Design Pattern Directed Clustering for Understanding Open Source Code*

Zhixiong Han, Linzhang Wang, Liqian Yu, Xin Chen, Jianhua Zhao and Xuandong Li
State Key Laboratory of Novel Software Technology
Department of Computer Science and Technology, Nanjing University
Nanjing, Jiangsu, P.R.China 210093
{hzx,yuliqian}@seg.nju.edu.cn, {lzwang,chenxin,zhaojh,lxd}@nju.edu.cn

Abstract

Program understanding plays an important role in the maintenance and reuse of open source code. Rapid evolving and bad documentation makes the understanding and reusing difficult. Design patterns are widely employed in the open source code. In this paper, we propose a design pattern directed clustering approach to help understand the structure of open source code. According to the approach, we have implemented a prototype tool. We also conducted an experiment on an open source system to evaluate it.

1. Introduction

As the modern software grows large and complex, open source becomes more and more popular nowadays. People contribute and reuse open source code in academia, industry and open source community. Open source code evolves quite rapidly, and usually lacks of documentation or the documents are out-of-date, which makes it hard to be reused and maintained. Program understanding plays an important role in aiding its reuse and maintenance.

In order to provide a high-level architectural view of source code, based on the top-down comprehension model, many software structure clustering approaches have been proposed, such as [3]. They aim at identifying areas of a system that are loosely coupled to each other according to the structural information in the code.

It is known that design patterns [1] are widely used in open source code. Design pattern information may help understanding the code. In this paper, we propose a new clustering approach according to design pattern relationships which are deemed to contain both structural and design information. The approach treats classes as players (i.e., participating classes) of design pattern instance (i.e., DPI [5]), and group them into different clusters according to the inherent association among the participating roles. We have implemented a prototype tool TasteJ[1] according to the approach and conducted an experiment on JHotDraw6.0[2] .

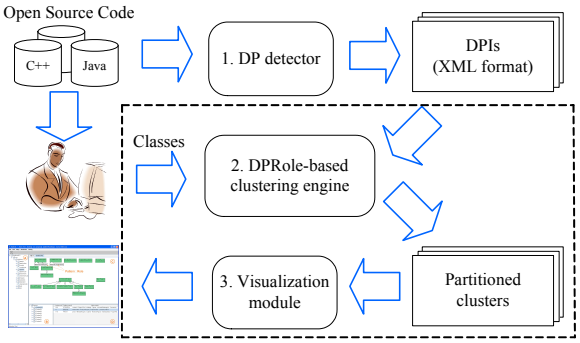

Figure 1. Overview of the approach

2. Approach

The overview of our approach is shown in Figure 1. It consists of three main components: the design pattern detector (DP detector), the design-pattern-role-based (DPRole-based) clustering engine and the visualization module. The DP detector [4] accepts open source code and generates recovered DPIs. The DPRole-based clustering engine, which is the core component of the approach, needs two inputs, one is user's concerning classes, e.g., the classes in a package or directory, the other is classes' related DPIs which are recovered by the DP detector. The engine treats classes as players of the recovered DPIs and groups them into clusters based on *close-role sets* [2], which is defined by pattern rules to indicate the high degree association among roles. Table 1 shows the default pattern rules

*This work is supported by the National Natural Science Foundation of China (No.90818022), the National 863 High-Tech Programme of China (No.2009AA01Z148), and by the Jiangsu Province Research Foundation (No.BK2007714).

