Recovery of Design Pattern from source code

Amit Kumar Gautam and T. Gayen
Software Engineering, IIIT Allahabad
tgayen@iiita.ac.in, ise2008004@iiita.ac.in

Abstract. The approach for detecting design pattern from source code. In this approach present the four step process to find the design pattern from the source code. Generate the UML class diagram by applying reverse engineering process. Using the JAVEX tool which can take the class file as a input and give the information about properties of the attributes like inheritance, interface and obtained the relation between the classes. After that, get the information about runtime objects by which we obtained the behavior of the object created at runtime. Combine result of all the three steps and analyze the structure of the software and fetch the design pattern using.

1 Introduction

The design pattern describes a problem which occurs over and over again in our environment and describe core of the solution of that problem. It is basically description of communicating object and classes that are customized to solve a general design problem in a particular context.

A design pattern are basically
• Smart: a novice would not think of it quickly
• Generic: a pattern is meant for a problem and does not depend on a domain or technology
• Well proven: identified from real systems that have been applied several times
• Simple: usually quite small with just a few classes
• Reusable: design patterns are well documented so that they can be used in different contexts
• Object oriented: Usually
• Reusable solutions to common problems
• Names of abstractions above the class level
• Handling both functional and non-functional requirements

The motivation behind this work is the role played by Design pattern in source code is important. A design pattern provides knowledge about the role of each class within the pattern, the reasons of certain relationships among pattern constituents and/or the remaining parts of a system, localizing instances of the design patterns in existing software, can improve maintainability of software with other benefits like code comprehension, analysis of effects of using design patterns in software development. Some approaches have been proposed to detect design patterns from
source code or a design model, such as the UML diagrams. This work is a step in this direction.

2 Related Work

There are many existing methods for discovering design patterns from design and source code. Rudolf et. al.[7] give a pattern matching-based method which use the Columbus framework with which they find out the pattern instances from the source code by considering the patterns’ structural descriptions, but with this method they could not detect the same type of design patterns such as State and Strategy. Graph theory has also been applied in detection of design patterns by ascertaining similarity between the classes (vertices) in different systems (graphs) using the similarity score and iterative algorithm. Kleinberg proposed link analysis function to get the main hub and source nodes for web pages. Blondel et.al. [3] generalized this idea to an iterative algorithm for computing the similarity score for any two vertex’s. This similarity score algorithm for design pattern detection has been applied in by encoding the source code and design patterns into different feature matrices. Kramer and Prechelt [4] have given a tool called Pat, to localize instances of structural design patterns, extracting design information from a CASE tool repository and using Prolog facts to how the pattern is express. Bergenti et.al [5] give us a tool called IDEA (Iteractive Design Assistant). This tool is very effective to find the realization and finding the design pattern. IDEA automatically (a) getect pattern from the UML diagram. (b) generating critiques about these pattern. Basically IDEA idea is automatically detect the design pattern from the architect of the modules. Stencel et al. proposed the function to detect various implementation of design pattern. They presented its proof-of-concept implementation and also compared its efficiency to other state-of-the-art detection tools. The presented method is customizable. Dong et. al.[6] presented a novel approach to discovering design patterns by defining the structural characteristics of each design pattern in terms of weight and matrix. Their discovery process includes several analysis phases. Their approach is based on the XMI standard so that it is compatible with other techniques following such standard. They also develop a toolkit to support their approach. Francesca et. al. described an approach to design pattern detection using supervised classification and data mining techniques based on sub-components, and summarized the results they obtained on behavioral Design Patterns. Their Experiments with neural networks showed some encouraging results, but their instability led them to decision of employment of different techniques. [6]Jing et. al. presented some experiments on design pattern discovery from open-source systems using the tool they developed for design patterns detection: DP-Miner. In particular, their experiments discover the Adapter, Bridge, Strategy, and Composite patterns from the Java.AWT, JUnit, JEdit, and JHotDraw systems and experimental results show that design patterns have been widely applied in these systems and can also be recovered. In addition, they compared their experimental results with those of others and found several discrepancies. They analyzed this issue and discussed possible reasons for the discrepancies. More importantly, they argue for benchmarks for design pattern discovery. Damir presented
ontology-based architecture for pattern recognition in the context of static source code analysis. The proposed system has three subsystems: parser, OWL ontologism and analyzer. The parser subsystem translates the input code to AST that is constructed as an XML tree. The OWL ontologism define code patterns and general programming concepts. The analyzer subsystem constructs instances of the input code as ontology individuals and asks the reasoned to classify them. The recognition system is envisioned as a framework that can be used as a stand-alone utility, or as a subsystem for various larger systems, such as a compiler front end or IDE plug-ins. There are many other techniques that have been proposed earlier. The main problems encountered in using the above mentioned techniques are related to scalability, to many false positive results, and to the impossibility to find several design patterns; hence we decided to explore the problem and trying to overcome some of the mentioned or encountered difficulties.

