Design Patterns: A Resource for Reverse Engineering

P. Niranjan Reddy
Kakatiya Institute of Technology & Science, Warangal, INDIA

Jayadev Gyani
University of Hyderabad
INDIA

P.R.K. Murti
University of Hyderabad
INDIA

Abstract
Design patterns are gaining popularity because they support modifiability and flexibility of designs. Design patterns are solutions to frequently recurring problems in design. Reverse engineering of source code primarily focuses on the software architecture. Understanding software architecture in terms of design patterns simplifies the process of identifying some key properties such as coupling, flexibility and maintainability. This paper presents a novel approach to extract design patterns using structural metrics of object-oriented programs. It involves two steps. In the first step, structural metrics are extracted from the source code. In the second step, these metrics are matched with the properties of structural design patterns of Gang-of-Four to identify a design pattern. Our approach is demonstrated by extracting design patterns from a Java program using our pattern extraction tool.

Keywords: *Design pattern, extraction, structural metrics, matching*

1. Introduction

Design patterns are solutions to frequently recurring problems. Extracting design patterns from source code is useful in understanding the evolutionary nature of software. Software that is developed with design patterns is more maintainable. Antoniol et al.[1] identified a method for extracting design patterns from source code or design when relationships of classes are mapped to Abstract Object Language(AOL). Learning AOL is similar to any other learning process. So it consumes some time for learning. We eliminated this process by building structural information directly from source code. Their focus was on C++ source code. Giuseppe et al.[2] formulated a method for extracting interaction design patterns from web applications. Their approach was based on the frequency of a feature F in a web page.

2. Model of Pattern Extraction

We propose a method for extracting design patterns from source code, which is an improvement over other works. Our approach is implemented in two phases. In the first phase we extracted structural metrics from source code. These metrics are stored in a hash table. In second phase, aggregations and associations are identified and stored in two separate tables. The pattern extraction process model is shown in **Figure 1**. In ***aggregation*** relationship, a method delegation is used with a member object of a class. In ***association*** relationship, a method delegation is used with a class object on temporary basis. We extracted two structural design patterns from Java source code namely bridge and composite.

Ex1: Representation of Aggregation:

```
class Television
{  Button b1;
 void on_off()
 { b1.push ();  }      }
```

Ex2: Representation of Association:

```
class Compiler
{ void compile()
  { Scanner s = new Scanner();
            s.scan();
    }
 }
```

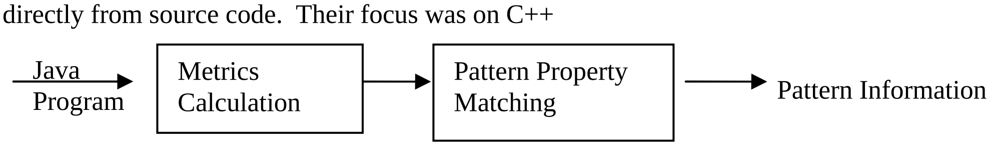

Figure 1 Pattern Extraction Process

Phase I: In the first step, structural information of classes is stored in a hash table. This information is used for building aggregation table and association table. The structural information of classes include number of super classes, names of super classes, number of subclasses, names of subclasses, method names of classes and names of interfaces a class is implementing. A snapshot of these metrics stored in a hash table for a sample program is shown in Table 1**.** The sample source program is based on the examples of Gamma et al.[8].

Phase II: Association and aggregation tables are formed with the metrics identified in the previous phase. These tables for the sample program are shown in Table 2 and Table 3 respectively. Every pattern can be stated as a set of elements with some relationships. In a formal way a pattern p can be represented as a graph <e, R> where e is the number of elements and R is a set of relations. If a relationship exists between a pair of elements then it must be a relation specified in set R. We focused on the structural design patterns because these patterns follow unique structural properties in design. We have eliminated Abstract Object Language representation specified by Antoniol et al.[1] and simplified the searching process by directly building a table for association and a table for aggregation. Every design pattern is bound by a set of constraints specified in terms of structural metrics. These constraints will vary from one pattern to another pattern.

