J. Basic. Appl. Sci. Res., 2(9)9080-9087, 2012
© 2012, TextRoad Publication
ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Extended Visualization of Overlapping in Recognized Design Patterns

Ghulam Rasool[1], Muhammad Umair[2], Ramzan Talib[3]

Lecturer at Department of Management Sciences, COMSATS Institute of Information Technology, Raiwind Road, Lahore Pakistan

ABSTRACT

A large number of design pattern recovery techniques supplemented with tool support are presented in the past to extract patterns from source code of legacy applications. Different tools present their results in different formats. The maintenance activities can yield to true benefits of pattern recovery if the results of pattern recovery tools are comprehensible and reusable. The comprehension of relationships between the results is important because most of pattern based designs use different patterns which are connected with each other. Visualization of overlapping in recognized design patterns plays a key role for comprehension, maintenance, reverse engineering and reengineering. This paper focuses on detection, analysis and visualization of overlapping in recognized design pattern instances which is important for the maintenance and comprehension of legacy applications. The scope of work in this paper is limited only to structural design patterns. Based on the results of experimental examples the concept of the approach is validated.
KEYWORDS: Design patterns, Design pattern recovery tools, Tool analysis, program comprehension, Reverse Engineering

1. INTRODUCTION

Design patterns are widely used in different application areas such as software architectures, user interfaces, security, services, etc and recovery of patterns from these applications support software maintenance, program comprehension and re-engineering disciplines. Due to the ever increasing trend of new types of pattern catalogues used for the development of different applications, the discovery of patterns from source code of legacy applications remained the focus of the research community during last decade. State of the art pattern recovery tools present their results which give information about the existence of patterns, but this information is not detailed enough to be used for different domains of reengineering, maintenance and comprehension. The recovery of relationships between different design patterns can help to understand the logics of the design model. In this paper, we focus on the detection and visualization of overlapping in design pattern instances.

Overlapping occurs when an individual design part plays a role in two different patterns which makes the design dense and more profound. Overlapping can be observed on all the building parts of a pattern that are suggested in a pattern definition. At a higher level, overlapping indicates strong integration between individual patterns because each class has several responsibilities and designers end up designing with fewer classes. The salient disadvantage of overlapping is that the pattern boundary is lost and pattern tracing becomes hard [8]. We analyzed existing solutions where we found that proven composite patterns are present with different overlapping. For example, the Java AWT framework is composed from different patterns and the detection of overlapped pattern roles becomes significant for understanding of design model. Similarly, MVC [9] is composed form Observer, Strategy and Composite patterns. The statistics of overlapped elements of patterns reflect repeated roles of different classes and patterns. Such information can be used for a change impact analysis. Pattern roles that are most reused may also be most change-prone.

There can be different types of overlaps in the design of systems such as one to one, one to many and many to many. One to one overlap means that one object/class in design of a system is reused in two different pattern instances. For example, a design model M has two patterns P1 and P2. R1 is the role which is used in both P1 and P2. One to many overlap means that a role in one pattern is reused more than one time in another pattern. For example, the role Product in Abstract factory method can be reused in the Bridge pattern as Abstraction and Implementation. Many to many overlapping occurs among patterns when more than one role in a pattern are reused more than one time in another pattern.

The major focus of different approaches on visualization of compositions and overlapping in design patterns is to visualize pattern information based on design model information. Currently, there is no modeling tool that can extract all the information of source code into a design model. Nonetheless, current approaches help in understanding of overlapping in design patterns at design level. Our approach is different because we focus on detecting overlapping at source code level and visualize detected overlapping at the design level. The detection of visualized overlapping in this paper is limited to the structural design patterns of GoF [1] design patterns, although the approach is flexible to be extended to other types of design patterns in the future. The rest of the paper is organized as follows:

***Corresponding Author**: Muhammad Umair, Department of Computer Science, COMSATS Institute of Information Technology, Lahore, Pakistan. mumair@ciitlahore.edu.pk

Section 2 discusses the related work. Section 3 discusses the background and motivation for overlapping detection in design patterns. Section 4 presents the methodology used for the detection and visualization. We perform experiments on open source examples to realize the concept of the approach in section 5. Empirical evaluation of results is discussed in section 6. Finally, Section 7 presents our conclusions and future work.

2. Related Work

An empirical study on the representation of design patterns using different notations is presented in [22]. Authors compared important publications [3 13 17]. They discussed strengths and limitations of three approaches and they proposed framework which can be used to compare past, current and future notations used for representation of design patterns. They also came up with requirements and guidelines which can be used by tool developers to select appropriate representation for particular tasks while developing prototyping tools.

