International Journal of Engineering & Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444
International Journal of Engineering and Technology Volume 1 No. 3, December, 2011

Grasp and GOF Patterns in Solving Design Problems

R.M.Noorullah
Professor, Department of CSE, Amina Institute of Technology, Shamirpet, Hyderabad

ABSTRACT

Pattern is a named problem and solution pair that applied in a particular context. Many patterns, given a specific category of problem, guide the assignment of responsibilities to objects. To solve various design problems RDD, GRASP and GOF patterns are used. In the present paper study of GRASP and GOF Patterns to solve various design problems are presented. A critical study proposed for utilization of these patterns in solving design problems and relationship among these patterns were also presented with the help of design tools.

Key words: *Pattern, RDD, GRASP, GOF Pattern, Creator*

1. INTRODUCTION

Design Patterns are descriptions of communicating objects and classes that are customized to solve a design problem at a particular context. It identifies the participating classes and instances, their roles, collaboration and responsibilities. A popular way of thinking about the design of software objects and components, is in terms of responsibilities, roles, and collaborations. This approach is called Responsibility Driven Design (RDD). In RDD, software objects have doing and knowing responsibilities. Responsibilities are assigned to classes of objects during object design. RDD leads to view an Object Oriented Design as a communication of collaborating responsible objects. The translation of responsibilities into classes and methods is influenced by the granularity of the responsibility. RDD also includes the idea of collaboration. Responsibilities are implemented by means of methods that either act alone or collaborate with other methods and objects. The GRASP Patterns are a learning aid to help us understand essential object design and apply design reasoning in a methodical, rational and explainable way. This approach understands in using design principles based on patterns of assigning responsibilities. The GRASP is a learning aid to structure and name the principles which will be represented with UML interaction diagrams. In section 2 Object Oriented Design Principles, in section 3 Description of common causes of design problems and proposed solutions, in section 4 Step by step approach to be used, in section 5 Applying GRASP and GOF Design Patterns and relationship, in section 6 Implementation of GRASP Patterns in solving design problems with case studies and concluded in section 7.

2. OBJECT ORIENTED DESIGN PRINCIPLES

In Object Oriented Designing some of common design principles and assigning responsibilities for these principles are considered. They are 1. Considering who creates an object? Assign class Y that responsible to create an instance of class X, if Y contains A, Y aggregates A, Y records X. 2. Who knows about an object? Assign responsibility to the information expert. 3. How to reduce an impact of change? Assign a responsibility to the coupling that remains low. 4. Which object controls a system operation? Assign a responsibility to an object to represent overall system or represent a use case scenario. 5. How to keep objects to support low coupling? Assign responsibility so that cohesion remains high.6. Who is responsible when behavior varies by type? Assign responsibility for the behavior by using polymorphic operations. 7. Who is responsible when objects are desperate? Assign a highly cohesive set of responsibilities to an artificial behavior. 8. How to assign responsibilities to avoid direct coupling? Assign the responsibility to an intermediate object, so that they are not directly coupled. 9. How to assign responsibilities to objects so that do not have an undesirable impact on other elements? Assign responsibilities to create a stable interface around them.

3. DESCRIPTION OF COMMON CAUSES OF DESIGN PROBLEMS AND PROPOSED SOLUTIONS

Design Problem	Proposed solution	GRASP Patterns(s)	GOF Patterns(s)	Related Patterns
1.To resolve incompatible interfaces	Convert original interface of a component with another interface	Protected Variation indirection polymorphism	Adapter Strategy	Façade
2.To provide better cohesion	Create pure fabrication Object	High Cohesion	Abstract Factory	Singleton

