RESEARCH ON SUPPORT TOOLS FOR OBJECT-ORIENTED SOFTWARE REENGINEERING

Xin Peng, Wenyun Zhao, Yijian Wu, Yunjiao Xue
Computer Science and Engineering Department, Fudan University, Shanghai, China

Keywords: Software Reengineering, CASE Tools, Reverse Engineering, Component.

Abstract: Reengineering presents a practical and feasible approach to transform legacy systems into evolvable systems. Component-based systems are evolvable and can be easily reengineered. Object-oriented software reengineering should base on component library and focus on seamlessly cooperating with component library and assembly tool to construct the whole reengineering system. So the reengineering discussed here concentrates on extracting components from legacy systems via comprehension and analysis. In this paper, we present our java-based tool prototype FDReengineer and introduce the component extraction algorithm. The method and the advantage is demonstrated through a case study.

1 INTRODUCTION

Object-oriented method has been the mainstream of software development. However, lack of experience of OO design and immaturity of OO technology during its development will make the existing systems lack evolvability and such systems cannot be easily adapted to meet the variety of requirements by maintenance. Reengineering presents a practical and feasible approach to transform legacy systems into evolvable systems (Yao, G. et Al.).

The increasing complexity of today's legacy systems demands for automated tool support and some commercial and academic tools are available, such as JBPAS (Xie, T. et Al.), FAMOOS (Bär, H. et Al.), McCabe IQ, etc.

Along with the development of CBSD (Component-Based Software Development), reengineering changes to focus on reconstructing existing object-oriented systems to component-based applications. Accordingly, the tool should also support this orientation and be integrated seamlessly with the component library. This paper presents a package-spreading based component extraction algorithm and the prototype is also introduced.

The paper is organized as follows: Sect. 2 introduces our prototype tool FDReengineer. Sect. 3 presents in detail the method we used in system partition and component extraction. Sect. 4 demonstrates the advantage by contrasting our method with other ones in a case study. Finally Sect. 5 draws our conclusions.

2 FDREENGINEER

FDReengineer is the prototype of our tool, the kernel task of which is to extract, filter, encapsulate and submit components according to the concrete reengineering requirements. In the forward process, it cooperates with enterprise component libraries and assembly platforms (Figure 1). Here are some features of FDReengineer:

- UML diagrams generation: Two kinds of class diagrams, namely global view and local view are supported. The global view presents structure of the whole system, while the local view concentrates on single class by presenting the inner structure of the class and the relations with other classes.
- Metrics: Some general metrics, such as length of codes, depth of inheritance, are provided. Especially for measurement of the tightness of a package we define cohesion as below:

(1) $R_d(P) = \{(a,b) \mid a \in P \wedge b \in P \wedge (a \quad refers \quad b)\}$

$$TCC_d(P) = \frac{|R_d(P)|}{|P| \times (|P|-1)}$$

(2) $R_i(P) = R_d^*(P)$

$$TCC_i(P) = \frac{|R_i(P)|}{|P| \times (|P|-1)}$$

Here P is a package of classes (the package here is not the term in Java, it only presents a set of classes), (1) is the direct cohesion of package P, while (2) is the indirect cohesion of package P.

Peng X., Zhao W., Wu Y. and Xue Y. (2005).
RESEARCH ON SUPPORT TOOLS FOR OBJECT-ORIENTED SOFTWARE REENGINEERING.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 399-402
DOI: 10.5220/0002513703990402

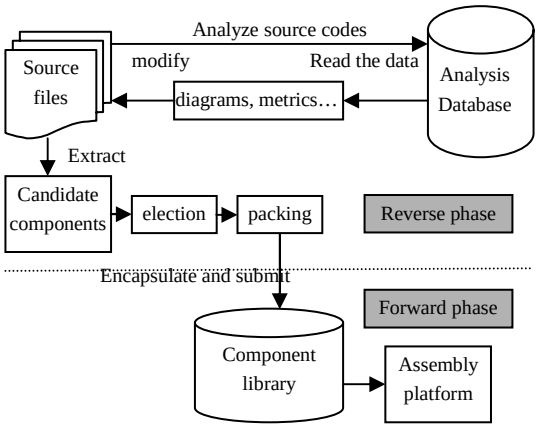

Figure 1: Functional model of FDReengineer

3 COMPONENT EXTRACTION

According to the RE2 (Reverse Engineering and Reuse Reengineering), a reuse reengineering process is modeled by five sequential phases: Candidature, Election, Qualification, Classification and Storing, Search and Display (Canfora, G. et al., 1955). This is a clear reuse reengineering process aiming at reuse reengineering of structural programs.

In object-oriented programs, classes are atomic units. So we should aim at composing reusable components from atomic classes by analyzing relations between them. Components of higher abstract levels can be composed of lower ones.

Class diagrams present the bottom structure of legacy systems. In order to extract reusable components for the forward engineering phase, the tool must support the promotion of abstract level and provide designs at a higher abstract level.

