\leq n\}$. Bi represents the set of edges in the design pattern graph connecting ui with vertices in μ. By making Bi as large as possible, the algorithm imposes the most constraints on corresponding subgraphs of a potential system design graph. That is, in the subgraph isomorphism process, RI first will be matched to nodes that are highly connected (i.e., a large number of constraints to verify) with nodes already matched.

""We use a greedy algorithm called Greatest Constraint First"" (Bonnici *et al.*, 2013) to find a good sequence of vertices μ. GreatestConstraintFirst visits the pattern graph based on a scoring function. It starts from a vertex u_0 in the design pattern graph that has the maximum number of neighbours among vertices in the design pattern graph. The algorithm iteratively proceeds until all vertices in the design pattern graph are inserted in μ. For each vertex v not yet in the sequence $(u_0, u_1,\ldots, u_{m-i})$ the concept of vertex parent is maintained, that is the vertex ui in the sequence with the smallest index i such that $<u_i, v> \in E$.

The scores are assigned in the following way (Bonnici *et al.*, 2013). Let m be the next step of a visit in the design pattern graph and let μ $(u_0, u_1,\ldots u_m\text{-}_1)$ the visited vertices so far, let u_m be the next candidate vertex to be inserted in μ.

One can ascribe a score to u_m using the following three sets.

1. $V_{m,vis} = \{u_i : 0 \leq i < m : \langle u_m, u_i \rangle \in E$, the set of vertices in μ that are neighbour of u_m.
2. $V_{m,neig} = \{u_i : 0 \leq i < m, \exists j > m: (u_i, u_j) \in E, (u_m, u_j) \in E\}$, the set of vertices in μ each of which is a neighbour of at least one vertex outside μ that is connected to u_m.
3. $V_{m\ unv} = \{u_j: j > m, (u_m, u_j) \in E, \forall\ i < m\ (u_i, u_j) \notin E\}$, the set of vertices that are not in μ, not even neighbour of vertices in μ but are neighbours of u_m.

The score of candidate u_m is a lexicographic score based on $|V_{m,vis}|$ as the high order quantity, followed by $|V_{m,neig}|$ and finally $|V_{m,unv}|$. Thus, suppose u_a and u_b are both candidates. The score of u_a is greater than the score of u_b if either (i) $|V_{a,vis}| > |V_{b,vis}|$ or (ii) $|V_{a,vis}| = |V_{b,vis}|$ and $|V_{a,neig}| > |V_{b,neig}|$ or (iii) $|V_{a,vis}| = |V_{b,vis}|$ and $|V_{a,neig}| = |V_{b,neig}|$ and $|V_{a,unv}| > |V_{b,unv}|$. If highest value is equal for the two vertices then choose one of them randomly.

Example:

The above algorithm can be used to order vertices of any design pattern that one want to search in the system design graph. We will see how this algorithm can be used to order vertices of command design pattern shown in Fig. 3.

In the graph corresponding to command design pattern shown in Fig. 3 there is node "c" with maximum degree. Thus μ= (c).

In the next step, the next candidate may be one of the remaining nodes. For all remaining nodes we will calculate $V_{m,vis}$, $V_{m,neig}$ and $V_{m,unv}$.

$V_{a,vis}$= 1, $V_{b,vis}$=1, $V_{d,vis}$=1 and $V_{e,vis}$=0.

Since there are three nodes having same value of $V_{m,vis}$, we will calculate $V_{a,neig}$, $V_{b,neig}$ and $V_{d,vis}$.

$V_{a,neig}$=1, $V_{b,neig}$ =1 and $V_{d,neig}$=0

$V_{m,vis}$ is same for node "a" and node "b". Thus we will calculate $V_{m,unv}$ for node "a" and node "b".

$V_{a,unv}$= 0 and $V_{b,unv}$ = 0

Since value of vertex "a" is equal to value of vertex "b", we select one of them randomly. Let us choose node "b" as a next candidate. So now μ= (c, b)

In the next step, again we will calculate $V_{m,vis}$ for the remaining nodes that are not included in μ.

$V_{a,vis}$= 2, $V_{d,vis}$=1, and $V_{e,vis}$=0

Thus our next candidate node is node "a". so μ= (c, b, a)

Next again we calculate $V_{m,vis}$ for the remaining nodes that are not included in μ i.e. for node "d" and node "e".

