EXTRACTION OF OBSERVER PATTERN AND VISITOR PATTERN THROUGH OBJECT ORIENTED TECHNIQUE

Kamna Singh[1], Amrita Bhatnagar[2], Shweta Chaku[3]

[1,2,3]Department of Computer Science and Engineering,
Inderaprasth Engineering College, Ghaziabad, (India)

ABSTRACT

Design patterns are used as guidelines for faster and better understanding of software systems during software development. A design pattern has its own unique intent and describes the roles, responsibilities, and collaboration of participating classes and instances. Thus, by extracting design patterns from source code, we are then able to reveal the intent and design of a software system.In this paper we have addressed some design patterns. The approach to reverse engineer dynamic design patterns has been discussed here. we have taken three design patterns so far and proposed some techniques for extracting them from the java source code.

Keyword*: *Composite Pattern*, *Observer Pattern, Visitor Pattern,

I. INTRODUCTION

A software system is subject to changes throughout its lifetime. The understanding of existing system is important for the maintenance of software systems. Generally, the lack of documentation leads to high costs of reverse engineering in maintenance. The usage of design patterns benefits developers by helping them to reuse the knowledge of their experienced people.

Pattern is a description of common problem and a likely solution , based on experience with similar situation. A Design Pattern is essentially a description of a commonly occurring object-oriented design problem and how to solve it [12]. Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution.Hence, the common definition of a pattern: A solution to a problem in a context.Patterns can be applied to many different areas of human endeavor, including software development. [13].

In 1995 the now-classic text *Design Patterns* by Erich Gamma,Richard Helm, Ralph Johnson, and John Vlissides was published. *Design Patterns* basically focuses on one issue:Devising a set of objects and orchestrating an interaction between them to perform a computation can be a non-trivial problem. *Design Patterns* is essentially a catalog of 23 commonly occurring problems in object-oriented design and a pattern to solve each one.The authors are often called the Gang of Four (GoF)[12].

A design pattern names, abstracts, and identifies key aspects of acommon design structure that makes it useful for creating a reusable object-oriented design."[13]. Design patterns were first described by architect Christopher

Alexander in his book *A Pattern Language: Towns, Buildings, Construction* (Oxford University Press, 1977). The concept heintroduced and called *patterns* -- abstracting solutions to recurring design problems – caughtthe attention of researchers in other fields, especially those developing object-orientedsoftware in the mid-to-late 1980s [14].

II. DESIGN PATTERNS

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context. A design pattern abstracts a reusable object-oriented design that solves a common recurring design problem in a particular context .It is an effective means for the design, for the composition of several types of reusable components, and for the development of complex systems. It can improve the quality and understanding of programs, facilitate their development and increase their reuse[11].A pattern classification for reverse engineering should indicate whether or not each pattern is detectable and if there exist traceable concretepattern definitions to categorize detectable patterns[10].If design-patterns could be captured and reused in reverse engineering, the reverse engineering would be very helpful those who develops and maintains software.It has been seen that the main focus of any research is on the development of new approach. There are approaches on design pattern discovery have been proposed in the literature.A number of experiments on open-source systems have also been conducted by these approaches. However, different approaches reported different results when discovering the same design patterns in the same open-source systems.so the need to to validate and standardize the approaches of design pattern generation and set a benchmark.

III. PROPOSED TECHNIQUE

In this section we proposed the three technique of obtaining design patterns from java source code, these patterns are Observer pattern,visitor pattern and composite pattern

3.1 OBSERVER PATTERN Observer pattern defines a one-to many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It is a way of notifying change to a number of classes.

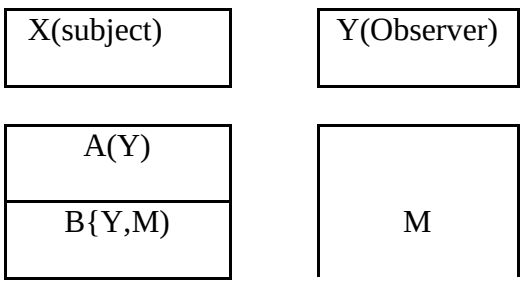

Figure 1: Classes showing Observer Pattern

In the above shown diagram A is the RegisterObservermethod,B and C are notify methods,M is an Update method.

In the above figure(1),the two classes X and Y depicts observer pattern.The class X is the subject class and class Y is the observer class .in class X ,we have the method A that has a parameter of type class Y,methods B and C which have method calls M from the class Y. In class Y we have a method M. The method A is called RegisterObserver method as it registers the objects from class Y as observer objects. The methods B and C in class X are called notify methods. The method M in class Y is called the update method.

