Automated Extract Component Refactoring

Hironori Washizaki and Yoshiaki Fukazawa

Department of Computer Science, Waseda University
3-4-1, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan
{washi,fukazawa}@fuka.info.waseda.ac.jp

Abstract. We propose a new refactoring "Extract Component" to support the organizational reuse of components and improve the productivity under Agile methods. Our refactoring can extract reusable components composed of classes from object-oriented programs, and modify the surrounding parts of extracted components in original programs. We have developed a tool that performs our refactoring automatically.

1 Introduction

Agile methods such as Extreme Programming (XP)[1] mainly aim to make the current project succeed, rather than improving the productivity of the entire organization. We limit this study to the use of OO language for the implementation of programs. To improve productivity, the organization has to engage in software reuse. Refactoring, which is a XP's practice, can help make software reusable. We propose a technique for transforming part of the OO classes into reusable software components automatically. In the following, we provide a strict definition of component. JavaBeans[2] is the target component system.

我们提出了一种将部分面向对象类自动转换为可复用软件组件的技术。

Definition 1.1 (Component) A component is composed of one or more Java classes that satisfy all the following requirements ($1 \sim 5$).

1. One of the classes is a Facade class, which becomes a front class for outside.
2. The Facade class implements a certain interface (the Facade interface).
3. The Facade class has one public default constructor (without any arguments).
4. All classes/interfaces necessary for instantiating an object of the Facade class are packaged into one JAR (Java ARchive) file.
5. Objects of all classes except the Facade class are not directly instantiated from outside of the classes.

The component which satisfies requirements $1 \sim 5$ has no dependency on elements outside of the component. Therefore, it is possible to reuse the classes in the form of the component. In order to obtain such components, it is necessary to transform parts of existing programs into components. We first define a class relation graph (CRG). CRG represents the relation of classes/interfaces in the target program. Next, we define a component cluster on CRG.

CRG表示目标程序中类/接口的关系。

M. Marchesi and G. Succi (Eds.): XP 2003, LNCS 2675, pp. 328–330, 2003.

类关系图

Definition 1.2 (Class Relation Graph) The multi-graph is a class relation graph, denoted $\Gamma = (V_\Gamma, \Lambda_\Gamma, E_\Gamma)$, where V_Γ is the set of class/interface nodes corresponding to classes/interfaces, Λ_Γ is the set of label names that are composed of the names of program source code and the target token locations in program source code, and E_Γ is the set of edges that are ordered trios of the source node, destination node, and label name. E_Γ is composed of the inheritance edges, refer edges, default instantiate edges, and argument instantiate edges.

The inheritance edge indicates that the class/interface inherits from another class/interface, denoted $\multimap\!\triangleright$. The refer edge indicates that the class/interface refers to another class/interface, denoted $\overset{ref}{\rightarrow}$. The default instantiate edge indicates that the class/interface instantiates an object of another class by using the default constructor, denoted $\overset{def}{\Rightarrow}$. The argument instantiate edge indicates that the class/interface instantiates an object of another class by using the constructor with one or more arguments, denoted $\overset{arg}{\Rightarrow}$.

Definition 1.3 (Reachability) The node v_i satisfies the inheritance reachability with the node v_j where there is a path from v_i to v_j and all edges in the path are inheritance edges, or $v_i = v_j$, denoted $v_i \overset{*}{\multimap\!\triangleright} v_j$. Similarly, $v_i \overset{*}{\rightarrow} v_j$ means that there is a path from v_i to v_j and all edges in the path are inheritance/instantiate/refer edges, or $v_i = v_j$.

Definition 1.4 (Component Cluster) The pair of the Facade node v_f and the set of nodes V_f is a cluster where CRG is $\Gamma = (V_\Gamma, \Lambda_\Gamma, E_\Gamma)$, V_f is the superset of $V' = \{v' \mid (v_s \overset{*}{\rightarrow} v') \text{ in } \Gamma\}$ for the starting node v_s, one node v_f has incoming default instantiate edges from nodes in $V_\Gamma - V_f$, and no nodes in V_f except v_f have incoming default/argument instantiate edges from nodes in $V_\Gamma - V_f$. The cluster $c = (v_f, V_f)$ is the candidate of the component that is composed of all classes/interfaces in V_f, and the Facade class of the component is v_f. The component corresponding to the cluster satisfies the above-mentioned requirements.