$V_{d,vis}$=1, and $V_{e,vis}$=0

Thus next candidate node is node "d". so μ= (c, b, a, d). now only one node is remained to be included in μ thus we can add that node in μ as the last node. Thus μ= (c, b, a, d, e).

If vertices of the command design pattern are matched in this order it will be fast to match/ search the graph of command design pattern in the system design graph.

Graph Matching Algorithm

The graph matching algorithm that we are using is based on some subgraph isomorphism conditions. A simple enumeration algorithm to find all the subgraph isomorphism of a design pattern graph in a model graph works as follows: generate all possible maps between the vertices of the two graphs and check whether any generated map is a subgraph isomorphism. "Whereas this algorithm is inefficient if done naively, it serves as a good starting point" (Bonnici *et al.*, 2013).

All the maps can be represented using a search space tree. The tree has a dummy root. "Each node represents a possible match between some vertex u of the design pattern G and some vertex u' of the system design graph G'" (Bonnici *et al.*, 2013). A matching sequence is similar in both, design pattern G and system design graph G' at every point of matching the node. At the time of checking nodes the isomorphic conditions are considered for proper matching. If isomorphic conditions are not matched at particular node, we return to parent of that node.

This algorithm generates an ordered sequence for design pattern graph vertices from root to leaf node.

Each step consists of choosing at each level i, the candidate vertices in the system design graph, $u_i = m(u_i)$ to match u_i, among the neighbours of the matched vertices of the parents of u_i. Parents of the design pattern graph are constructed in Greatest Constrain First.

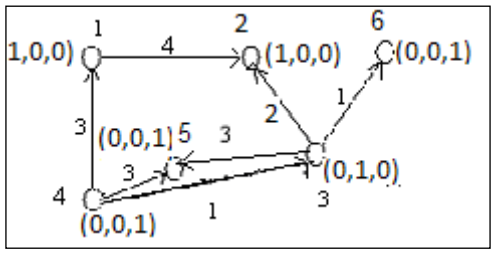
The following isomorphism conditions discard unnecessary node and edges.

1. Neither u_i nor $M(u_i)$ is already matched in the current path.
2. The matched vertices are compatible, i.e., $lab(u_i) = lab(M(u_i))$.
3. The number of edges connected to $M(u_i)$ in V' is greater than or equal to the number of edges connected to u_i in V. that is $|\{(v',M(u_i)) \in E'\}| \geq |\{(w,u_i) \in E\}|$ and $|\{M(u_i),v') \in E'\}| \geq |\{(u_i,w) \in E\}|$. In the case of undirected graphs it verifies that $|(M(u_i),v') \in E'\}| \geq |(u_i,w) \in E\}|$.
4. Labels of edges connected to $M(u_i)$ in V' is same as the labels of edges connected to u_i in V.
5. The constraints deriving from the topology of the pattern graph up to this point in the path are met, for all $u_i, u_j \in V$ where $0 \leq j \leq i (u_i,u_j) \in E \Rightarrow (M(u_i),M(u_j)) \in E'$. The compatibility of the edge labels are then verified since edges are labeled.

The isomorphism conditions are tested in the above. Condition i is verified only if condition i-1 dose not fail. Condition 1, 2, and 4 ensure the isomorphism, whereas the third one is a filtering test that often obviates the need for the substantial work needed to verify condition 4.

We can elaborate the condition 5 as follows: after getting a new matched pair, we check the connectivity (set of edges E1) of the node of design pattern present in newly matched pair with all other nodes of design patterns that have already been added in the mapping M. Similarly we will check the connectivity (set of edges E2) of node of target graph present in newly matched pair with all other nodes of target graph that have already been included in mapping M. If $E1 \subset E2$ then this pair will be added in the mapping.

Fig. 4: Model Graph Corresponding to System Design



We will see the execution of this algorithm by searching command design pattern graph shown in Fig. 3 in the following target graph corresponding to some design.

The order of vertices, to be searched in the target graph, in the command design patterns are $\mu = (c, b, a, d, e)$.

