J. Basic. Appl. Sci. Res., 3(1)139-147, 2013

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

Discovering Variants of Design Patterns

Ghulam Rasool, Hassan Akhtar

COMSATS Institute of Information Technology, Department of Computer Science, Lahore, Pakistan

ABSTRACT

Design pattern recovery techniques and tools are used to recognize patterns from source code of different applications. The accuracy of presented techniques and tools is still debatable due to different factors. One of the major factors that affect the accuracy of existing techniques and tools is the problem of variations. Developers adopting patterns apply standard patterns with the same intent for solving generic problems by using different programming constructs. The structural variations are also noticed when roles in standard patterns are merged or extended. Such variations hamper the accuracy of pattern detection techniques and tools. In this paper we present four new variants of standard patterns realized during analysis of different source applications which are developed using different patterns. These variants can be used by developers to solve different generic problems and they will also extend the existing catalogues used for specification of patterns. The presented variants are analysed and validated through state of the art design pattern recovery tools.
KEYWORDS: Design Patterns; Variants;Pattern Recovery Tools; Program Comprehension; Reverse Engineering

1. INTRODUCTION

Since the publication of GoF [1] book, there is continuous debate on the applications of design patterns for developing high quality software applications. It is revealed in [2] that proper use of patterns increases development productivity of 25-40% depending upon expertise and skill level of developers. The key benefits of using patterns in software development applications are communication, implementation and documentation of software systems [3]. Standard patterns have gone through number of evolutions and they are realized with different variants by developers. These variants follow the same intent defined by the GoF but occur in different implementations that suits better according to the situation. New programming languages constructs cause the implementation variants of patterns. Similarly, designers can opt structural variants according to the situation and content of problem. The major challenge during design pattern recovery process is identification of patterns which have different variants.

Recovery of instances of design patterns and their variants from source code of legacy applications remained challenging due to nonexistence of formal definitions for all patterns and their variants. Different pattern specification languages [4, 5, 6] are used to specify patterns but these languages are not completely capable to specify all possible ways of implementing the patterns and their variants. The standard catalogue of all patterns with their variants is required to solve problem.

We analyzed the intent of Variants presented are of Abstract Factory, Decorator, Adapter and Proxy and discovered one variant of each pattern. The variations in these patterns follow the intent given by GOF but they are possible evolutions of standard patterns. We came across these variations during review and analysis of open source applications such as JHotDraw, JFreeCharts, JUnit, JText etc. We define these variants using feature types discussed in our previous papers [7, 8, 9] and implement each variant using Java programming language. Finally, all variants are tested using state of the art design pattern recovery tools [7, 10, 11]. The room for availability in design patterns variations is an unlimited space; we will try to cover up the implementations with example scenarios for the proof of concept and their application that exists in the real world. The discussed variants can benefit the community in three contexts: firstly when developers come across some scenario where they need to tweak the implementation but at the same time they don't want to break the intent of design patterns they can map structures with presented scenarios. Secondly, the accuracy of design pattern detection tools can be evaluated on presented variants. Thirdly, new tool developers can incorporate knowledge of these variants during development of new design pattern recovery tools.

***Corresponding Author:** Ghulam Rasool, COMSATS Institute of Information Technology, Department of Computer Science, Lahore, Pakistan. Email: grasool@ciitlahore.edu.pk

2.RELATED WORK

Authors in [12] presented eight implementation variants of singleton design patterns. They presented an intuitive definition of the Singleton pattern using first order logic that covers different variants of the Singleton. They claim that pattern detection approach used by authors is only capable to recognize all presented variants of singleton but the tool used by authors is not available publically to validate their claim. Authors did not discuss applications of these variants to solve real problems. The same group presented another paper on the diverse implementation variants of design patterns. They discussed variants of Singleton, Abstract factory and builder and tested variants using their own prototype tool and other state of the art tools. They concluded that applied tools are not able to recognize variants of standard patterns due to hard coded algorithms used by prototyping tools.

A meta-modeling approach [13] using GEBNF notation is used by authors to specify design patterns and their variants based on structural and behavioral properties. The approach enables formal reasoning about patterns, their composition and transformation and provides automatic tool support for applying patterns at design stage. They specified all 23 patterns. The approach is extensible for specifying variants of design patterns. The major concern is integration of approach with existing state of the art pattern detection tools.

Authors in [14] presented approach based on OCL backtrack algorithm to specify and identify structural variants of composite design pattern. Four different variants of composite pattern are discussed. The applied algorithms are pattern specific and their generalized is questionable. Furthermore, the approach is not able to handle dynamic features of patterns.

