Proc. of the 3rd International Software Architecture Workshop (ISAW), 1998, pp.101-104

Software Architecture Recovery and Restructuring through Clustering Techniques

Chung-Horng Lung
Software Engineering Analysis Lab, Nortel
P.O. Box 3511 Station C, Ottawa, Ontario K1Y 4H7, Canada
phone: (613) 765-4820,
email: lung@nortel.ca

1. ABSTRACT

Capturing of software architecture is critical for maintenance and evolution. However, existing approaches often are limited only to software architecture recovery in the reverse engineering process. What is needed more is a systematic and effective approach to help the designer to restructure or reengineer an architecture for improvement. This paper presents an quantitative approach based on clustering techniques for software architecture restructuring and reengineering as well as for software architecture recovery. Clustering techniques are built on top of reverse engineering tools. The approach has been applied to several examples at various levels of abstraction. Two case studies are presented in this paper. One is an empirical study of a decoupling effort for a real-time telecommunications system. The other example shows a research potential to enforce the designer to improve an architecture by adopting a design pattern based on the clustering results.

1.1 Keywords

Architecture recovery, restructuring, evolution, clustering, patterns

2. INTRODUCTION

Software architecture is a critical asset to a project due to the ever increasing complexity and the demand to reduce maintenance cost for evolution. One of the areas in software architecture is architecture recovery through reverse engineering of existing implementations. Numerous articles have been published on this topic. More recent papers include [2,7]. There is little doubt that software architecture capture or recovery is important. However, most existing approaches often are limited to this particularly activity in the reverse engineering process only. What is needed much more is an approach that can systematically and effectively support restructuring or reengineering to improve the architecture based on the stakeholders concerns.

Clustering techniques have been used in many disciplines to support grouping of similar objects of a system. Clustering analysis is one of the most fundamental techniques adopted in science and engineering. The primary objective of clustering analysis is to facilitate better understanding of the observations and the subsequent construction of complex knowledge structure from features and object clusters. Examples include botanic species and mechanical parts. The key concept of clustering is to group similar things into clusters, such that intra-cluster similarity or cohesion is high, and inter-cluster similar or coupling is low. Coupling has great impact on many quality attributes, such as maintainability, verifiability, flexibility, portability, reusability, interoperability, and expandibility [4]. The main objective of clustering is similar to that of software partitioning.

Clustering techniques can be applied to software during various life-cycle phases. Lung [8] demonstrated that clustering techniques can be used to effectively support both software architecture partitioning at the early phase in the forward engineering process and software architecture recovery of legacy systems in the reverse engineering process. The objective of this paper is to take one step further by showing more added values. In other words, the paper shows that the clustering techniques can also be used to effectively facilitate software architecture restructuring instead of simply being used for software architecture recovery of existing systems.

This paper is organized as follows. Section 2 briefly explains the basic clustering concept and the clustering technique adopted in this research. Section 3 presents two examples of how the clustering technique is used to support software architecture restructuring to minimize coupling. One is an empirical study of a legacy system, the other one is an initiative idea of using clustering techniques to support the adoption of a design pattern. Both examples demonstrate significant improvement from the coupling perspective. Finally, Section 4 is the summary of this paper.

3. CLUSTERING TECHNIQUES

Many clustering algorithms have been presented, but they

comprise the following three common key steps:

- Obtain the data set.
- Compute the resemblance coefficients for the data set.
- Execute the clustering method.

An input data set is an object-attribute data matrix. Objects are the entities that we want to group based on their similarities. Attributes are the properties of the objects. A resemblance coefficient for a given pair of objects shows the degree of similarity or dissimilarity between these two objects, depending on the way the data represents.

Several algorithms have been adopted and slightly modified to calculate resemblance coefficients. The example presented in this paper are based on the Sorenson algorithm. Given a resemblance matrix, a clustering method is then used to group similar objects. A commonly used clustering algorithm called UPGMA (unweighted pair-group method using arithmetic averages) is used to cluster the objects. The algorithms are described in detail in [9].

To apply the clustering techniques to software architecture recovery and reengineering, the object-attribute data matrix is modified to object-object data matrix, so that the input reflects the interconnectivity of components. The clustering techniques are then used to minimize interconnections among components [8].