Dong et al. [3] presented static and a dynamic analysis technique for visualization of design patterns in system design. They present a UML profile that defines new stereotypes, tagged values and constraints for tracing design patterns in UML diagrams. Their approach is to store design pattern related information in stereotypes and tagged values manually and then apply their web based tool VisDP[16] to visualize patterns in UML diagrams. The pattern related information is hidden in UML diagrams and displayed on demand using different colors. They performed experiments on parts of a conference management software system which is not publically available and the scalability of applied tool for visualization of large systems is questionable. Our approach is different because we first extract design patterns from the source code and then visualize the overlapping of patterns instances in UML diagrams which is important for the comprehension of design patterns.

Pattern enhanced class diagram approached presented in [13] is strongly visual. Authors used color scheme on border of each class to visualize role of individual class in each pattern. The canonical representation of approach is simple, comprehensible and easy to locate classes participating in patterns. Authors argue that they enhanced UML representation but statistically it is difficult to measure improvement in the presented approach. The experiments are performed on very small examples and generalization of approach for large and complex systems is questionable.

A graphical notation based on pattern: role annotations in presented in [14]. The concept of approach is to tag each class with a shaded box which contains pattern and role information for each class. This notation is scalable, readable and informative for simple design. The additional information with each class makes design to occupy double space and comprehension becomes difficult in the case of increased number of overlaps among different roles. The operations and attributes also play important roles in design patterns and this notation cannot represent the roles that an operation and attribute plays in a design pattern.

UML Collaboration annotations [17] by attaching dashed ellipse to class diagrams are used to highlight design patterns information. Patterns are highlighted in class diagram by using dashed lines connected with ellipses. Ellipse represents the name of the pattern and dashed lines represent the role that a class plays in one or more design patterns. Annotations used in diagrams can explicitly represent the participant role a class plays but the roles that an operation (attribute) plays in a pattern are not addressed in this approach. The increased number of dashed lines clutter the representation and it is difficult to scale up large diagrams when a role is participating in number of design patterns.

A prototyping tool is presented in [4] for visualizing pattern based design information. This tool supports the recovery of design patterns using automatic, manual and semi-automatic clustering techniques. The structure of the tool consists of four main components: code reverse engineering, a design repository, design representation and design clustering. The tool supports three types of diagrams: pattern-enhanced class diagrams, pattern analysis diagrams, and pattern collaboration diagrams. The authors visualize design information in UML class diagrams which makes comprehension of individual pattern easy. The focus of our approach is to visualize relationships between design patterns and extract statistics of overlapped elements.

The approach presented in [6] has focused on the composition and overlapping of design patterns. A concept of composition of patterns with respect to overlap is formally defined based on specialization and lifting operations on patterns. The approach focuses on composition and formal verification of patterns but it has no connection with the visualization of recognized design patterns which is a major focus of our approach. The theoretical background of the approach can be used by different design pattern recovery tools for detecting overlapping in design patterns.

Heričko et al [7] presented a composite design pattern identification technique based on coverage and overlapping pattern metrics. Their approach uses three steps for detecting compositions of design patterns. The overlapping pattern matrix indicates the level of overlapping between different artifacts playing roles in different atomic patterns. They presume that proven design pattern compositions are already analyzed in the examined systems and demonstrate their composition detection process using the MVC pattern. The focus of authors is to detect new compositions of design patterns at design level, although the definition of new compositions is not clearly explained.

Riehle [11] discussed role diagrams for design pattern composition. Author introduced the concept of a role relationship matrix which contains composition constraints for design pattern classes. He assumed implemented design patterns as a collaboration of classes, where each class may play multiple roles in its respective abstract patterns. The approach is based on basic theoretical concepts and it is not applied on any real case study. Further, author discussed only composition of different

design patterns but he did not focus on detection of composition which can be used for visualization.

Reference[12] presented approach which is used to detect and visualize web design patterns. The approach parses the web pages using Jericho HTML Parser for the detection of patterns and JGraph package is used for the visualization of detected patterns. Authors selected patterns for experiments which can be detected only with the help of static analysis. The generalization of approach for other types of web design patterns is questionable. In visualization, the focus in only on individual web design patterns while our major concern is the visualization of overlapping and composition in design patterns.