Constraint verification algorithms for each of the three patterns are given below:

2.1 Algorithm for Bridge pattern extraction

```
For each row r in the aggregation table do
    If r[0] is an abstract class and r[1] is an interface then
        If r[0] has one or more subclasses and r[1] is implemented
                                by one or more classes then
            Display bridge detected
        Endif
    Endif
Endfor
```

2.2 Algorithm to detect Composite pattern

```
For each abstract class C which has at least two subclasses do
    If composite exists then
    //composite is identified as the class with ArrayList, Hashtable, LinkedList, Stack,
    // Vector or Dictionary as member or contains reference to parent class
    If number of subclasses of C is one more than the number of composite
            Classes then
            mark the subclasses which are not composite as leaves
            Display Composite pattern detected
        Endif
    Endif
Endfor
```

After verifying the constraints patterns are generated After verifying the constraints patterns are generated dynamically from the source code. If the source code is modified the corresponding patterns are affected. In the given sample code there are 2 design patterns namely one bridge and one composite pattern. The patterns which are generated from our tool are shown in results section .

3. Sample program

```
interface WindowImp
{
    final int x = 20;
    abstract void DevDrawText();
    abstract void DevDrawRect();
}
abstract class Window
{
    WindowImp k;
    abstract void DrawText();
```

```
    abstract void DrawRect();
}
class IconWindow extends Window
{
    int z;
    void DrawBorder()
    {
        int a;
        System.out.prinltn("testing") ;
    }
}
class TransientWindow extends Window
{
    int z;
    void DrawCloseBox()
    {
        int a;
        System.out.prinltn("testing") ;
    }
}
class XWindowImp implements WindowImp
{
    int k;
    void DevDrawText()
    {
        int a;
        System.out.prinltn("testing") ;
    }
}
class PMWindowImp implements WindowImp
{
    int k;
    void DevDrawText()
    {
        int a;
        System.out.prinltn("testing") ;
    }
}
abstract class MyComponent
{
    void operation()
    {
        System.out.println("component operation") ;
    }
    void add(MyComponent c)
    {
        System.out.println("add operation") ;
    }
    void remove()
    {
        System.out.println("remove operation") ;
    }
    void getChild(int n)
    {
        System.out.println("getchild operation") ;
    }
}
class Leaf extends MyComponent
{
    void operation()
    {
        System.out.println("Leaf operation") ;
    }
}
class Composite extends MyComponent
{
    ArrayList a;
    Composite()
    {
        a = new ArrayList() ;
    }
    void operation()
    {
        System.out.println("composite operation") ;
    }
    void add(MyComponent c)
    {
        System.out.println("composite add operation") ;
        a.add(c) ;
    }
    void remove()
    {
        System.out.println("composite remove
operation") ;
    }
    void geftChild(int n)
    {
        System.out.println("composite getchild
operation") ;
    }
}
```

4. Results

Table 1 : Structural Metrics

CLASSNAME	SUPERCLASS	SUBCLASSES	INTERFACES	METHODS
TransientWindow	Window			void DrawCloseBox()
IconWindow	Window			void DrawBorder()
PMWindowImp			WindowImp	void DevDrawText()
MyComponent		Leaf Composite		void operation(),void remove(),void add(MyComponent),void getChild(int)
Window		TransientWindow IconWindow		void DrawRect(),void DrawText()
Leaf	MyComponent			void operation()
Composite	MyComponent			void operation(),void remove(),void geftChild(int),Composite(),void add(MyComponent)
XWindowImp			WindowImp	void DevDrawText()