3.1 The process of extraction

In our method java classes are the atomic units of partition. The initial partition object is the whole system, including all the classes in it. At first the whole system is divided into several subsystems according to relations between classes. That is the partition at the first abstract level. Then similar operations can be done within each subsystem. When performing partition within a subsystem, only relations between classes of the current subsystem will be taken into account.

Blocks of appropriate granularity will be available when partition reaches certain abstract level. The block here comprises some close related classes and can be chosen as a candidate component if certain functions are well encapsulated in it. So candidate components are the results of certain phase of partition. Granularities of candidate components can vary a lot depending on different business functions and aims of acquirement (for public reuse or only for the reuse in reconstruction). Generally, the granularity of business components for single domain is relatively coarse, while components for public reuse are finer.

3.2 The spread algorithm

Some methodologies for system partition have been proposed, such as RBCI (Xin, et al.) and cluster algorithm (Xu W., et al., 2003). Both of them treat relations between classes as undirected ones, while we know these relations, such as generalization, dependency, aggregation etc, are all directed. On the other hand, they view the relations as separate ones, which only exist between two classes. The complex relations among a set of classes are not taken into account.

The closure method is also a partition algorithm. The closure here is a kind of transitive closure, which is a recursively defined unit including all the classes related directly or indirectly by each class in it. The closure method is a rough way of component extraction and a closure is often too unwieldy to turn into a component.

In view of these shortcomings, we proposed our spread algorithm. The improved method, which we call the spread algorithm, is based on further analysis of the reference relations. The spread expands a package by performing spread on the network of relations. Notice that the “package” is just an organization unit here, which is not Java package. The concrete algorithm is given as follow:

Step 1: The initial package is a collection of entry classes (one or several).

Step 2: Pick out a class C from current package, consider all the classes directly referenced by C and determine whether a class should be added into the package one by one by means of specific algorithms.

Step 3: Perform step 2 until each class of current package is considered and the package does not expand any more, then the spread stops.

Step 4: The one on which the last spread stopped becomes new entry class if it is not covered by any of the existing packages

Step 5: Repeat step 1-4 till each of the classes take part in the partition is classified to certain package.

Entry classes are the start of spreading. Initially the entry is often the class which has the largest closure, e.g. the main class of whole system. More entry classes may be needed if the system has several independent branches. That ensures that all the classes of the system can be reached from at least one entry class via a reference chain.

During partition the evaluation criterion mainly represents how closely the class under consideration and the current package are related. For a class C and a package P: Let,

$$RCP = \sum_{i \in P} RCi, RCN = \sum_{i \in N} RCi,$$

$$refCP = |\{A | (A \quad refers \quad C) \wedge ((A \quad refers \quad P) \vee A \in P)\}|,$$

$$refC = |\{A | A \quad refers \quad C\}|,$$

and N is the set of all the classes, RCi is the relating degree between class i and C, then our criteria combines three measurements below:

$$1.\frac{refCP}{refC} \quad 2.\frac{RCP}{RCN} \quad 3.\frac{|N|-refC}{|N|}$$

Refer operator represents directed relations, such as generalization, dependency, aggregation etc. In these factors, RCP represents the summation of relating degrees between C and P; RCN represents the summation of relating degrees between C and N; refCP represents the number of the classes which refer both class C and package P; refC represents the number of the classes which refer both class C. Calculation of the relating degree between two classes are discussed in (Xin, et al.) and (Xu W., et al., 2003). For example we can assume that: generalization is 3, aggregation is 2 and dependency is 1. Here we consider directed relations, that is, only the situations that C is referred by other classes are taken into account when considering whether C should be added into P.

These factors represent different aspects of the evaluation. The first one reflects the probability that class C and package P are used together. The second one shows to which extents C particularly serves P. The third one describes the degree of class C as a common function provider. The more widely a class is used, the fewer score it can get from factor 3.

Each time when evaluating whether class C should be added into package P, the tool calculates the score got from these three factors and compares it with the criterion specified to make the decision. FDReengineer provides a default criterion according to the level of each partition. Additionally, the user can adjust the criterion at will, cancel the last partition and redo it.

After this kind of partition, the legacy system is divided into several parts. Some of them are close organized subsystems, and the classes providing common functions are picked out separately. The resulting packages can be reused in the forward phase. Some common ones can still be valuable in other domains.

In the closure method all the classes under consideration will be added into the package. In fact, the closure method is a special case of the spread method in condition of setting the criterion to 0. The improved method has two advantages:

1) It can determine which package the class falls into by means of specific algorithms in the case of junction classes.

2) The classes which provide common functions can be extracted separately.

Figure 2 shows an example of system partition. If partition begins from class B, the resulting package will be {B, E, F, I, J, K}. Class J and H will repeat in several packages, since they provide service for many classes.