So we will start the searching of mapping of node "e" (of design pattern graph) in the system design graph. Initially no node of the target graph is already matched in the current path thus all node of the target graph are the candidate nodes.

1. The node "e" is compatible with two nodes of target graph: "1" and "2" as the label of node "e" is matched with label of nodes "1" and "2" in the target graph. Number of outgoing edges from node "1" ≥ number of outgoing edges of node "e". Edge labels of these outgoing edges are same i.e. "4". The condition, number of incoming edges to node "1"≥ number of incoming edges to node "e" is also satisfied. Thus node "e" is mapped to node "1" of the design pattern. While number of outgoing edges from node "2"< number of outgoing edges of node "e". Thus node "e" cannot be mapped to node "2". Thus the mapping M is {(e,1)} till now.
2. The next vertex in the order vertices is node "c". Thus the next step is to search the mapping of node "c". Only node "3" is compatible to node "c" as labels of both nodes are same. Number of incoming edges of node "3" ≥ number of incoming edges of node "c". As well as number of outgoing edges of node "3" ≥ number of outgoing edges of node "c". Labels of outgoing edges are "2" and "3" and labels of incoming edge are "1" in both design patterns and target graph. Thus node "c" can be mapped to node "3" of the design pattern. Now we will check the condition (5) to identify whether this pair (c,3) should be added into the mapping M or not. There is no edge from "c" to "e" as well as "e" to "c" in the design pattern graph. Similarly there is number edge from node "3" to node "1" as well as node from "1" to node "3" in the target graph. Thus we will include this pair the mapping. Thus the mapping till now is {(e,1), (c,3)}.
3. Next the mapping of node "d" will be searched. The label of node "d" is matched with labels of nodes "1" and "2". But node "1" is already considered for mapping. Thus we will check if node "d" can be mapped to node "2" or not. Number of incoming

edges to node "2" ≥ Number of incoming edges to node "d". There is number of outgoing edge neither from node "2" nor from node "d". Labels of both these incoming edges are also same. Thus node "d" is mapped to node "2". Now it is to be checked that whether pair (d,2) can be added in mapping M or not. Again we will do this by checking condition (5). So we will check whether all the edges from node "d" to nodes "e" and "c" in design pattern graph are present in edges from node "2" to nodes "1" and "3" in graph of target graph. There is only one edge from node "e" to node "d" and one edge from node "c" to node "d". Similarly there are one edge from node "1" to node "2" and one edge from node "3" to node "2". Labels of corresponding edges are also same. Thus we will add pair (d,2) in the mapping M.

4. Next node in the order is node "b". It is compatible to three nodes in the target graph: node "4", "5" and "6". But number of incoming edges of node "b" ≥ number of incoming edges of node "5". Labels of these edges are also same. While number of incoming edges of node "b"< number of incoming edges of nodes "4" and "5". Thus this condition is not true for node "4" and "6". And so only candidate node for mapping is "5". There is number outgoing edge from node "b" as well as from "5". Thus node "b" can be mapped to node "5". Let us check the condition (5) to explore whether this pair (b,5) can be added to mapping M or not. The connectivity among node b and nodes that has already been included in mapping M i.e. "e", "c", "d" is to be checked. There is only one edge from node "c" to node "b" with label "3". Similarly we will check connectivity of node "5" to nodes of target graph already added in the mapping i.e. nodes "1", "3" and "2". There is a single edge from node "3" to node "5" with label "3". Thus this connectivity is same and we can add pair (b,5) in the mapping M.
5. The last node in the sequence is "a". Label of this node matches with label of node "4", "5", and "6". Node "5" is already considered so there are two nodes applicant for mapping. Number of incoming edges to node "4" and "6" ≥ number of incoming edges to "a". Number of outgoing edges from "4"≥ number of outgoing edges from "a". Labels of these edges are also same. Thus node "a" is mapped to node "4." Now we will check the connectivity (condition 5) among node "a" and nodes "e", "c", "d", and "b". Similarly the connectivity among node "4" and nodes "1", "3", "2", and "5". There is one edge from node "a" to "b" with label "3" and one edge from node "a" to "c" with label "1". There is one edge from node "4" to node "1" with label "3". Similarly there is one edge from node "4" to node "5" with label "3". There is one edge from node "4" to "3" with label 1. Thus the connectivity among nodes "a" and nodes "e", "c", "d" and "b" are subset of connectivity among node "4" and nodes "1", "3", "2" and "5". Thus this pair (a,4) will be added in the mapping.