3. Background and Motivation

New types of pattern catalogues have motivated the quest for recognizing new design patterns and analyzing their relationships in the source code. Pattern-based designs foster designers to reuse existing patterns and pattern combinations. Patterns usually lead to an increased number of software artifacts, which normally increases the static complexity of a software system considerably [10]. A high level of overlap in a pattern prevents the undesired increase of artifacts. Upholding this level forces the designer to spend much effort for the integration of a new pattern while keeping the quality of the resulting architecture high. A high level of overlapping indicates strong integration between individual patterns. The following list shows possible types of overlapping in design patterns which are the major motivations for this paper:

- A role is overlapping in same type of pattern instances with similar role.
- A role is overlapping in same type of pattern instances with different roles.
- A role is overlapping in different pattern instances.
- More than one role is overlapping in similar types of pattern instances.
- More than one role is overlapping in different types of pattern instances.
- How many times a role is overlapping in a design pattern?
- How many times a role is overlapping in different patterns?

The following motivations for design pattern visualization are also presented in [3] but these concepts are not implemented in the prototyping tool presented by authors. Our prototyping tool will focus on visualizing such information which is very important for the comprehension of design patterns.

- Identifying all the roles which participate in a design pattern.
- Identifying the role a class plays in a given design pattern.
- Identifying all design patterns in which a class participates.

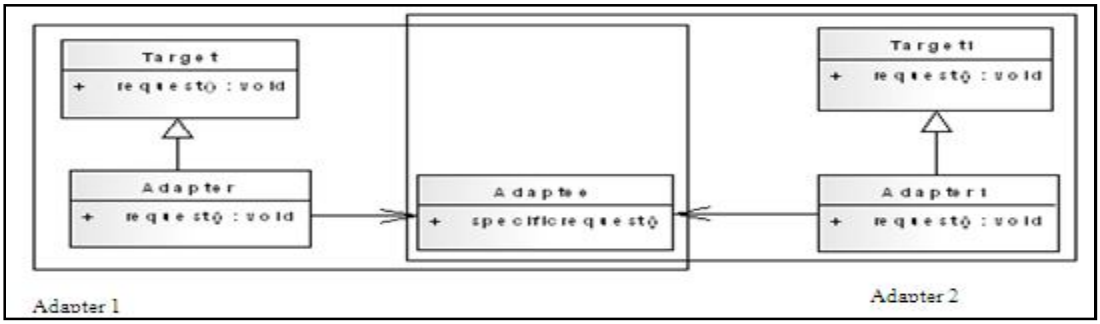

Fig. 1. Overlapping in same pattern instances with similar role

Fig 1 explains that class Adaptee is overlapping in two adapter pattern instances with a similar role and Fig 2 represents overlapping of the Adapreal class in the Adapter and Proxy patterns with different roles. These are simple types of overlapping which can be highlighted to indicate the traceability links of reused elements. Overlapping of Director role in Builder pattern and Template method pattern is presented in Fig 3 which is also discussed in [15]. Similarly Fig 4 represents different types of overlapping as mentioned in above described bullets. Detection of these different types of overlapping in different patterns is major motivation for this paper. It is worthwhile to mention that we focus on one to one type of overlapping detection, analysis and visualization in this paper.