Reference [15] discussed eight variants of Factory method pattern using generative model which is kind of algebra for problem and solution spaces and projection from problem to solution space. The applied tool understands pattern catalogue, construct generative model schema and then construct individual pattern models. Finally, it evaluates relationships among individual patterns. The applied model is only tested for creational design patterns and its generalization for other patterns needs to be investigated.

Authors presented a survey on micro-structures: elemental design patterns, clues, sub-patterns, and micro patterns which are used for the detection of design patterns and their variants [16]. They also compared these micro-structures based on different parameters. Authors discussed how different micro-structures handle the variations of design patterns. The major problem is lack of community consensus on using these micro-structures.

3.DISCUSSION ON DISCOVERED VARIANTS

In this section we discuss some of the possible scenarios that provide a room for new variations in design patterns. We categorize each pattern into subsections which include the intent, motivational background, UML diagram, critical review and feature types [7, 9]. Feature types are basically characteristics extracted from object oriented implementation of design patterns with inspiration taken from class and sequence diagrams. These feature types are generic and can be overlapped across multiple programming languages. A complete catalog published by [7] is the motivational source of support for discovered variants which we are using and extending throughout our work. The presented variants differ from the standard form by either an addition of already defined feature types in pattern definition or an introduction of new feature type in the catalog. We highlight the change by yellow (change in parameter in existing feature types) and grey (new features types) colors in the diagrams given in the following subsections. Figures 1, 3, 5 and 7 represent the GoF structure and discovered variant structure of each pattern. Yellow color in each of these figures highlights the key role which causes the variation of a standard pattern. Similarly, Figures 2, 4, 6 and 8 represent the feature types for standard and variant representation of the same pattern.

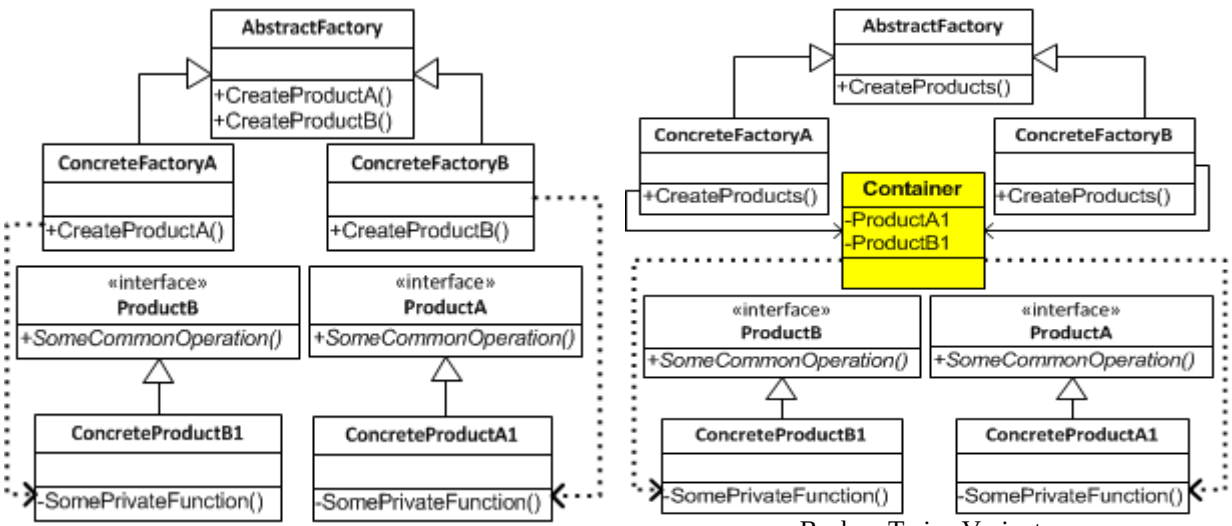
3.1 Abstract Factory (back up twins)

Abstract factory is termed as families of interrelated objects which actually deal with possible representations of objects. We can interpret the usage of this pattern as highly recommended solution when same kind of objects switch their instantiation when intended to conform through a single interacting interface. Being called as factory of factories, the responsibility holds with a single class to return products leaving the client unaware to specify the exact product. An interacting interface is used to encapsulate the actual product created.

Motivation

Dealing with legacy environments, we often optimize the functionality with performance tweaks in terms of complexity, speed and reliability. In production systems we do not immediately commit the change to be ready for application usage with a single go. The impact of code-refactoring techniques ends up in breaking existing

functionality which turns into a big disaster for data critical systems. Therefore architects classify the existing functionality and the new functionality under one roof and switch their calling based on the current state of the system. The states can be differentiated something like high memory, low disk or vice versa. So these versions of classes operate on suitable environments at the execution level in switch case scenario providing back up to each other. This variation of Abstract Factory deals with the instantiation of all products of a family at a certain time. The need to instantiate all products can be best interpreted as UI elements where objects are like fraternal twins which behave differently.