The proposed algorithm to extract the observer pattern is as follows-

Step 1: Observer Set=empty
Step 2: Repeat for each class X step 3 to11 Step 3: TempSet=empty
Step 4: Repeat for each method A in X steps 5 to 10
Step 5: Repeat for each parameter type y in A steps 6 to 10
Step 6: if(Y is not a subclass of X) and (X is not a subclass of Y) and (X≠Y) then
Step 7: if (is Registry (X.A)) then
Step 8: Repeat for each call from X.B to Y.M steps 8 to 10
Step 9: if is Notify (X.B,Y.M)then
Step 10: TempSet= TempSet{(X.A,X.B,XM)}
Step 11: ObserverSet=Observer Set Temp Set

The isNotify and isRgister functions are implemented in the following manner-

isNotify(): It verifies whether a method behaves as a notify method. For example, A method X.B is a notify method iff X.B calls Y.M and Y is not a parameter of X.B.

isRegister(): It verifies that whether a method is a register method or not. The verifying condition tests whether the method potentially stores the passed argument for future use.

3.2 Visitor Pattern

Visitor defines a new operations without changing the classes of the elements on which it operates. It represents an operation to be performed on the elements of an object structure.

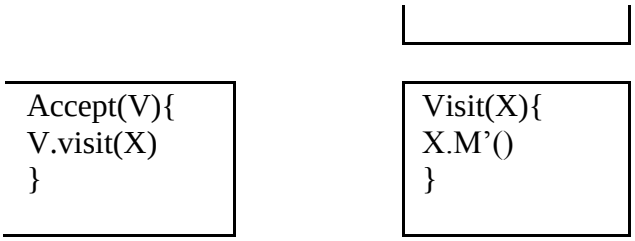

Figure 2: Visitor Pattern

The two classes above shown the visitor pattern.Class X is the element and class V is the visitor class . In class X we have a method accept which takes an object in its parameter list of the type of class V and has b method call from the class V and passes itself an argument. In the class V , there is a method named visit which takes an object of the class X and has a method call for a method ' of class X.

The proposed algorithm for extracting this pattern is as follows:

Visitoset=empty For each class X do Tempest=empty

For each method A in X do tempest=empty

for each method A in X do

for each parameter V in A do if V is a class then

for each method M in V do if(regular expression V.M(X) in

method A of class X)then tempest;=tempSet {(X,A,V,M0}

for each(X,A,V,M) tempSet do for each method A' in x do

if (regular expression X.A'()in method M of class V)then

VisitorSet=VisitorSet{(X,A,V,M)}

3.3 Composite Pattern

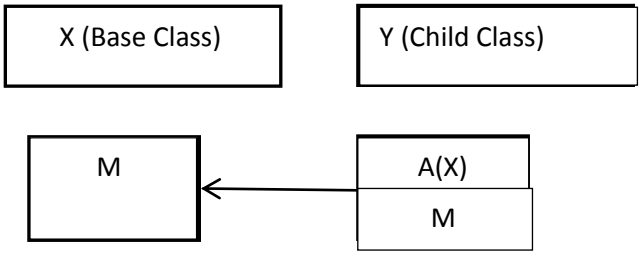

Figure 3: Composite Pattern

In the above figure ,we can see the composite design pattern. The class X in the base class and the class is the erieved or child class.the child class has a method that that accepts an object of type of the base class in its parameter list. The proposed algorithm is as given below:

The algorithm is as given below:

Step 1: CompositeSet=empty

Step 2: Repeat for each class X steps 3 to 11 Step 4: TempSet=empty

Step 5: Repeat for each method A in X steps 5 to 10

Step 6: if(X is a subclass of Y) then

Step 7: if(isRegisterComponent(X.A))then

Step 8: Repeat for each call from X.B to Y.M steps 8 to 10

Step 9: if(isComposite(X.B,Y.M)then

Step 10: Tempset=TempSet{(X.A,X.B,Y.M)

Step 11: CompositeSet=CompositeSet-TempSet

The following functions are required in the algorithm:

1)isComposite()

It verifies whether a method behaves as a composite method. For example,A method X.B is a composite method iff X.B calls Y.M and Y is not a parameter of X.B.

2) isRegisterComponent()

It verifies whether a method is a register component method or not. It tests,whether the method potentially stores the passed argument for future use.

3.4 Xpattern Tool

It is a comprehensive proposed tool to generate some of the dynamic design patterns.it generates observer, visitor and composite patterns. The tool has several sub components to support the different functionalities.