3.To design with exactly one instance of a class	Define a static method of class that provide a singleton.	Information Expert	Singleton	Façade
4To design for varying, but related algorithms or policies	Define each algorithm/policy in a separate class with a common interface	Factory	Strategy	Low Coupling Polymorphism
5.To design a group or composition structure of objects as a non composite object	Define class for Composite	Protected Variation Polymorphism	Composite	Strategy Command
6.To design a common, unified interface to a set of implementations	Define a single point of contact to the subsystem	Low Coupling Indirection	Façade	Singleton
7.To design publisher which wants to maintain low coupling to subscriber	Define subscriber or listener interface	Publish – subscribe	Observer	Controller Polymorphism
8.Recovery from remote service failure	Location transparency using service look up and local service partial replication	Protected Variation	Adapter	Proxy
9.Failure to a local service from the product information	Add a level of indirection with a surrogate proxy object	Indirection	Proxy	
10.To access external physical device	Read from a system property to load a set of classes and return instances	Controller	Factory Method	
11.To create families of related classes that implement a common interface	Define a concrete factory class for each family of things to create.	High Cohesion	Abstract Factory	
12.To create persistent Objects	Persistence service from a persistence framework	Framework	Template Method	State Command
13.To access a persistence service	Provide a unified interface to a subsystem	Low Coupling Indirection	Facade	
14.To Mapping objects	Use other objects to do mapping for the persistent objects	Database Broker Database Mapper		
15.To maintain materialized objects in a local cache	Use database mapper for maintaining its cache	Cache management pattern		
16.To design persistent object does not cause immediate dbase update	Create state classes for each state		State	
17.To design a transaction	Make each task a class that implements a common interface		Command	
18.To design lazy materialization	Proxy another object that meta dbase real subset		Virtual Proxy	

4. STEP BY STEP APPROACH TO USE A RIGHT DESIGN PATTERN TO SOLVE DESIGN PROBLEM

Once we have picked a design pattern to solve a design problem apply the following step-by-step procedure effectively:

Step1: Pay particular attention to the applicability and consequences of selected design pattern(s) to solve design problem.

2: Understand the classes and objects in the pattern and relationship among them by knowing their participants, structure and collaboration.

3: Implement the pattern with specific code.

4: Name Pattern Participants used in solving design problem.
5: Identify existing classes that pattern will affect and modify them accordingly.
6: Use responsibilities and collaborations associated with each operation involved.
7: Implement the operations to be carry out the responsibilities and collaborations in the selected pattern(s) to solve the design problem.

5. APPLYING GRASP AND GOF DESIGN PATTERNS AND THEIR RELATIONSHIPS

GRASP Patterns are Creator, Info expert, Low coupling, Controller, High cohesion, Polymorphism, Pure fabrication, Indirection, and Protected Variation. Creator Pattern is responsible for creating new instances of some class, useful for assessment of responsibilities, to support low coupling, encapsulation and reusability. Information Expert or Expert pattern is responsible for assigning responsibilities to object, to distribute behavior across classes. Low Coupling Pattern measures how strongly one element is connected to or relies on other elements. An element with low coupling is not dependent on too many other elements. Controller Pattern is responsible for receiving or handling a system operation message. It is a part of the domain layer and controls the handling of work request. It is used to choose a suitable receiver in the pure domain layer. High Cohesion pattern is a measure of how strongly related and focused the responsibilities of an element. Polymorphism pattern is responsible when behavior varies by type. Pure Fabrication pattern assign responsibilities only to domain layer software classes leads to problems in terms of poor cohesion or coupling. Indirection pattern is used to avoid direct coupling between two or more things and reuse potential remains higher. Protected Variations used to design objects, subsystems and systems so that variations or instability in these elements do not have an understandable impact on other elements.

GOF Patterns are Abstract Factory Provide an interface for creating families of related or dependent objects, Adapter convert the interface of a class into another interface, Bridge Decouple an abstraction from its implementation, Builder Separate the construction of a complex object from its representation, Chain of Responsibility avoid coupling the sender of a request to its receiver, Command encapsulate a request as an object, Composite compose objects into tree structures to represent part-whole hierarchies, Decorator attach additional responsibilities to an object dynamically, Facade provide a unified interface to a set of interfaces in a subsystem, Factory Method define an interface for creating an object, Fly weight use sharing to support large numbers of fine-grained objects efficiently, Interpreter define a representation for its grammar, Iterator provide a way to access the elements of an aggregate object sequentially, Mediator define an object that encapsulates how a set of objects interact, Memento without violating encapsulation, capture and externalize an object's internal state, Observer define a one-to-many dependency between objects, Prototype specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype, Proxy provide a surrogate to control access to it, Singleton ensure a class only has one instance, State allow an object to alter its behavior when its internal state changes, Strategy define a family of algorithms, encapsulate each one, and make them interchangeable, Strategy lets the algorithm vary independently from clients that use it, Template Method define the skeleton of an algorithm in an operation, Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure and Visitor represent an operation to be performed on the elements of an object structure.