GoF Abstract Factory Pattern

Backup Twins Variant

Figure 1. Abstract Factory and Abstract Factory Twins Variant

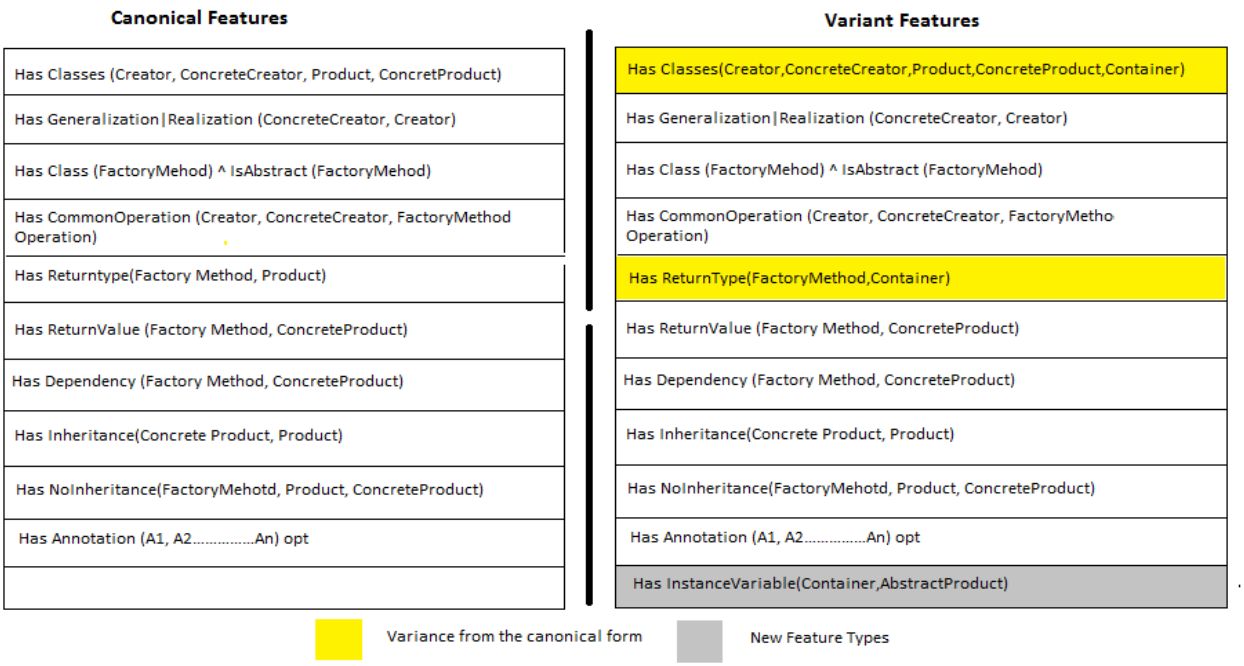


Figure 2. Feature types of Abstract Factory Pattern (left) and Abstract Factory Twins(right)

Critical Overview

Following are the points of discussion on the back twins' variant:

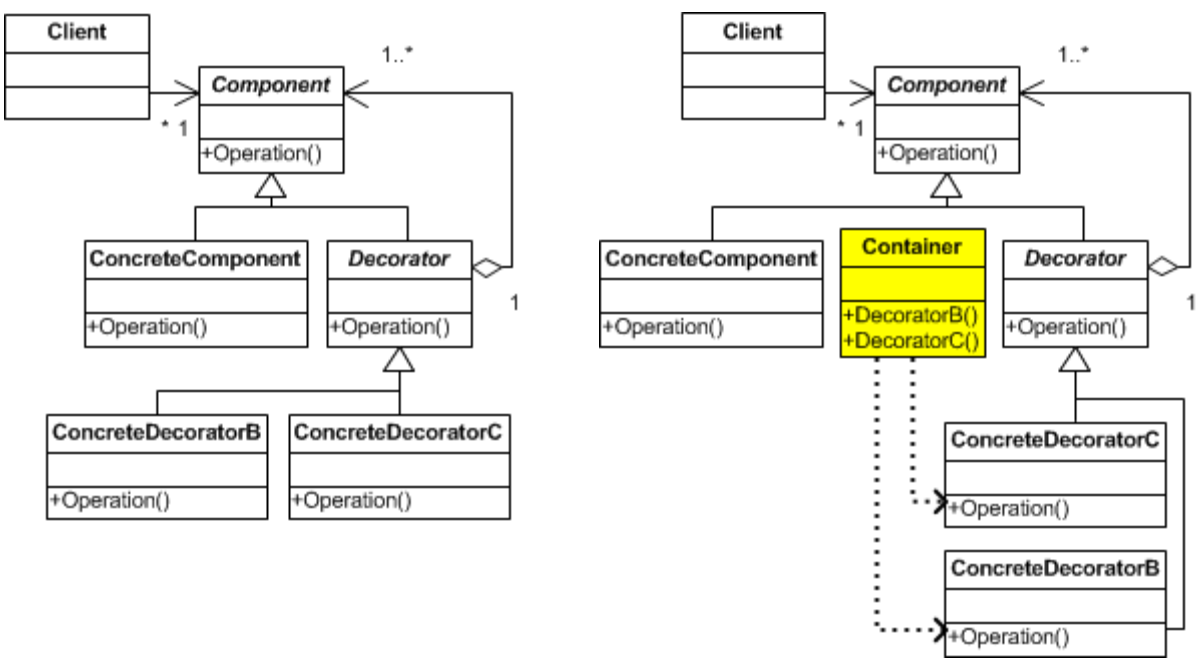
1. This variant is differentiated from the canonical representation in terms of an additional participant role (place holder class) and factory method to return place holder class instead of abstract product.
2. The variation advocates the use of a third class which hold references to all products of families. Factory returns all products in a single shell, providing a full choice to the client instead of requesting the factory again and again with all products pre-populated
3. The applicability of this variant can be in Paint applications where a canvas have all the colors, brushes, text art and other formatting place in a tool box with default properties.

3.2 Decorator

Cosmetic changes to an object that does not impact the class itself but object involves the Decorator pattern to be adopted. Usually objects are dressed from inner to outer layer by layer. Each layer is an additional responsibility that is added to the Component class at any time by the client. Self-dressed decorator is a concept where components are dressed based on suggestions or specific events.

Motivation

During literature review we studied the original GOF pattern used over multiple scenarios to dress up sole components. Decoration is like layers on objects just in case we see when we give birthday gifts. We wrap up the gift in a box, then cover up with some fancy creepy paper along with a bow at the outer layer which adds value to gift and the receiver at the other end realizes how perfectly you cared his emotions. Imagine without decorators it could be so much difficult e.g. ready-made gift boxes available in different sizes and colors that sometimes doesn't suit the occasion or the personality which you are presenting. This is why we have each component separately available, all you need is to dress up with your need. Decoration usually follows set of steps that are must followed when you actually make up a component. The variation which we are suggesting is responsible enough to dress up a component from a list of available or suggested decorators that holds compatibility.