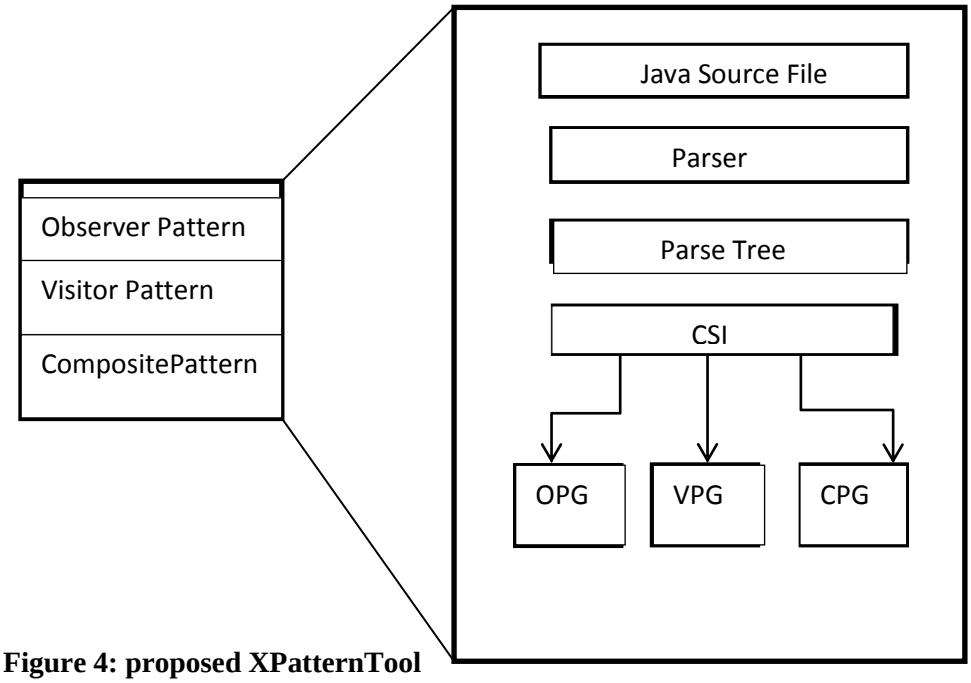

Figure 4: proposed XPatternTool

CSI : Class Structure Information

OPG : Observer pattern generator

VPG : Visitor pattern generator

CPG : Composite pattern generator

The sub components are explained as follows:

1) CSI(Class structure Information)

This component makes several passes in the parse tree to collect the information with respect to the classes. The following information is gathered in the multiple passes. Class Name Local Parameters ,Methods ,Method Calls ,Actual Parameter ,References of object variables

2) OPG(Observer Pattern Generator):

The OPG uses the information stored by the CSI component to process the following two tuple information which gives the observer patterns in te give in JAVA source code.

(<X.RegisterObserver><X.Notify>

<Y.Update>

Where X is the subject class and Y is the observer class.

3) VPG (Visitor pattern Generator)

The VPG subcomponents generated a four tuple information giving the visitor patterns present in the given JAVA source code.(<X><A><V><M>).The information in the tuple is

- X-it is the element class
- A-it is the accept method
- V-it is the visitor class
- M-it is the visit method.

4) CPG (Composite Pattern Generator):

The CPG sub component finds the class pairs which represent composite patterns. The reported tuple contains the classes showing composite patterns in the given JAVA source files.

(<X.RegisterComponent><X.Composite><Y.M>,Where

- X is the base class
- Y is the child class

IV.CONCLUSION

This paper proposed a technique for automatic generation of design patterns from the java source code. It is useful for beginners to understand java source code with the help of knowledge based visualization tool which provide result as a design patterns. Extracting design pattern instances from source code can help to understand and analyze the software systems.

V.FUTURE SCOPE

Our future work will expand its pattern recognition capability to recognize more complicated user-defined data structures; explore its use to detect design patterns in specific application domains, such as concurrent and real-time patterns; experiment with its use in tracking software evolution by design; and extend its overall usability by providing a visual specification language for defining patterns and exporting our analysis results as XMI for external viewing.

REFERENCES

[1] Anton Janson ,janBosh,parisAvgeriou, Documenting after fact:recoveringarchitectural design decisions,journal of Systems and software v.81,n.4,p536-557,April 2008.

[2] Andrea De Lucia,Vincenzodeufemia,CarmineGravino,MicheleRisi,Design pattern recovery Through visual language parsing and source Code analysis,Journal of System and Softwre ,v.82 n7,p-1177-1193,July 2009

[3] Dae-kyooKim,Wuweishen,An approach to Evaluating structural pattern conformance of UML models,Proceedings of the 2007 ACM Symposium on applied computing, March 11-15,2007,Seoul Korea

[4] Dirk Riehle,Design Pattern density defined,ACMSigplan notices,v.44 n.10,October 2009

[5] Jing Dong, Sheng Yang,KangZhang,Visualizing Design Pattern in their applications and Compositions, IEEE Transcations on Software Engineering, v.33 n.7, p.433-453, July 2007

[6] Susan kurian,Michael J Point,Themaintaince and Evolution of resources constrained embedded Systems created using design patterns,Journal of Systems and software ,v.80 n.1,p-32-41,Jan 2007

[7] Nija Shi and Ronald A. Olsson.Reverse Engineering of Design Patterns from Java Source Code ,Chikofsky E, Cross II J (1990) Reverse engineering and design recovery: a taxonomy. IEEE Softw7(1):13–17

[8] Gamma E., Helm R., Johnson R., and VlissidesJ.,Design Patterns, Elements of reusableObject- Oriented Software. Addison-WesleyPublishing Company, 1995.

[9] Berkane, M.L., Boufaida, M.: A process to reverse engineering based on aspect-oriented implementationof design patterns. In: 9thInternational Arab Conference on Information Technology. ACIT'2008, Tunisia. (2008).

[12] William H Mitchell "Design Pattern", proceddings Journal of System and Software 2003

[13] Bob torr , Design Pattern in Java,Journal of System and Software 2006

[14] Core Security Patterns - Best Practices and Strategies forJ2EE(TM), Web Services, and Identity Management, ChristopherSteel, Ramesh Nagappan and Ray Lai, Prentice Hall, 2005