4. APPLICATIONS OF CLUSTERIING TECHNIQUES IN SOFTWARE ARCHIEC-TURE REENGINEERING

Two examples are presented in this section. The first one is a result of a decoupling effort of a legacy system. The second example shows an application of the clustering technique to support the identification of a design pattern.

4.1 Empirical Study

Given the fact that software complexity is usually high and the need for software maintenance is inevitable, the challenge is how to effectively support software evolution. This study illustrates how the combination of use cases and clustering techniques help us restructure a system. The new system has much lower coupling index and is much easier to maintain.

The example belongs to one of the major subsystems of a real-time telecommunications application. The example project comprises about 80,000 lines of C++ code. We used a third party reverse engineering tool to identify the dependency or interconnection relationships among classes and applied the clustering technique to the project to validate the approach for software architecture recovery. The result was encouraging. About 80% of the components at the class level corresponds the modules partitioned by the subject experts. There are many possible reasons that cause the discrepancies. Based on our observations on several projects, the main reasons include:

- Not all interconnections are considered or identified, because boundaries are needed.
- Reverse engineering tools may not generate precise and complete information.
- Some components may have much more connections than other components, e.g., the components that serve as APIs or shared resources.
- Some modules may perform a specific task and the number of components in these modules are significantly less than other modules.
- The system may not be well partitioned by the designer.
- The limitations of the clustering techniques.

The main objective of this study is to build on top of the

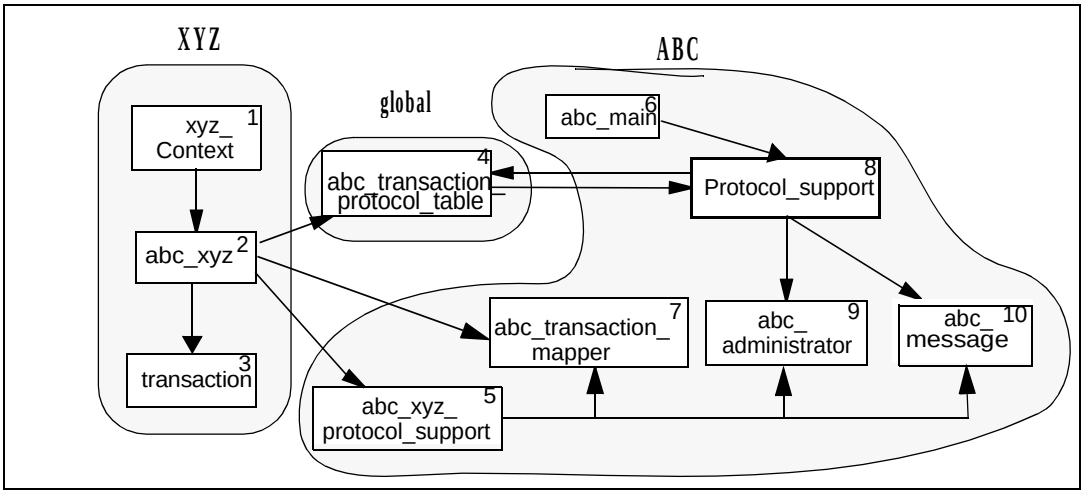


Figure 1. Initialization and Registration of Protocol Handlers - Existing Design

recovery process by decoupling some areas to improve modularity or support evolution through the clustering technique. We reduce the complexity by working on smaller pieces known as use cases or scenarios. Firstly, the critical use cases are identified and the software architecture are evaluated with the use cases. Figure 1 shows only relevant components and connections for a use case of the existing design, which is the initialization and registration of protocol handlers. There are a global table and two sub-systems, XYZ and ABC. For propriety reasons, the prefix of the components are changed, but the names are consistent with the real names.

First of all, the approach using file names to clustering the components does not work well for this example as discussed in section 3. More importantly, even if most of the component names in ABC start with abc, but the grouping of these components does not help much in terms of restructuring or decoupling. On the other hand, the result of the clustering analysis based on component interconnectivity reveals two different groups (5, 7, 2, 1, 3) and (9, 10, 6, 8, 4). Figure 2 shows the clustering tree or the dendrogram notation. This diagram reveals that components 5 (abc_xyz_protocol_support) and 7 (abc_transaction_mapper) are more related to subsystem XYZ than ABC, which is quite different from the design concept. The difference leads us to conduct further analysis on the coupling issues of these two components and both subsystems, ABC and XYZ between the existing design and the clustering results.