GoF Decorator Pattern

Self-Dressed Decorator

Figure3. Decorator Pattern and Self-Dressed Decorator Variant

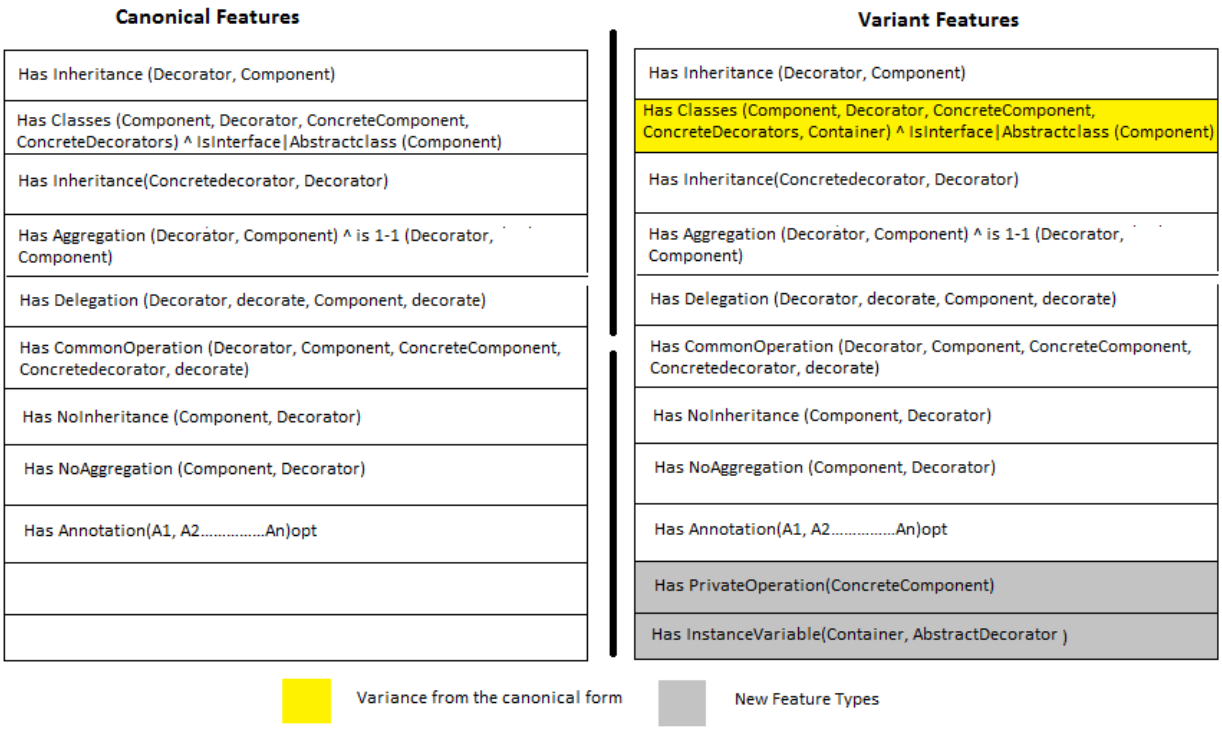


Figure 4. Feature types of Decorator Pattern (left) and Self-dressed decorators (right)

Critical Overview

Following are the points of discussion on the variant:

1) This variant is a compound implementation of twin's factory (suggested above) and Decorator pattern. The intent is to expose a single interface which returns a pre-ready component already decorated based on the parameterized information passed from the client. The client doesn't know or nothing to deal with the available / suggested rules how to make mix fabric, apply styles to buttons. They simply pick and wear it. This variation introduces a placeholder class which holds references to all the compatible decorators which dress up the component from inner to outer.
2) This variant differs from canonical representation in terms of place holder decorator classes.

3.3 Read Only and Write Only Adapters

Adapters are implemented as post development strategy when we really want to work with classes. Classes have possibly data and functions. In current implementations of adapter we have seen adapter use functionality but there can be a situation where possibly only data is needed which can be read-only or write-only.

Motivation

Data is a vital asset in every application; all the access grants are ensured that it is accessed through proper channel. Similarly there are some classes which do not expose their internal logic to the outside world. They act as data providers for other applications, so read-only, write-only properties are exposed to be used. Adapters for such adaptation merely requires the Adaptee to be associated as a reference, instead the data providing property is invoked directly in the adapter request method to pass the required data to the target interface.