3 Proposed Approach

In this approach the method by which obtained the information about the properties of object at compile time and runtime. Basically there are four stages to finding the design pattern from the source code of the java software.

![Flow diagram of Approach](image)

Here divided this approach into four stages.

First stage apply the reverse engineering process on our java source code. Apply the tool of reverse engineering process like Rational rose tool, StarUML tool which
has free available on internet. By using these tool, create the UML class diagram from the source code which is given in figure 2. And get the basic structure of the software. The UML Diagram of code given in figure 3. After getting the UML diagram of software, match the existing basic diagram of the Design pattern.

In UML Class diagram, classes are represented as boxes with the class name as a title and the association and dependency relationships shown by arrows going from one class to another. By applying the detection process obtained the following information to detect the design pattern. The results include class names, relationships between classes and attributes of the classes.

```java
class Abstraction {
    private Task task;
    public Abstraction(Task imp) {
        task = imp;
    }
    public void service1() {
        task.task1();
        task.task2();
    }
    public void service2() {
        task.task2();
        task.task3();
    }
    public void service3() {
        task.task1();
        task.task2();
        task.task3();
    }
    protected Task getImplementation() {
        return task;
    }
}

class ClientService1 extends Abstraction {
    public ClientService1(Task imp) {
        super(imp);
    }
    public void serviceA() {
        service1();
        service2();
    }
    public void serviceB() {
        service3();
    }
}

class ClientService2 extends Abstraction {
    public ClientService2(Task imp) {
        super(imp);
    }
```

Fig 2. Sample code

There are some disadvantages in the reverse engineering process like that the UML diagram does not give the runtime information about the objects. Sometimes its does not give the accurate structure of the software its may be miss the some relationship between the classes. So by recovering the accurate result followed the following steps.
In the Second stage using class file of the software to get the information about the classes and their relationship. For getting these relationships we use the JAVEX tool which take the class file as input and give the information about all the classes and their relationship. JAVEX tool are freely available on net. JAVEX is a fact extractor that extracts facts about all candidates or classes from the Java class files. These candidates are directories, source files, classes, interfaces, methods, variables, arrays, fields, constants, local variables and parameter variables. For each candidate JAVEX records all relations that occur in the source code, i.e. implements interface, extends class, instantiates, method invocations, containment (class contains classes, variables methods, etc.) and many more.

By combining the both result approximately guess that which design pattern is present in our software source code.
So we can find out the following static information is -

- uses for the relations.
- class extends class
- class implements interface
- class overrides method
- class references class
- class contains method
- class contains attribute

Third stage extracts the run-time information of the software source code that want to analyze. These dynamic facts represent the function calls that were executed during run-time. Explain how these dynamic facts are useful during the design pattern detection. Design patterns are not just explain as their UML class diagram, Their
dynamic behaviors are also very important and differentiate patterns that have an same or identical static structure, like the State and Strategy pattern. the dynamic behavior of objects get in terms of specific sequences of messages between objects with UML Sequence and collaboration diagrams. These diagrams consist of vertical lines which represent the objects during run-time and arrows that represent the messages from one object to another. example of the Adapter design pattern UML sequence diagram given in figure 4. Looking at the diagram it is easy to deduce the order in which messages are passed. First, the Client calls the Adapter object, and that one calls the Adaptee directly and passes the return value back to the Client.

Fig 4. Sample sequence diagram of adapter pattern

We use UML sequence diagrams to create our runtime definitions of design patterns. The UML specification for sequence diagrams allows very precise definitions of the sequential interaction of objects. So in the final step combine the result of the first, second and third stage and finally detect the used design pattern is use in the software.

4 Illustration

Here illustrate the some example code of bridge design pattern. 1st use the StarUML tool an apply reverse engineering process and create the UML diagram of code which is given in figure. The UML diagram is shown in figure.

After that we use the JAVEX tool, here pass the class file as a input. If we have the source code of the software then we need to compile the source first with javac. After that we find all class files and pass them as parameters to Javex. collect information about fields in classes as well as local variables with methods and arguments passed to these methods.
After that create the sequence diagram of that code and get the dynamic information of the object.
By this sequence diagram generate the runtime information of the source code, after that compare with the existing design pattern of the source code. And finally get the design pattern uses in the software.

5 Conclusion and Future Work

Proposing a approach to find a design pattern from the java application source code by applying reverse engineering process and getting the information about the candidates with the help of JAVEX tool and runtime information of the objects. Many methods have been proposed and implemented but still lack in flexibility and effectiveness.

For future work definitely see potential to improve the dynamic definitions. More detailed analysis of the GoF design patterns should allow better static and runtime information. Exploit all possibilities for the runtime information of the GoF design patterns. Believe that it will be useful to have multiple definitions of design patterns that do not have one common implementation.
References

1. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, ‘Design Patterns: Elements of Reusable Object-Oriented Software’, Addison Wesley, 1994