Most GOF Patterns can be seen as Specialization of a few GRASP patterns. Relationship will exist between GOF and GRASP Patterns. Some of them are represented in the following figures.

1) Adapter Design Pattern use interfaces and polymorphisms to add a level of indirection to varying APIs in other components. Adapter support Protected Variation with respect to changing external interfaces through the use of an Indirection object that applies interfaces and polymorphism. (fig. 5.1)

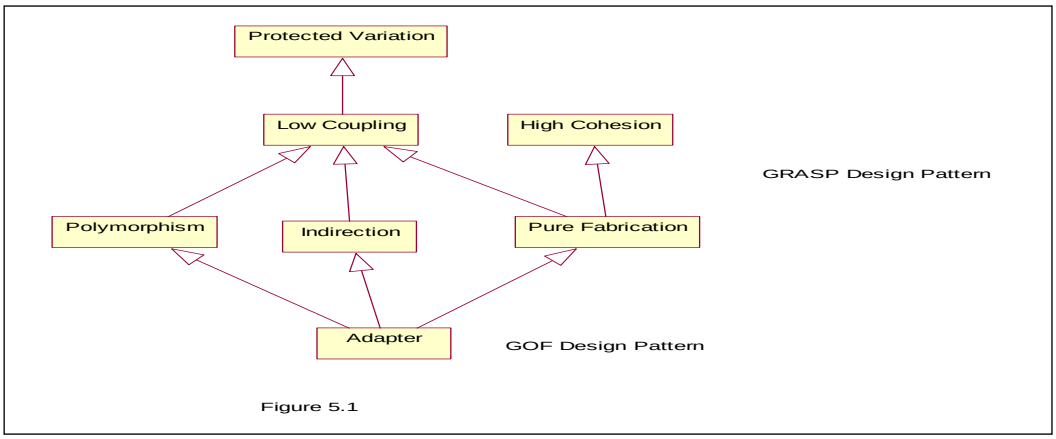

Figure 5.1

2) Abstract Factory Pattern is an application of the GRASP High Cohesion and Pure Fabrication. It allow introduction of object caching or recycling. (fig.5.2)

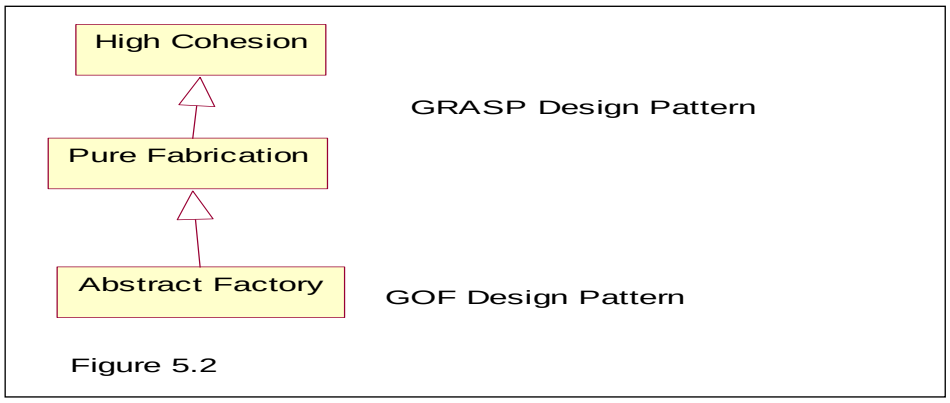

Figure 5.2

3) Strategy Pattern is an application of GRASP polymorphism, low coupling and Protected Variation. It defines family of algorithms encapsulate each one. (fig 5.3)