By examining the implementation based on the difference, we identified the areas that are highly coupled. We then worked with designers to validate those areas and capture the semantics behind them. Finally, we moved certain methods or routines from one class to another. Figure 3 demonstrates the new design. For this example, the changes included the move of an initialization method from class 2 to class 6, removal of classes 3 and 4 (w.r.t. this specific use case), and addition of a method call from class 8 to class 7. The coupling is significantly reduced. Two metrics were used as measurements. Data complexity [3] is reduced from 9 to 1, while the strength [6] is improved from 0.16 to 0.4. In addition, the number of operations is also reduced from 12 to 10. If this use case deals with a critical execution path, there could have tremendous performance gains. The new design also provides flexibility to dynamically link new protocol handlers. Both designs perform the exact same functions, but they look significantly different.

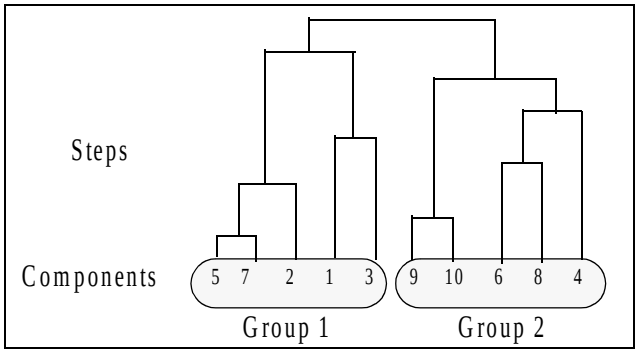
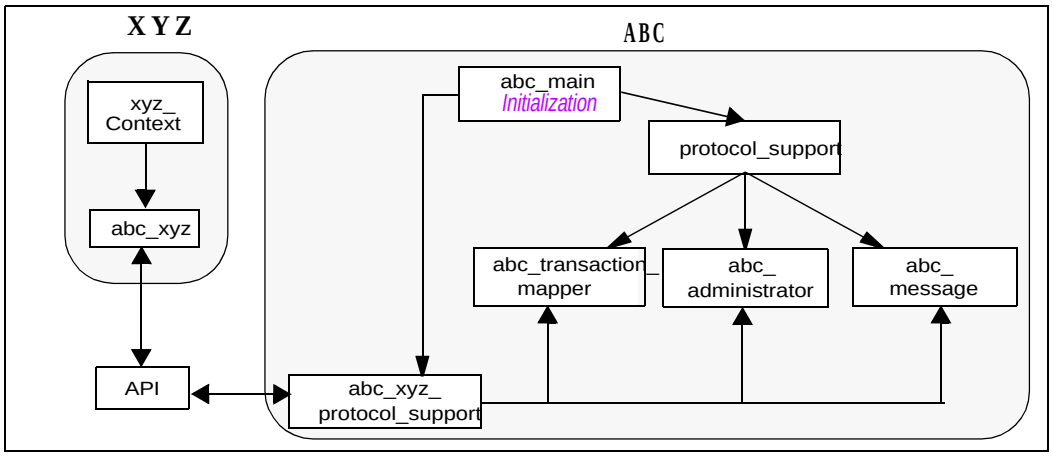


Figure 2. Clustering of Components Shown in Figure 1

4.2 An Example of Pattern Identification

Identification of appropriate patterns is not a trivial task. This section explains how the clustering technique could be used to support the identification of a pattern. The concept was inspired by C. Alexander's [1] work on identifying various styles for a village based on the features and their interactions. Figure 4 shows an example of components and their interconnections. There are some client classes that are accessing some subsystem classes. With existing software architecture recovery assistants, especially file names based approaches, the result may look perfect for the subsystem. In other words, the architecture recovered through this type of technique is close to or the same as the modules that are partitioned by the designer.

However, what are the values added to it besides explicitly



Data Complexity: 1
Strength: 0.4
# of Operations: 10

Figure 3. Initialization and Registration of Protocol Handlers - New Design

capturing the design in a high level architectural diagram? Certainly, architecture capture is important and valuable. But we are also concerned with ways to improve the architecture rather than simply capture it. Besides, in reality, the directory structures already often reveal the high level components of a system. Simply capturing a software architecture at a higher level abstraction often has limited benefits.