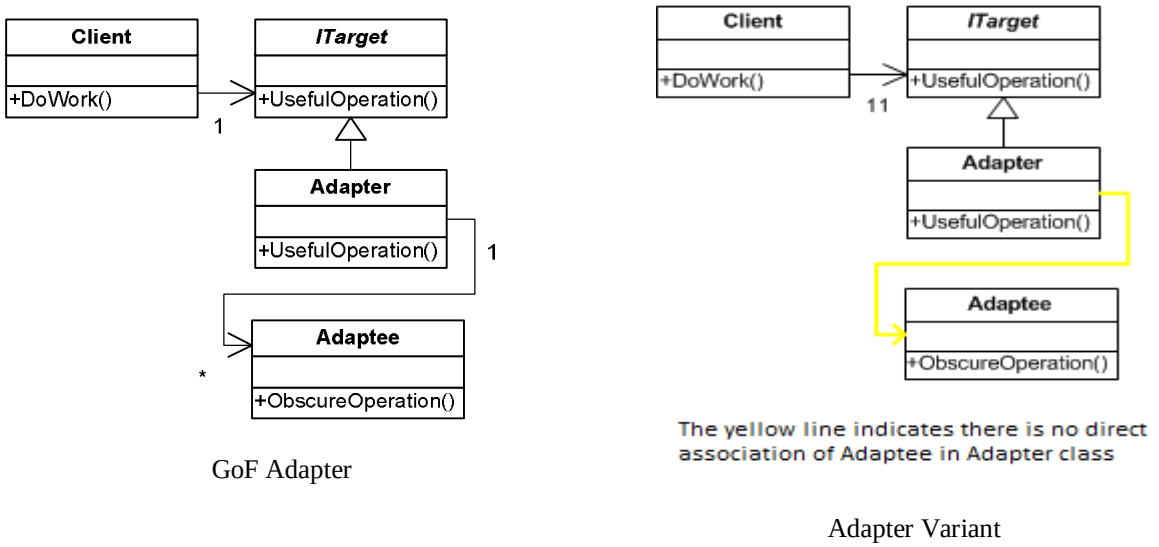


Figure 5. Adapter Pattern and Adapter Variant

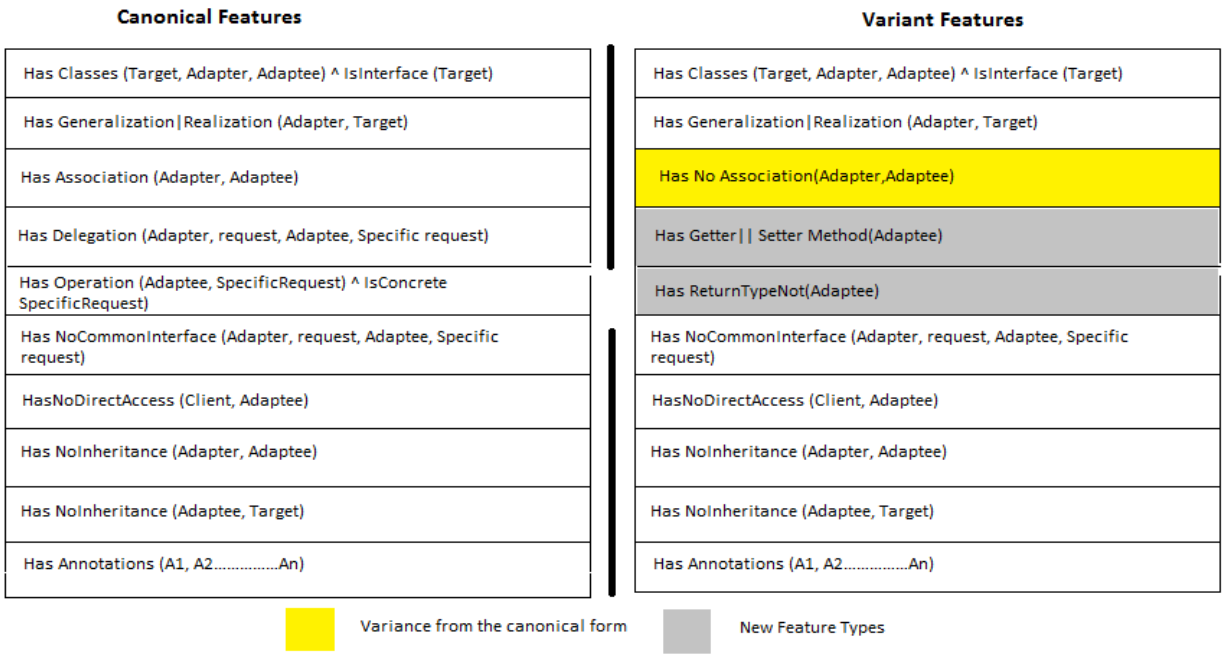


Figure 6. Feature types of Adapter Pattern (left) and read-only / write-only adapters(right)

Critical Overview

Following is a point of discussion:

1) This variance eliminates the use of Adaptee reference in the Adapter class. The specified method of the Adapter directly invokes the data elements from the Adaptee class which can be of sole instance scope. Apparently these adapters' exposes read or write methods which can be used to accept of retrieve data.

3.4 Interchangeable Proxy

Proxy pattern introduces a surrogate object that holds the responsibility of delegating costly calls to heavy weight objects. Complex objects frequent functionality which is used time by time is placed in a lightweight object to lighten the usage and control the access to the original object

Motivation

This scenario can be better interpreted as internal and external knowledge base. Software applications have a predefined local help which covers the commonly occurred problems and which normally reside on the mass storage. On the other hand for advance consultations the online knowledge base is used to seek more complex problems solutions via world wide web. Proxy Implementation can be tweaked into a situation where Real Subjects can be many in number. A common example that fits in the frame is Doctor-Patient interaction. A patient seeks consultation when he/she pays a visit for a problem ranging from fever, headache, food indigestion to skin rashes and orthopedic. The general physician is smart enough to dealt with such ailing, but the problem occurs when symptoms of serious diseases are diagnosed e.g. for heart problems one need to consult heart specialist, for skin rashes skin specialist etc. A point has been observed that if the symptom of the disease is according to his/her expertise he/she will prescribe a treatment himself but if the sufferings are beyond his expertise he would refer to some specialist of that area. Usually general physicians are cheaper while Specialists charge more because of their specialty.

The proxy pattern can be operated in the similar way as discussed above with a slight variation in the number of real objects. A proxy object corresponds with many real subjects according to the above example. A proxy object plays the role of a physician and the Real Subjects are Specialists which could be many.