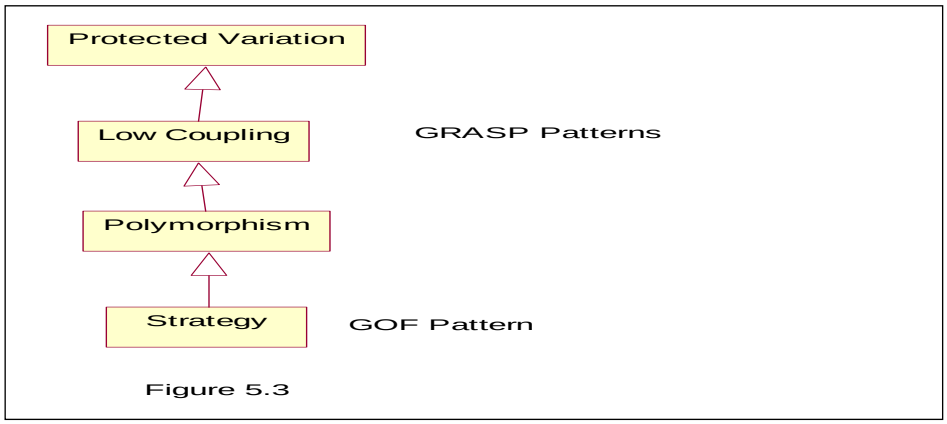

Figure 5.3

4) Composite Pattern is an application of GRASP polymorphism and Protected Variation. It represents part-whole hierarchies. (fig. 5.4)

5) Façade Pattern is an application of GRASP Indirection, Low coupling and Protected Variation. It provides a unified interface to a set of interfaces in a sub system. (fig. 5.5)