2 Extract Component Refactoring

When extracting components corresponding to specified clusters, the surrounding parts of clusters should use newly extracted components in order to avoid the situation where two sets of classes that provide the same function exist in the same organization/library. This modification is a kind of refactoring because this modification does not change the observable behavior of the original program. We call this refactoring, "Extract Component," and the necessary steps of this refactoring are shown below.

1) Make CRG $\Gamma = (V_\Gamma, \Lambda_\Gamma, E_\Gamma)$ of the target classes.
2) Select a class v_s that is thought to provide general/reusable functions.
3) Specify one cluster $c = (v_f, V_f)$ using v_s as the starting node.
4) Create a new Facade interface i_f. Implement i_f to the Facade class v_f.
5) Add the definitions of all public methods of the inheritance reachable classes $V_E = \{v \mid (v_f \overset{*}{\multimap\!\triangleright} v) \text{ in } \Gamma\}$ to i_f.

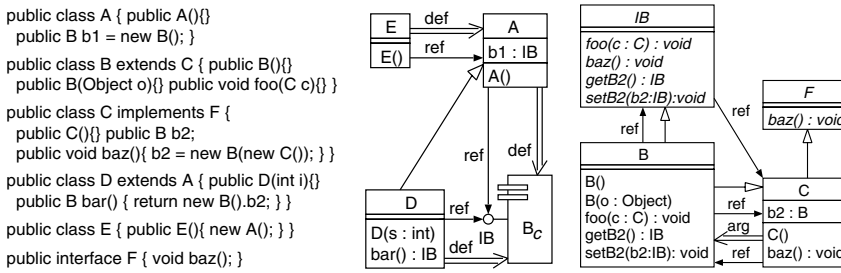

Fig. 1. (a) Code before refactoring, (b) CRG after refactoring, (c) Inside of B_c

6) Add the definitions of the setter methods and getter methods corresponding to all public attributes of V_E to i_f. Add the implementations of the setter methods and getter methods to v_f.
7) Package all classes/interfaces in V_f and i_f into one JAR file.
8) Change the types of reference to $v_f$in $V_\Gamma - V_f$ into i_f.
9) Change the code in $V_\Gamma - V_f$ which assign new values to attributes of V_E (or refer the values of attributes of V_E) into the code which invoke the setter methods (getter methods).
10) Insert the explicit reference conversion code, from the class/interface in V_E to i_f, into the part in $V_\Gamma - V_f$ where the implicit widening reference conversions from v_f to another class/interface in V_E exist.

For example, Figure 1(b) shows CRG after detecting the cluster $c = (\mathrm{B}, \{\mathrm{B}, \mathrm{C}, F\})$ using the class C as the starting node corresponding to the program source code in Figure 1(a), and extracting a JavaBeans component B_c according to the cluster c by Extract Component refactoring. Figure 1(c) shows the inside of B_c. B_c has a new Facade interface IB and new setter/getter methods. The surrounding parts of the cluster are modified to use B_c via IB.

3 Automatic Tool and Conclusion

We have developed a tool that analyzes Java program source code, displays CRG, and performs automatically all necessary steps of the Extract Component refactoring. Using our tool, we extracted 13 components from KFC [3] (number of all classes/interfaces: 224) using all classes in KFC as starting nodes. Many of them are components with high generality, such as Button, Panel, DTD and Border. These components can be reused independently in other projects.

In conclusion, since programmers can make a component library using extracted components from a developed classes without almost any costs, our tool will support the organizational reuse of components based on Agile methods.

References

1. Beck, K.: Extreme Programming Explained, Addison-Wesley (1999)
2. Hamilton, G.: JavaBeans 1.01 Specification, Sun Microsystems (1997)
3. Yasumatsu, K.: KFC, http://openlab.jp/kyasu/