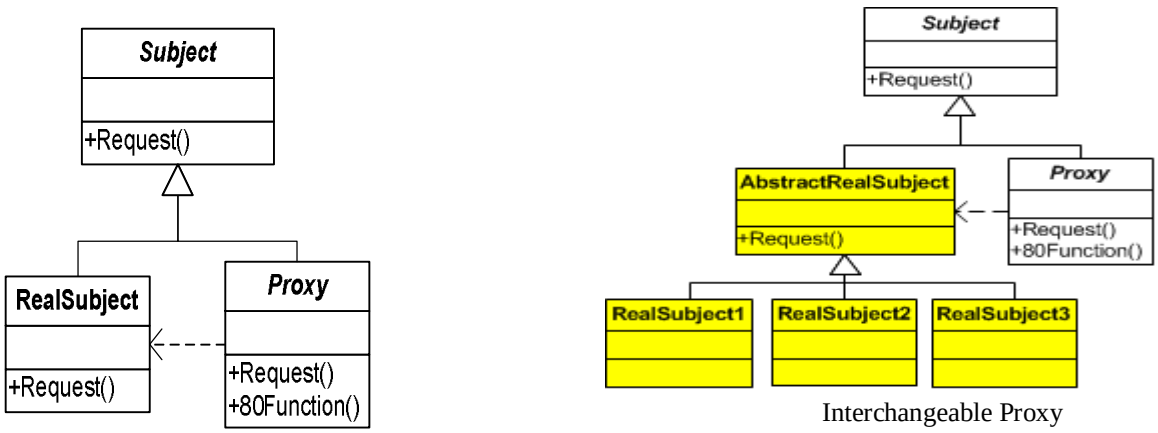


Figure 7 Proxy Pattern and Interchangeable Proxy variant

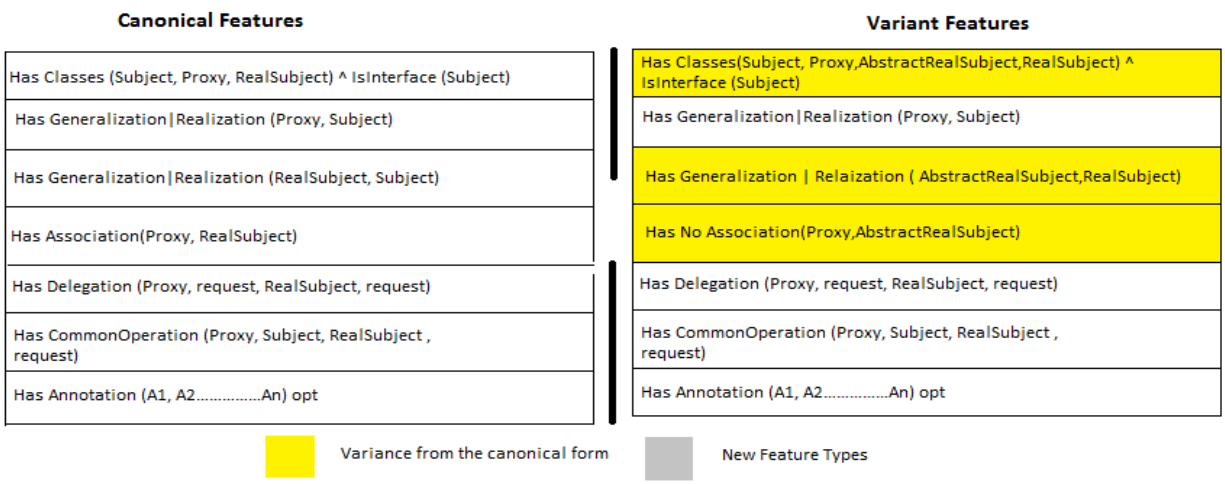


Figure 8. Feature types of Proxy Pattern (left) and Interchangeable proxy(right)

Critical Overview

This variation is a compound implementation of factory method pattern and proxy pattern. The proxy object has

the immunity to control access to authorized situations only by interchanging the Real Subjects. Justifying the criteria for the existence of multiple subjects, the factory method pattern is used and the client is unaware of which subject to create. In our example we categorized the Real Subjects based on specific characteristics. The selection of the Real Subject happens with the occurrence of a certain characteristic and based on that the factory returns the required Real Subject. This variant can also be viewed in the context of load balancer where a number of replica database or web servers are available for entertaining multiple client requests; proxy of certain type can forward requests to multiple instances of database servers or reporting servers interchangeably one at a time in case of a failure.