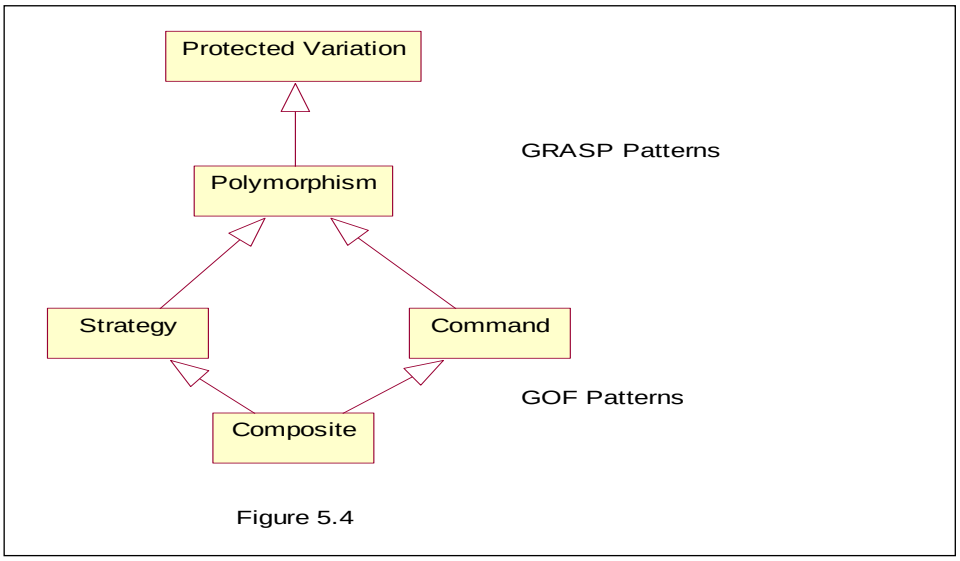

Figure 5.4

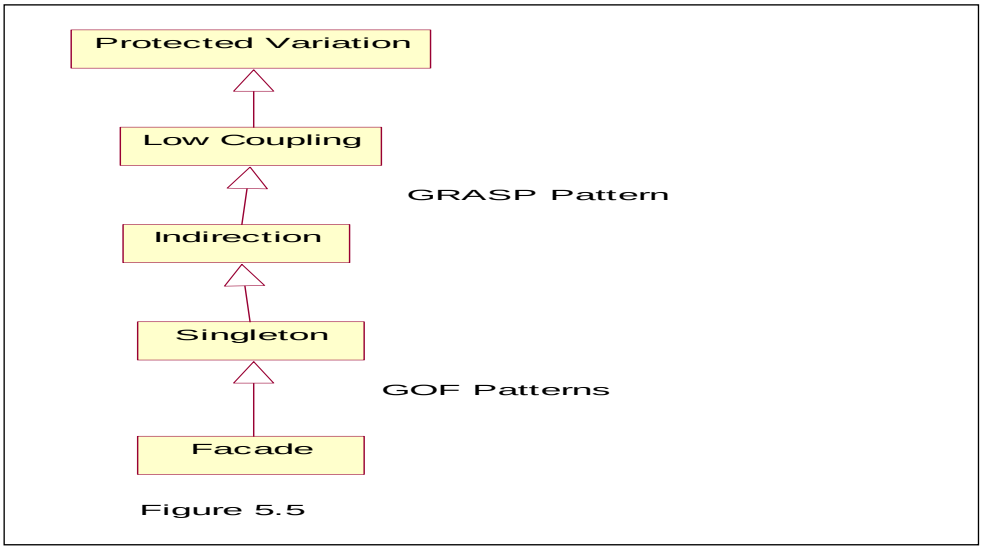

Figure 5.5

6) Observer Pattern provides a way to loosely couple objects in terms of communication. It is an application of Controller, Polymorphism, Low coupling, Protected Variation GRASP patterns. (fig. 5.6)

7) Command Pattern rely on polymorphism GRASP pattern, which encapsulate request as an object.(fig. 5.7)