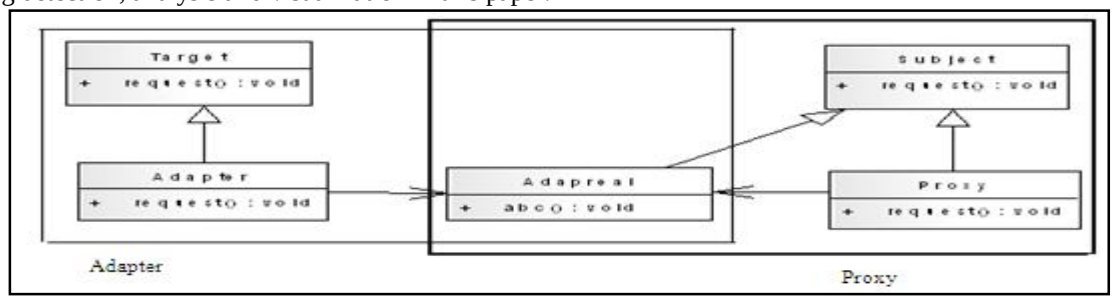

Fig.2. Overlapping in different pattern instances with same role

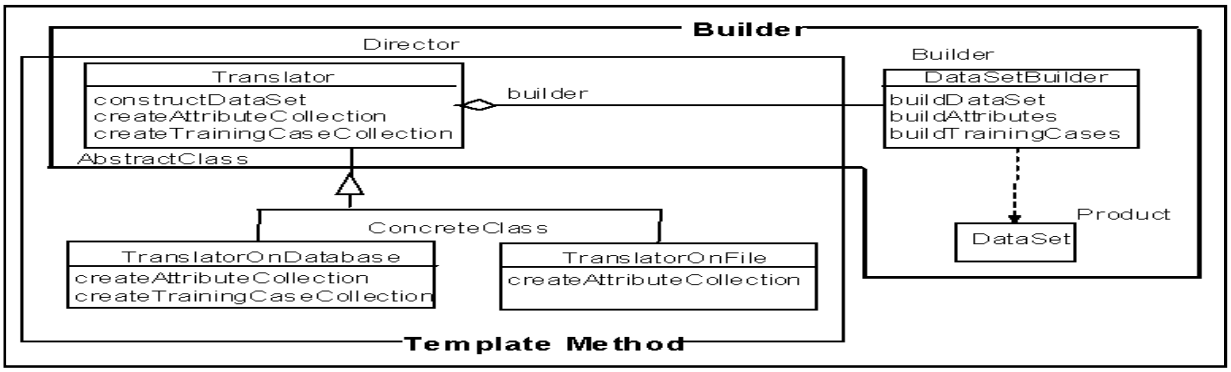

Fig.3. Different types of overlapping

Finally, Overlapping of more than one role in Factory method pattern and Visitor pattern is discussed in Fig 5. Bridge pattern and Abstract Factory method pattern can also overlap more than one roles. For example, Abstraction and Implementor classes in Bridge pattern can play the roles of Abstract Products in Abstract Factory method pattern. Similarly, Composite and Iterator can overlap more than one role. Such overlapping makes patterns tightly coupled and modifications in an individual class can impact all other classes and patterns. Best design practices recommend that strong coupling should be avoided in patterns only with the exception of Singleton pattern because application of that pattern results in code efficiencies.