4.DETECTION AND EVALUATION

The variants discussed in above section are implemented using Java programming language. These variants are further analyzed and tested using three well known state of the art design pattern recovery tools[10 11 7]. Table 1 shows the results of tools on four variants. It is clear from Table 1 that first two tools are not able to recognize these variants only with the exception of Self-Dressed Decorator variant recognized by DPD[11]. Our prototyping tool [7] was also not able initially to recognize all of these variants. The new feature types were added in the patterns definition catalogue used by our tool to recognize these variants. Although, there is no end on the number of variants but our prototyping tool can handle new variants due to its flexibility by changing parameters in existing feature types or adding new feature types. The other two applied tools are not able to recognize these variants due to hard coded algorithms used in these tools for the detection of patterns. We modified our pattern definitions used in our customizable prototype tool [7] to recognize these variants. Design patterns recovery tool developers can use our definitions of these variants while developing new tools.

Table 1. Results of detected variants

S.No	Variant	Fujaba[10]	DPD[11]	Prototype[7]
1	Abstract Factory (Twins)	No	No	Yes
2	Self-Dressed Decorator	No	Yes	Yes
3	Read-Only / Write Only Adapter	No	No	Yes
4	Interchangeable Proxy	No	No	Yes

5.CONCLUSION AND FUTURE WORK

The topic of design pattern recovery is still challenging and open due to low accuracy of existing pattern recovery techniques and tools. The disparity in the results of different tools on the same examined systems motivates researchers to explore new techniques for the accurate recovery of design patterns which are helpful for comprehension of legacy applications. Variation is one of the major problems which affect accuracy of design pattern recovery techniques and tools. We presented four new variants of patterns in this paper which are unique in characteristics that state of the art tools are not able to recognize these variants. The developers can use these variants to solve different generic problems which are closely related with standard representation of these variants. The research groups working on new design pattern recovery tools can include new features highlighted in these variants while developing new tools. The future work will focus on investigating possible variants of other patterns and updated catalogue of all variants with feature types will be available on web.

REFERENCES

1. Gamma, M. , R. Helm, R. Johnson andJ. Vlissides, 1995.Design Patterns: Elements of Reusable Object Oriented Software.Addison-Wesley Publishing Company.
2. Nucleus Research Inc, 2009. Nucleus Research Report: Microsoft Patterns and Practices.Nucleus Research Inc. 100 State Street Boston, MA 02109.
3. Riehle, D., 2011. Lessons Learned from Using Design Patterns in Industry Projects, Transactions on pattern languages of programming II, pp 1-15.
4. Taibi, T. andD. Chek Ling Ngo,2003. Formal specification of design pattern combination using BPSL. Journal of Information and Software Technology., 45(3) : 157–170.

5. Eden, A.H. , 1999. Precise Specification of Design Patterns and Tool Support in Their Application, PhD thesis, University of Tel Aviv.
6. Mapelsden, D., Hosking, J. and Grundy, J, 2002. Design Pattern Modelling and Instantiation using DPML. In Proceeding of TOOLS Pacific, Sydney, Australia.
7. Rasool, G., and P. Mader, 2011. Flexible Design pattern detection based on feature types. Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference, pp : 243 - 252.
8. Rasool, G., M. Umair and R. Talib, 2012. Extended Visualizatin of Overlapping in Recognized Design Patterns. Journal of Basic and Applied Scientific Research, 2(9), pp: 9080-9087.
9. Rasool, G., I. Philipow and P. Maeder, 2010. Design pattern recovery based on annotations. Journal of Advances in Engineering Software, 41(4) : 519-526.
10. Fujaba Home Page : http://wwwcs.uni-paderborn.de/cs/fujaba.
11. DPD Home Page:http://java.uom.gr/~nikos/pattern-detection.html.
12. Stencel, K. and P. Wegrzynowicz, 2008. Implementation Variants of the Singleton Design Pattern, Lecture Notes in Computer Science, Volume 5333, pp: 396-406.
13. Bayley, I., and H. Zhu 2009. Formal Specification of the Variants and Behavioural Features of Design Patterns, Journal of Systems and Software, pp: 2009.
14. Bouhours, C., H. Leblac andC. Percebois 2006. Structural variants detection for design pattern instantiatio. International Workshop on Design Pattern Detection for Reverse Engineering (DPD4RE), Benevento, Italy.
15. Vojislav D., Radonjic and J.P. Corriveau, 2005. Making Patterns Better Design Tools: Requirements Analysis for a Family of Navigators for Design Pattern Catalogs. In Proceeding of IASTED Conference on Software Engineering Innsbruck, Austria, pp: 349-354.
16. Fontana F.A., S. Maggioni and C. Raibulet 2011. Design patterns: a survey on their micro-structures. Journal of Software Maintenance and Evolution: Research and Practice, pp: 1-29.