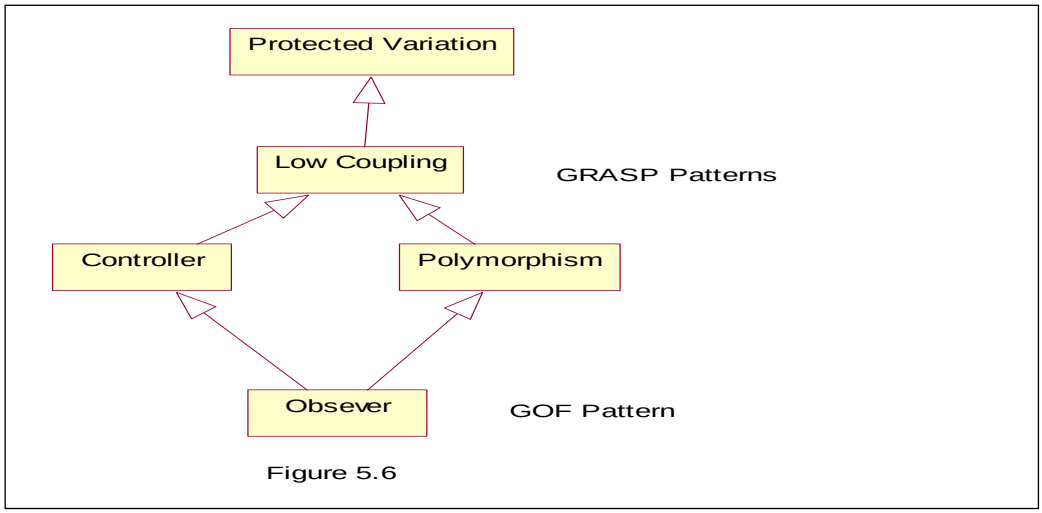

Figure 5.6

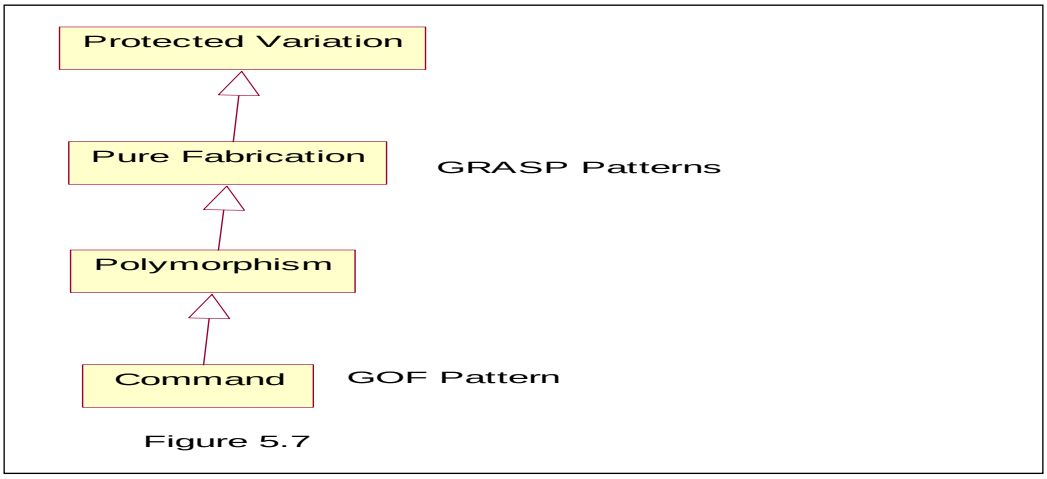

Figure 5.7

8) Proxy Pattern rely on polymorphism GRASP pattern, which provide a surrogate. (fig.5.8)

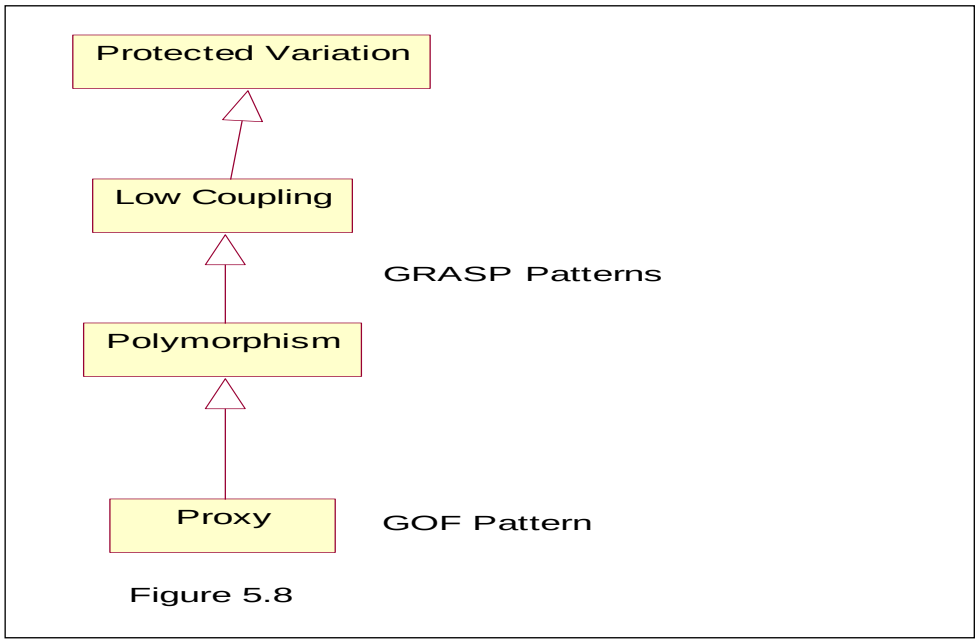

Figure 5.8

6. IMPLEMENTATION OF GRASP PATTERNS TO SOLVE DESIGN PROBLEMS

They form a foundation for designing Object Oriented Systems. Some of design problems are considered and solution for these problems with GRASP Patterns implemented with the help on UML diagrams with Rational Rose software.

1. To create new instances of some class and the design can support low coupling, increased clarity, encapsulation and reusability. This problem can be solved by using Creator Design Pattern. Assign class B the responsibility to create an instance of class A, then it must satisfy one of the following conditions i.e. B contains or compositely aggregates A, B records A, B closely uses A, B is an expert with respect to creating A. B is a creator of A objects. Consider in course registration application, who is responsible for creating courses? By Creator design pattern, we should look for a class that aggregates contains courses class. Partial domain model is shown in fig. 6.1 another case Diagram Editor is shown in fig. 6.1a and other case Lawn Mover is shown in 6.1b