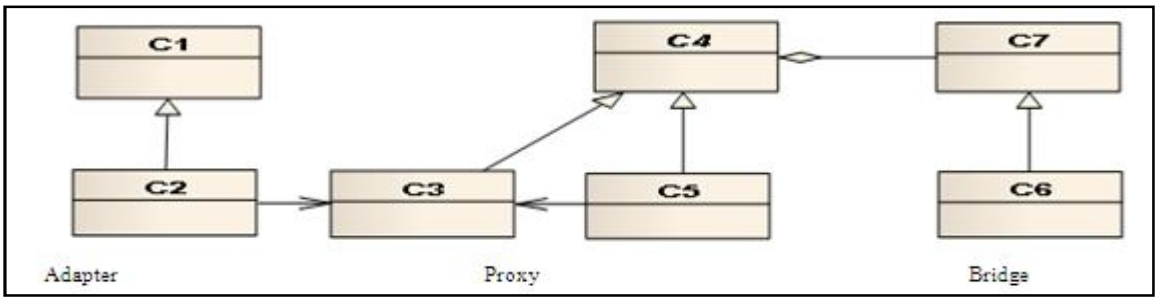

Fig.4. Different types of overlapping

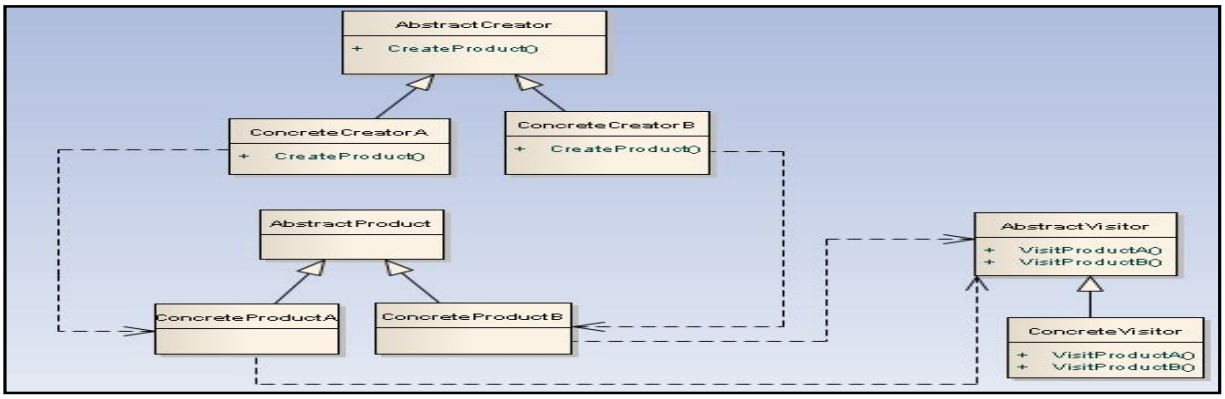

Fig.5. Overlapping of more than one role in different pattern instances

4. METHODOLOGY

We detected design pattern instances from the source code using our pattern recognition technique presented in [2 18] with complete roles. We believe that complete roles are important for comprehension of a pattern and its interconnections. We developed a prototyping tool using .Net framework as Add-In with Sparx System Enterprise Architect Modeling tool [19] as proof of concept for presented methodology. The used prototyping tool can be integrated with other UML modeling tools such as IBM Rational rose [20] and Borland Together [21]. The prototyping tool accepts input of recognized patterns and generates output in different views which are important according to requirements of the users. The tool presents results in class, table and report views. The class view visualizes overlapping using UML class diagrams. Color schemes are used in class diagrams to highlight different types of roles belonging to different patterns. Notes in class diagrams show detailed information about each overlapping element and its connection with other patterns. The table view presents overlapping roles in tabular form which is easy to comprehend and provides a quick view of elements overlapping in different patterns. By

pointing to single elements with the mouse cursor detailed information about overlapping roles used in different patterns is shown. Finally, the report view generates reports of overlapping elements for an individual pattern and for all patterns. The prototype tool in use is flexible enough to accept output generated from other design pattern recognition tools, which is also a precondition for our tool. However, complete roles of each pattern which are specified by GOF [1] are also important precondition for using prototyping tool used in our approach.

The conceptual diagram used in detecting, analyzing and visualizing overlapping in design pattern instances is mentioned in Fig 6. The tool uses a text file and a database model of the source code as input and uses different algorithms to detect overlapping in design pattern instances. The user can guide the detection step to detect overlapping in a single design pattern or in all design patterns. The analysis step takes data from the detection phase and produces overlapping metrics as well as a report view. Overlapping metrics give detailed information about the role of each overlapping element in different design patterns. For example, we extract the number of each overlapping role and the overlapping of each pattern in relation to other patterns. The report view is used for the detailed analysis of the results which are important for maintenance of the software. Finally, the visualization step presents the output in class and tabular views.

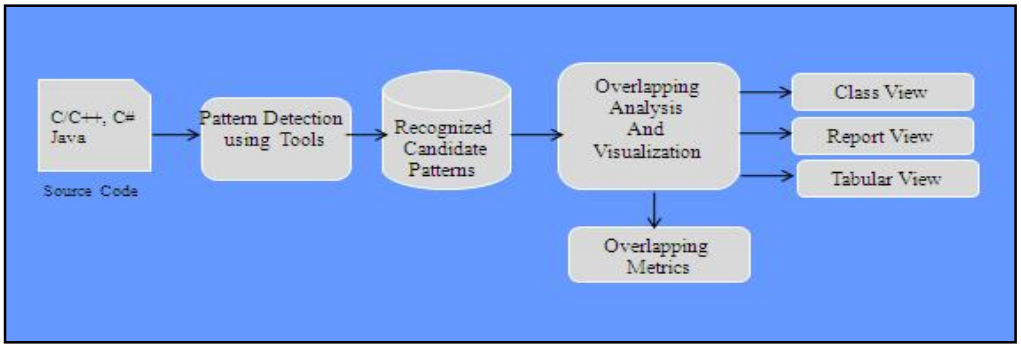

Fig.6. Conceptual diagram used in visualizing overlapping

5. Experimental Examples

We performed experiments using Junit3.7 [5] and JHotdarw5.1 [23] which are developed using different design patterns. Junit3.7 is a unit testing framework used for writing repeatable tests. It is developed with 85 classes stored in 45 files. The total number of lines of the code is 142.8 KLOC for this version. JHotDraw5.1 is a Java framework for drawing two-dimensional technical and structural graphics. It is commonly used to develop and customize graphical editors for different applications. This version contains 136 classes and total lines of source code are 30860. The objective of selecting these examples is to evaluate the initial concept of our approach on structural design patterns.

Table 1 presents the overlapping elements detected from Junit3.7. The detailed information about the complete roles of patterns is reduced due to limitation of space. The last column in Table 1 shows statistics of overlapping elements used in different design patterns. These statistics are very important for maintainers to know the impact of individual role in different design patterns.

Fig 7 shows the tabular view of detected overlapping roles for the Adapter design pattern instances from Junit3.7. User can select any single instance from tabular view and can display that particular instance in class view. Similarly, Fig 8 presents the class view of detected overlapping elements. We want to clarify that Fig 8 presents the class view of overlapping elements using only the Adapter design pattern. Our prototyping tool is capable to visualize overlapping of all structural design patterns. The report view is used for a detailed analysis and documentation purposes as presented in Fig 9.