By applying the clustering technique to this example, we get very different partitions. In fact, the subsystem does not exist anymore, since many subsystem classes are directly accessed by or related to client classes. In other words, the clustering technique reveals that some classes in the subsystem are more closely related to client classes, which contradicts the design concept. Ideally, the subsystem classes should be grouped together as one unit.

Clustering techniques could be used in this type of situations to enforce the architect to reason ways to keep the subsystem classes in a more cohesive manner. Figure 5 shows an equivalent design by adopting the Facade design pattern [5] for the system. Facade pattern provides common interfaces to subsystem classes and facilitates separation of concerns. The subsystem classes in the new pattern-based design are grouped in the same unit according to the clustering method, which is consistent with the original design. In this example, the clustering technique helps the adoption of a design pattern to reduce the coupling between the subsystem and the clients.

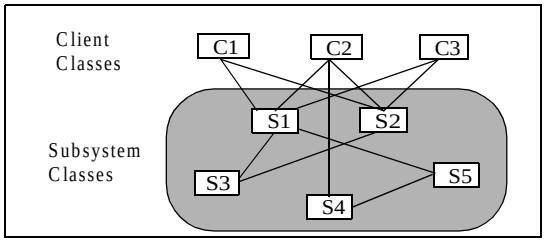


Figure 4. Components and Interactions: an Design Example

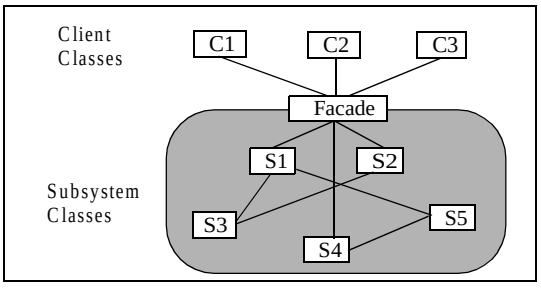


Figure 5. Adoption of Facade Design Pattern

5. SUMMARY

This paper presented an approach based on a clustering technique to not only recovering software architectures, but also improving them to give more added values. Use cases are used together with the clustering methods to reduce complexity. Several clustering algorithms have been implemented and have been applied to various problems at different levels of abstraction [8]. The results are encouraging and consistent. A visualization tool, SPV (Software Partition & Visualization) is also being built on top of the clustering methods to provide a user friendly environment.

Some open issues still lie ahead for further investigations. The main issues that are currently under study include

- Study the impact of different weights for different types of connection between components.
- Study and compare various clustering algorithms for suitability of different applications.
- Study how clustering techniques could be used to support the identification of patterns in a more effective way.

6. ACKNOWLEDGEMENTS

Thanks are due to Alex Cachia, Kennedy Ngo, David Wang, and Li Zhang for providing insights of the problem domain. I also would like to thank Anant Jalnapurkar, Kalai Kalaichelvan, and Rama Munikoti for their encouragement and support.

7. REFERENCES

[1] C. Alexander. *Notes on the Synthesis of Form*. Harvard University Press, Cambridge, MA, 1964.

[2] N. Anquentil and T. Lethbridge. Extracting Concepts from File Names: A New File Clustering Criterion. *Proc. of Int'l Conf on Software Eng*., 1998, pp. 84-93.

[3] D.N. Card and R.L. Glass. *Measuring Software Design Quality.* Prentice Hall, 1990.

[4] H. Dhama. Quantitative Models of Cohesion and Coupling in Software. *J. of Systems and Software*, vol. 29, 1995, pp. 65-74.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[6] G. Heyliger. Coupling. *Encyclopedia of Software Engineering*, J. Marciniak (ed.), 1994.

[7] R. Kazman and S. J. Carriere. Playing Detective: Reconstructing Software Architecture from Available Evidence. *J. of Automated Software Eng*., 1998.

[8] C.-H. Lung. Effective Software Partitioning through Clustering Techniques. *Nortel Design Forum*, Oct 1997.

[9] H. C. Romesburg. *Cluster Analysis for Researchers*. Krieger, Malabar, Florida, 1984.