Thus mapping is M= {(e, 1), (c, 3), (d, 2), (b, 5), (a, 4)}. In this example there is only one instance of command design pattern. Practically there may be more than one occurrences of a design pattern in the system design graph. All these occurrences can be identified using the above algorithm.

Related Work

Gueheneuc *et al*. (2004) determine functionality of key classes in design by some measures and these measures are called fingerprints of the classes. Wenzel and Kelter (2006) explain differences of two graphical structure of design. The benefit of this approach is that the partial matchings are also found. Tsantalis *et al*. (2006) work on graphical structure of design and determine how to two vertices of different graphs are same. Dong, Sun, and Zhao (2008) find out one block of code from open source which follow the rules of particular pattern. These blocks are also showing different implementation of that pattern on the basis of eight characteristic of design like generalisation, association etc. Rasool and Maeder (2011) find out the reusable features of design pattern and give the solution of implementation variation of same pattern. Gupta, Pande, Rao, and Tripathi (2010) determine how much two graphical structure of design are same. For this they use "Normalised Cross Correlation" method. Pande and Gupta (2010) proposed a method of design pattern identification for GIS application using Graph Distance Approach, Normalised Crossed Correlation, and Subgraph Isomorphism. Dong, Lad, and Zhao (2007) present a design pattern detection approach based on the use of matrices and weights that explain the pattern matching process as mathematical calculation. DP-Miner tool is used to represent a matrix of inspected source code. Antoniol, Casazza, Penta, and Fiutem (2001) examine

the structure of design for selecting a particular pattern and measure the non-functional characteristics of the software.

Conclusion

A new approach for design pattern detection is discussed where the vertices of design patterns are ordered so that complexity of matching process can be reduced. In this paper we show two relationship graph one for design pattern and second for system design. Then we use vertex ordering algorithm for ordering the vertices of design pattern graph. This algorithm orders vertices such that the node whose connectivity is more would be taken first. After that the graph matching algorithm is applied on both of the graphs based on five conditions. Our future work is to develop a tool by implementing this detection algorithm and then identifying the design patterns from open source software frameworks/applications and then comparing this result from the result of existing tools for the same.

References

Antoniol, G., Casazza, G., Penta, M. D., & Fiutem, R. (2001). Object-oriented design patterns recovery. *Journal of Systems and Software*, 59(2), 181-196.

Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D., & Ferro, A. (2013). *A sub-graph isomorphism algorithm and its application to biochemical data.* From 9th Annual Meeting of the Italian Society of Bioinformatics (BITS) Catania, Sicily.

Dong, J., Lad, D. S., & Zhao, Y. (2007). *DP-miner: Design pattern discovery using matrix*. 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-based Systems, (pp. 371-380).

Dong, J., Sun, Y., & Zhao, Y. (2008). *Design pattern detection by template matching*. The Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC), (pp. 765-769).

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns elements of reusable object-oriented software*. Addison- Wesley.

Gueheneuc, Y., Sahraoui, H., & Zaidi, F. (2004). Fingerprinting design patterns. *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE).*

Gupta, M., Pande, A., Rao, R. S., & Tripathi, A. K. (2010). *Design pattern detection by normalized cross correlation*. International Conference on Methods and Models in Computer Sciences (ICM2CS-2010).

Pande, A., & Gupta, M. (2010). *Design pattern mining forGIS application using graph matching techniques*. 3rd IEEE International Conference on Computer Science and Information Technology. (pp. 09-11).

Rasool, G., & Maeder, P. (2011). *Flexible design pattern detection based on feature types*. 26th IEEE /ACM International Conference on Automated Software Engineering, (pp: 243-252).

Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., & Halkidis, S. (2006). *Design pattern detection using similarity scoring.* IEEE Transaction on Software Engineering, 32(11), 896-909.

Wenzel, S., & Kelter, U. (2006). *Model-driven design pattern detection using difference calculation*. In Proceedings of the 1st International Workshop on Pattern Detection for Reverse Engineering (DPD4RE), Benevento, Italy.