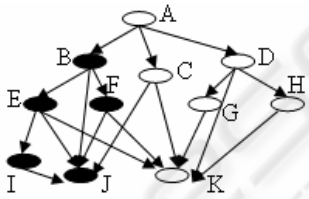

Figure 2: An example of system partition

Let we redo the partition by the spread method. Suppose that the three factors are combined with the scale of 3:1:1 and the criterion is 70 while the full score is 100. So the full score of the three factors are 60, 20, and 20 respectively. We perform the partition from B, so the initial package is {B}. Here are the steps:

1) Consider the classes referred by B. The scores of E and F are: 60+20+18.2=92.2, which is above 70, so they are added into the package. Now the package is {B, E, F}. The score of J is: 36+12+10.9=58.9, so it should not be added here.

2) Consider I, J and K, which are referred by E. The score of I is: 60+20+18.2=92.2, so it is added into the package. Class J is considered again. Its score is: 48+16+10.9=74.9, and it can be added into current package this time. Now the package is {B, E, F, I, J}. Class K will not be added because its score is: 30+6.7+9.1=45.8.

3) All the classes referred by the classes in the current package are considered, and no more classes will be added. Therefore, this partition ends in a package {B, E, F, I, J}.

Furthermore, class K will not fall into any package of other classes. We will get a package only containing K. This is what we expect, since K provides service for many subsystems. Thus it can be seen that this method can get a better effect.

The first partition of the legacy system produces some subsystems, then partitions can be performed on each subsystem using the same method**.** When doing this, only the references between classes of the subsystem will be taken into account. The right criterion varies with the abstract level. According to our experience, the partition at upper levels should be loose to produce subsystems of coarse

granularity, and tighten when the abstract level lowers to get blocks of fine granularity.

3.3 Advanced partition strategies

The partition algorithm described in section 3.2 is somewhat deficient in practical application. Some advanced partition strategies are needed:

breakpoints setting: Packages gained from the root probably contains entry classes of some subsystems which are only referred by the root class. That is disadvantageous if we want to acquire the distinct partition of subsystems in vertical direction, because these subsystems will not be separated from the root by our algorithm. A breakpoint indicated that the former spread will stop here, then it will become the start of a new spread. In this way we can get the subsystems in vertical direction.

UI separation: Another problem is that when we concentrate on the extraction of business components we must specially separate UI classes from business classes. UI separation can make the spread stop between UI classes and business classes. Identification of UI class can be automated to some extent, e.g. classes inheriting Java UI classes may be identified as UI classes.

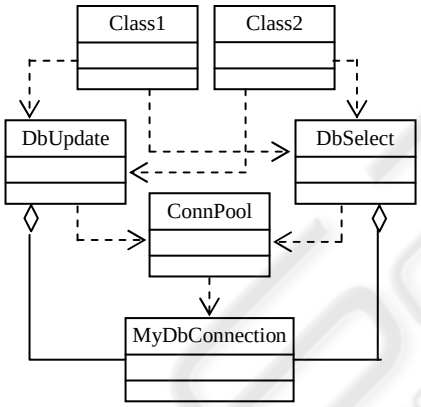

Figure 3: A segment of a store management system

4 CASE STUDY

Figure 3 demonstrates a part of a store management system, in which class DbUpdate provides functions of database operation and DbSelect provides functions of database query. ConnPool is a simple connection pool. MyDbConnection is a wrapping class of database connection. It is explicit that these four classes compose a component providing database service, other classes use DbUpdate and DbSelect to perform database operations, such as Class1 and Class2. So the perfect partition result is packing these four classes together.

ConnPool only has dependency relation with the other three classes, which is the weakest relation. If we perform partition according to the RBCI algorithm in (Xin, et al.) we can not achieve the goal. If we adopt the spread algorithm we can obtain a more perfect result package consisting of four database operation classes.

5 CONCLUSION

CBSD presents a new framework for reengineering. Reengineering tools should focus on producing well-encapsulated reusable components under the direction of the user. FDReengineer has gained a good effect on component extraction. However, how to determine the criterion and the scale to combine the evaluation factors still needs further study. We will further our research on these aspects.

REFERENCES

Guo Yao, Yuan Wanghong, Chen Xiangkui, Zhou xin. Reengineering: Concepts and Framework [Electronic version]. *Computer Science (China)*, 26, 78-83.

Tao XIE, Wanghong YUAN, Hong MEI, Fuqing YANG. JBOOMT: Jade Bird Object-Oriented Metrics Tool. Retrieved from http://www.cs.washington.edu/homes/taoxie.

Holger Bär , Markus Bauer , Oliver Ciupke, etc. The FAMOOS Object-Oriented Reengineering Handbook. Retrieved from http://dis.sema.es/projects/FAMOOS/.

Gerardo Canfora, Anna Rita Fasolino, Maria Tortorella. Towards Reengineering in Reuse Reengineering Processes. Proceedings of International Conference on Software Maintenance, 1995, 147-156.

ZHOU Xin, CHEN Xiang-kui, SUN Jia-su, YANG Fu-qing. (2003). Software Measurement Based Reusable Component Extraction in Object-Oriented System [Electronic version]. *ACTA ELECTRONICA SINICA (China)*, 5, 649-635.

Xu W, Yin BL, Li ZY. (2003). Research on the business component design of enterprise information system [Electronic version]. *Journal of Software (China)*, 7, 1213-1220.