[1] http://seg.nju.edu.cn/tastej
[2] http://www.jhotdraw.org

Table 1. Pattern rules and close-role sets

Pattern Rule	DPI	Close-role set(s)	Close-player set(s)
Bridge[1]	DPI(abstraction: ClassA; refindAbstraction: SetA; implementor: ClassI; concreteImplementor: SetI)	{abstraction, refindAbstraction, implementor, concreteImplementor}	{ClassA, SetA, InterfaceI, SetI}
Composite[1]	DPI(component: ClassC1; composite: ClassC2; leaf: SetL; client: ClassC3)	{component, composite}	{ClassC1, ClassC2}
Adapter[1]	DPI(adapter: ClassA1; adaptee: ClassA2; target: ClassT; client: ClassC)	{adapter, adaptee, target}	{ClassA1, ClassA2, ClassT}
Proxy[1]	DPI(proxy: ClassP; subject: ClassS; realSubject: SetS)	{proxy, subject, realSubject}	{ClassP, ClassS, SetS}
Template Method[1]	DPI(abstractClass: ClassC1; concreteClass: ClassC2)	{abstractClass, ConcreteClass}	{ClassC1, ClassC2}
Visitor[1]	DPI(visitor: ClassV; element: ClassE; concreteVisitor: SetV; concreteElement: SetE; objectStructrue: ClassS)	{visitor, concreteVisitor} {element, concreteElement}	{ClassV, SetV} {ClassE, SetE}
Strategy[2]	DPI(context: ClassC; strategy: ClassS; concreteStrategy: SetS)	{strategy, concreteStrategy}	{ClassS, SetS}
State[2]	DPI(context: ClassC; state: ClassS; concreteState: SetS)	{state, concreteState}	{ClassS, SetS}

[1] Rules on structure-driven patterns [2] Rules on behavior-driven patterns

and related close-role sets of several patterns. We know that a class may play roles in different DPIs, we form a compact cluster by merging the clusters that contain a common player. In addition, as Cluster 6 through 8 in Figure 2, each of which contains only a class, we use inheritance and sibling relationships to conglutinate them with other clusters. The visualization module visualizes the resulting clusters.

3. Preliminary results

We perform preliminary experiments on the packages of JHotDraw6.0. Figure 2 shows clustering results of the package `figures` which is one of JHotDraw6.0's big-size packages. The results give us a clear division about the package's internal members. Cluster 1 seems to comprise too much members, whereas its contained members apparently have the same theme, i.e., "figure". Similar to Cluster 1, Cluster 2, 3, 4 and 5 also have their respective themes, i.e., "handle", "tool", "connector" and "command". It is worth noting that we did not use any lexical matching skills in the clustering process. The clustering results of other packages are described in [2]. Moreover, we compared the results with those produced by the tool Bunch [3]. The comparison shows that our clustering results are more recognizable.

4. Conclusion and future work

We have proposed a novel design pattern directed clustering approach and developed a prototype tool to assist open source code comprehension. Moreover, we conducted a case study on an open source system JHotDraw6.0 to evaluate the approach. The preliminary results show that it is

Cluster 1: AttributeFigure, BorderDecorator, ElbowConnection, EllipseFigure, GroupFigure, ImageFigure, LineConnection, LineFigure, NullFigure, NumberTextFigure, PolyLineFigure, RectangleFigure, RoundRectangleFigure, TextFigure, LineDecoration, FigureAttributes;

Cluster 2: ElbowHandle, FontSizeHandle, GroupHandle, PolyLineHandle, RadiusHandle;

Cluster 3: BorderTool, ConnectedTextTool, ScribbleTool, TextTool;

Cluster 4: ChopEllipseConnector, NullConnector, PolyLineConnector, ShortestDistanceConnector;

Cluster 5: GroupCommand, InsertImageCommand, UngroupCommand;

Cluster 6 through 8: AbstractLineDecoration; ArrowTip; PolyLineLocator.

Figure 2. Preliminary clustering results

feasible and promising to generate easily understandable results. So far, the approach is applied to cluster package's classes. In ongoing work, we will improve the implementation to suit an entire system and conduct more experiments.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements Of Reusable Object-Oriented Software*. Addison-Wesley Reading, MA, 1995.

[2] Z. Han, L. Wang, L. Yu, X. Chen, J. Zhao, and X. Li. Design pattern directed clustering for understanding open source code. Technical Report 2009-01, SEG-NJU, China, 2009.

[3] S. Mancoridis, B. Mitchell, C. Rorres, Y. Chen, and E. Gansner. Using automatic clustering to produce high-level system organizations of source code. In *Proc. of 6th IWPC*.

[4] N. Shi and R. Olsson. Reverse engineering of design patterns from java source code. In *Proc. of 21st ASE*, 2006.

[5] L. Wendehals. Improving design pattern instance recognition by dynamic analysis. In *WODA*, 2003.