Table 1. Overlapping elements and their statistics

Object/Pattern	Adapter	Composite	Decorator	Count
C:\Junit3.7.\framework\Test.java	-	Component:Test	Component:Test	**2**
C:\Junit3.7\ framework \TestListner.java	Target:TestListner	-	-	**3**
C:\Junit3.7\swingui\TestRunner.java	Adapter:TestRunner	-	-	**2**
C:\Junit3.7.\framework\TestSuite.java	-	Composite:TestSuite	ConcreteComponent: TestSuite	**2**
C:\Junit3.7\swingui\TestSuitePannel.java	**Adapter:TestSuitePannel Adaptee:TestSuitePannel**	-	-	**2**

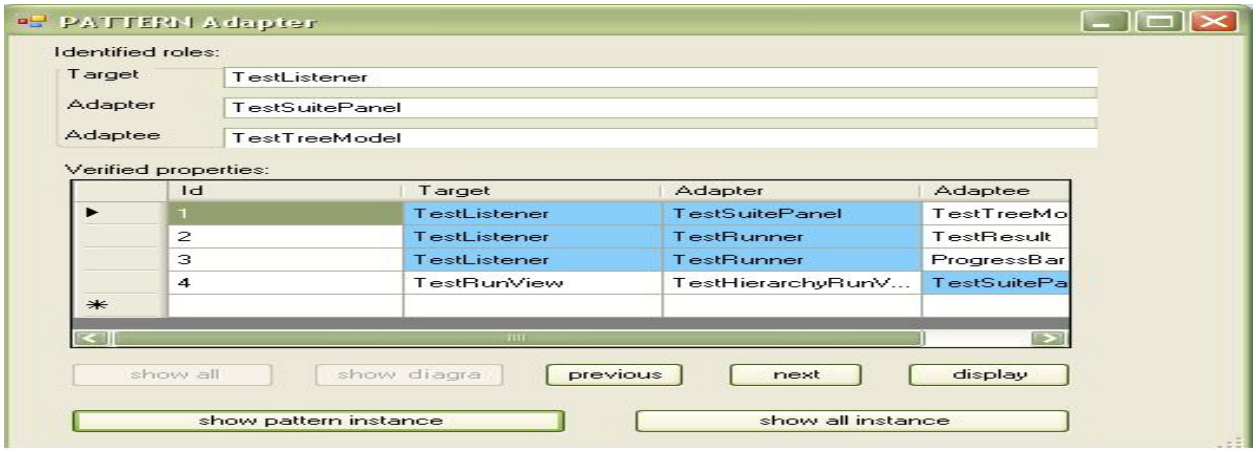

Fig.7. Tabular view of overlapping elements

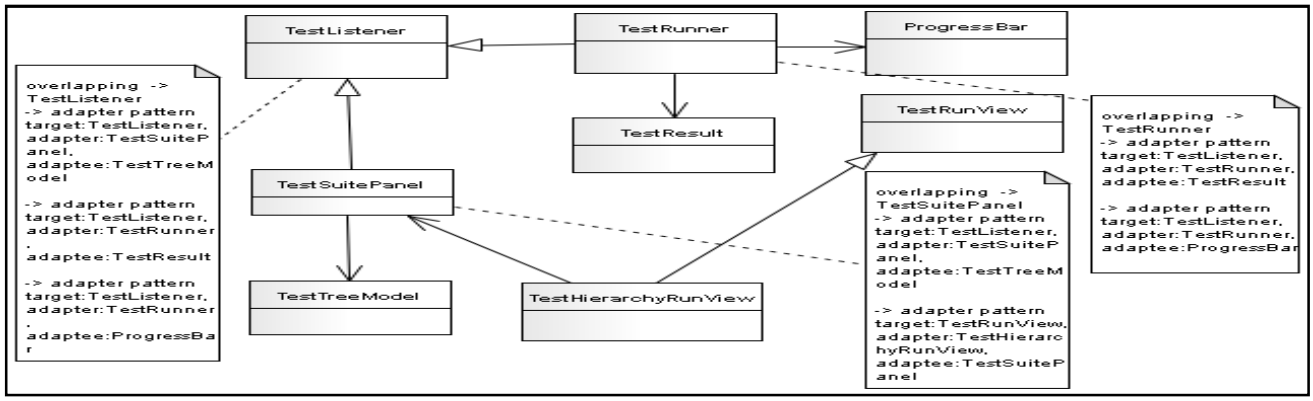

Fig.8. Class view of overlapping elements

The report of overlapping elements in adapter pattern instances

AbstractFigure is the overlapping element of adapter patterns:

1) Figure, AbstractFigure and FigureChangeListner
2) AbstractFigure, TextFigure and OffsetLocator