Table 2 : Association Table

no Associations found

Table 3 : Aggregation Table

CLASSNAME	C/I NAME
Window	WindowImp

C/I→ refers to Class/Interface

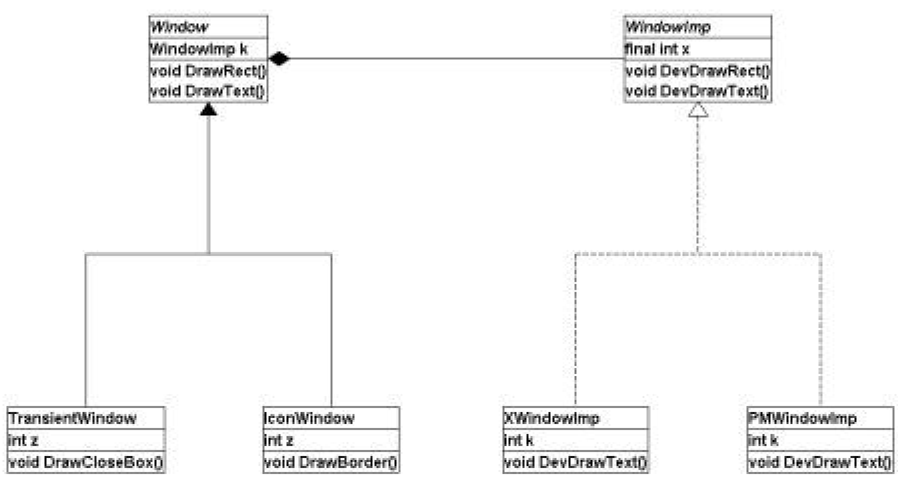

Figure 2 : Bridge Pattern Instance

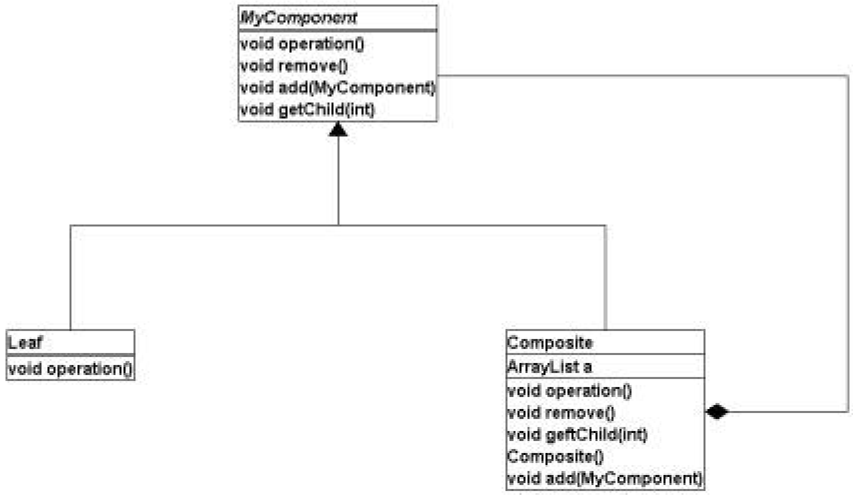

Figure 3: Composite Pattern Instance

5. Conclusions and Future Work

Design pattern extraction is essential in understanding the design of the software. Even though our example is simple it is sufficient to prove our concept. Currently we are working on extracting all the remaining GOF structural design patterns. Our approach simplifies the extraction process by eliminating intermediate code generation. Other design patterns will also be extracted using dynamic behavior of objects in our future work.

References

[1] G.Antoniol, R.Fiutem and L.Cristoforetti,*"Using Metrics to Identify Design paterns in Object-Oriented Software"*, Proceedings of the Fifth International Symposium on software (METRICS98), Nov, 1998

[2] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, Porfirio Tramontana,"*Recovering Interaction Design Patterns in Web Applications"*,Ninth European Conference on Software Maintenance and reengineering(CSMR'05),366-374,2005

[3] Herve Albin-Amiot, Pierre Cointe, Yann-Gael Gueheneuc, and Narendra Jussien,*"Instantiating and detecting design patterns: Putting bits and pieces together"*,Proceedings of 16th conference on Automated software Engineering, IEEE Computer Society Press, 2001, pg.166-173

[4] Asencio, A.; Cardman, S.; Harris, D.; Laderman, E.; "*Relating expectations to automatically recovered design patterns."* Reverse Engineering, Proceedings. Ninth Working Conference on , Page(s): 87 –96, 2002

[5] Kyle Brown, *"Design Reverse-engineering and Automated Design Pattern Detection in Smalltalk"*,1996

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "*Pattern-Oriented Software Architecture: A System of Patterns."* John Wiley & Sons, New York, April,1996

[7] Eide, E.; Reid, A.; Regehr, J.; Lepreau, J.; **"***Static and dynamic structure in design patterns"* Software Engineering. ICSE 2002. Proceedings of the 24rd International Conference on , Page(s): 208 –218, 2002

[8] E.Gamma, R. Johnson, and R. Helm, and J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software."* Addison-Wesley,1995.

[9] C.Kramer and L. Prechelt. "*Design recovery by automated search for structural design patterns in object-oriented software."* In Proc. Of the 3rd working Conference on Reverse Engineering(WCRE), Monterey,CA, page 208-215. IEEE Computer Society Press, November 1996.

[10] Paolo Tonella and Alessandra Potrich, *"Static and Dynamic C++ Code Analysis for the Recovery of the Object Diagram."* In Proc. of ICSM2002, International Conference on Software Maintenance, 2002.

[11] Winn, T.; Calder, P**.; "***Is this a pattern?"* IEEE Software, Volume: 19 Issue: 1 , Page(s): 59 –66, Jan/Feb 2002

[12] Antoniol, G., Fiutem, R. and Cristoforetti, L. "*Design pattern recovery in object-oriented software."* In 6th International Workshop on Program Comprehension, 153-160, Ischia, Italy, June 1998