AbstractHandle is the overlapping element of adapter patterns:

1) AbstractHandle, FigureRadiusHandle and RoundRectangeFigure
2) AbstractHandle, PolygonHandle and roundRectangleFigure
3) AbstractHandle, LocatorHandle and Locator

Fig 9. Report view of overlapping elements

Overlapping roles and their statistics extracted from JHotDraw5.1 are presented in Table 2. We list only roles of limited classes as sample from Adapter, Composite, Bridge and Decorator design patterns. Our prototyping tool creates such information automatically for each individual role from all structural design patterns. The last column in Table 2 gives information that how many times each role is overlapping in one or more patterns. The results of approach and tool extracted from JHotDraw5.1 in tabular view and class view are presented in Fig 10. The report view is curtailed due to limitation of space. The prototyping tool can be operated in user interactive mode which allows user to highlight detailed information according to his requirements. For example, user can click on any role in class view and tool displays information about overlapping of that role in different design patterns. Due to large number of classes in JHotDraw 5.1, it was difficult to display complete results and we present partial results on limited number of classes.

Table 2. Overlapping roles and their statistics

Object/Pattern	Adapter	Composite/Brdige	Decorator	Count
C:\JHotDraw5.1.\draw\framewor\Figure.java	Target: Figure Adaptee:Figure	Component:Figure Implementor:Figure	Component:Figure	**8**
C:\JHotDraw5.1.\draw\framewor\Tool.java	Target:Tool	Implementor:Tool	Component:Tool	**3**
C:\JHotDraw5.1.\draw\figure\TextFigure.java	Adapter:TextFigure	Implementor: TextFigure	-	**2**
C:\JHotDraw5.1.\draw\framewor\Locator.java	Adaptee:Locator	Implementor:Locator	Component:Locator TestSuite	**5**
C:\JHotDraw5.1\draw\applet\ DrawApplet.java	Adapter: DrawApplet	Abstraction:DrawApplet	-	**5**
C:\JHotDraw5.1.\draw\framewor\Figure.java	**Target:DrawingView**	**Implementor: DrawingView**		**9**

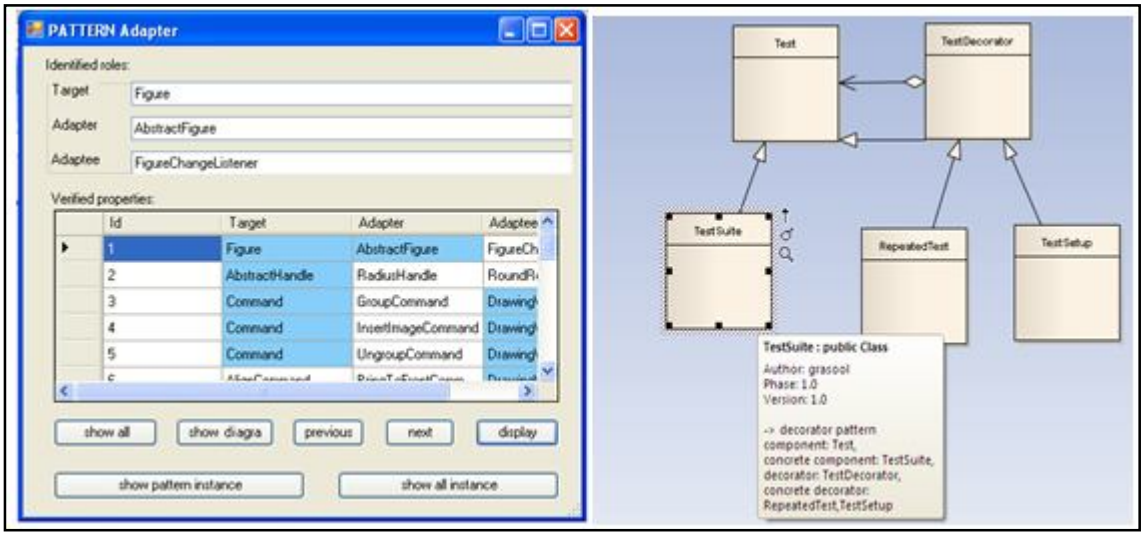

Fig 10. Table and Class view of overlapping elements from JHotdraw5.1

6. Validity Threats

Validity is the key challenge for researchers and practioners in conducting empirical research work. One of the major threats to the results of our approach is the lack of standard definitions for design patterns and unavailability of trustable baselines for validation of our results. While one can precisely define a pattern, but there is no agreed upon definition of different variants for each design pattern to-date. We reduced this threat as we used results of extracted patterns from our own baselines which were manually verified for most of patterns presented in [18]. The internal and external validity can be assured because the experimental examples are open source systems. The source code and results of extracted patterns are available on web for review and comparisons. However, generalization of our results for class view may affect external validity of our visualized results in the case of large systems.

7. Conclusions and Future Work

We presented an approach on the visualization of overlapping in structural design patterns which positively influences software maintenance and program comprehension activities. Visualization and analysis of overlapping in recognized design patterns is important practical approach for understanding legacy applications, which is not intensively explored yet. We detect overlapping and visualize information using class, table and report views. The concept of approach is realized with prototyping tool developed as Add-In with Sparx System Enterprise Architect Modeling tool. The prototyping tool can be integrated with different design pattern recovery tools to detect overlapping in recognized design pattern instances. We plan to extend our approach to creational and behavioral design patterns. The scalability of approach will be evaluated on large and complex systems. The future work will also focus on detection of overlapping from incomplete roles of design patterns.

Acknowledgements

The authors would like to thank Xiang Yu for implementing parts of the prototype used in this research.

REFERENCES

1. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.M., “Design Patterns: Elements of Reusable Object Oriented Software”, Addison-Wesley Publishing Company, Reading, MA, 1995.

2. Rasool, G., Philippow, I., and Mader, P., A Design Pattern Recovery Based on Annotations, In Journal of Advances in

Engineering Software, ISSN(0965-9978), Volume 41, Issue 4, Pages 519-526, April 2010.

3. Dong, J., and Zhang, K., Visualizing design patterns in their applications and applications, IEEE transactions in software Engineering, Volume 33, No. 7, pp.433-453, July 2007.

4. Schauer, R., and Keller, R.K., Pattern Visualization for Software Comprehension, In Proceedings of 6th International Workshop on Program Comprehension, pp. 4-12, Ischia, Italy,1998.

5. Junit Homepage: http://www.junit.org. [Accessed on 20.04.2011].

6. Bayley, I., and Zhu, H., On the Composition of Design Patterns, In Proceedings of Eighth International Conference on Quality Software, pp. 27-36 Oxford, UK, August 2008.

7. Heričko, M., Beloglavec, S., A Composite Design-Pattern Identification Technique, Journal of Informatica, Volume 29, pp. 469–476, 2005.

8. Gogolla, M., Kobryn, C., UML 2001, The Unified Modeling Language, Lecture Notes in Computer Science, Volume 2185, pp. 151, 2001.

9. Kramer, G. E. and.Pope, S. T., A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80, Journal of Object-Oriented Programming 1, p. 26-49, August/September 1988.

10. Wendorff, P., Assessment of design patterns during software reengineering: Lessons learned from a large commercial project, Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR'01), 2001.

11 Riehle, D., Composite design patterns. OOPSLA'97, pages 218-228, Atlanta, GA, October 1997.

12. Aminzadeh, N., and Salim S.,D., "Detecting and Visualizing Web Design Patterns", The 2nd International Conference on Computer and Automation Engineering, pp. 100-103, 2010.

13 Schauer R., and Keller R., Pattern visualization for software comprehension. In Proceedings of the 6Th International Workshop on Program Comprehension, pages 4–12. IEEE Computer Society, 1998.

14 Vlissides J., "Notation, Notation, Notation," C++ Report, Apr.1998.

15. G. Masuda, N. Sakamoto, and K. Ushijima. Applying design patterns to decision tree learning system. Proc. ACMSIGSOFT Int. Symp. Foundations of Software Engineering, pages 111–120, 1998.

16. Dong, J., Yang, S., and Zhang K., "VisDP: A Web Service for Visualizing Design Patterns on Demand," Proc. IEEE Int'l Conf.Information Technology Coding and Computing (ITCC '05), pp. 385-391, Apr. 2005.

17 Booch, G., Rmbaugh, J., and Jacobson, I.,The Unified Modeling Language User Guide. Addison-Wesley, 1999.

18 Rasool, G., and Mäder, P., Flexible design pattern detection based on feature types, In Proceedings of ASE 2011, pp. 243-252.

19. Sparx System Architect Modeling tool. http://www.sparxsystems.com/products/ea/

20. IBM Rational Rose Website, http://www.ibm.com/software/rational

21. Borland Together, http://www.borland.com/us/products/together/

22. Gerardo Cepeda Porras and Yann-Gaël Guéhéneuc. An Empirical Study on the Efficiency of Different Design Pattern Representations in UML Class Diagrams. Empirical Software Engineering (EMSE), 15(5), January 2010.

23. JhotDraw Homepage, http://www.jhotdraw.org/