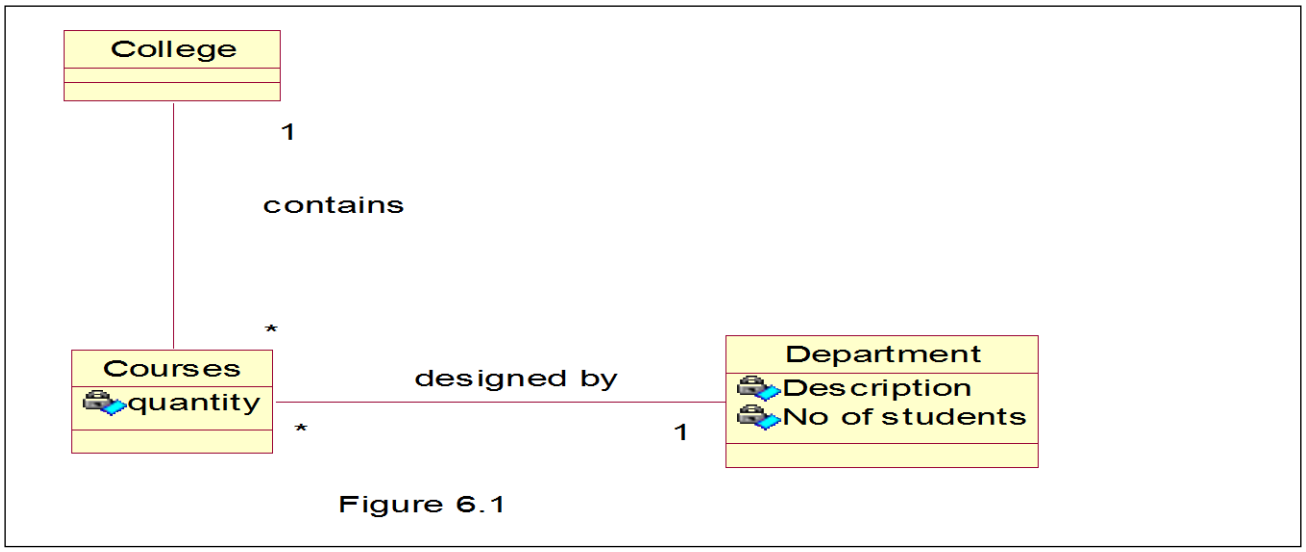

Figure 6.1

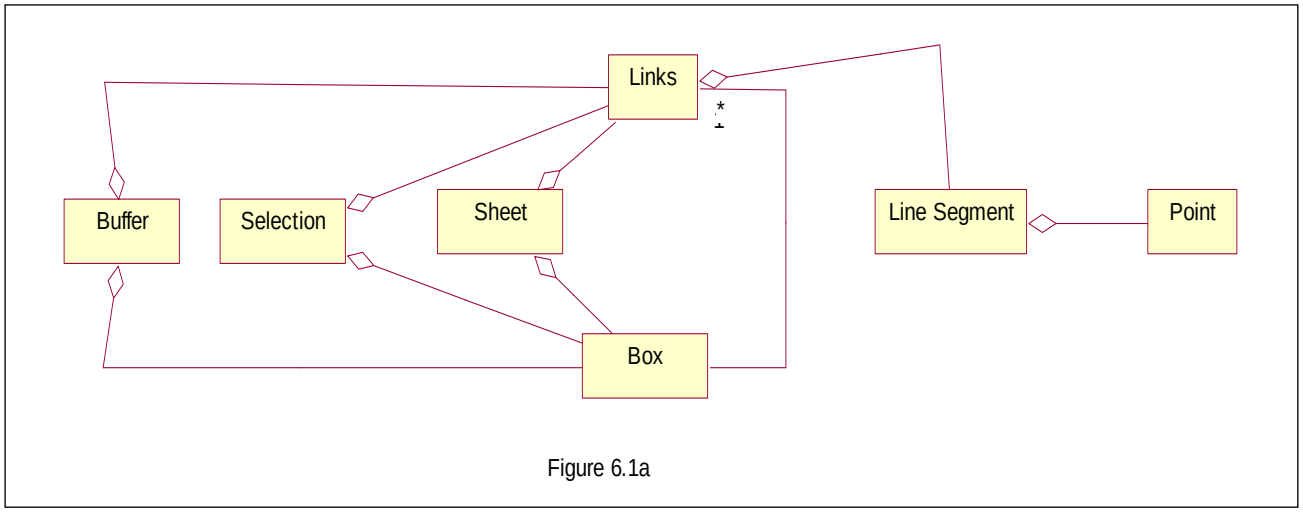

Figure 6.1a

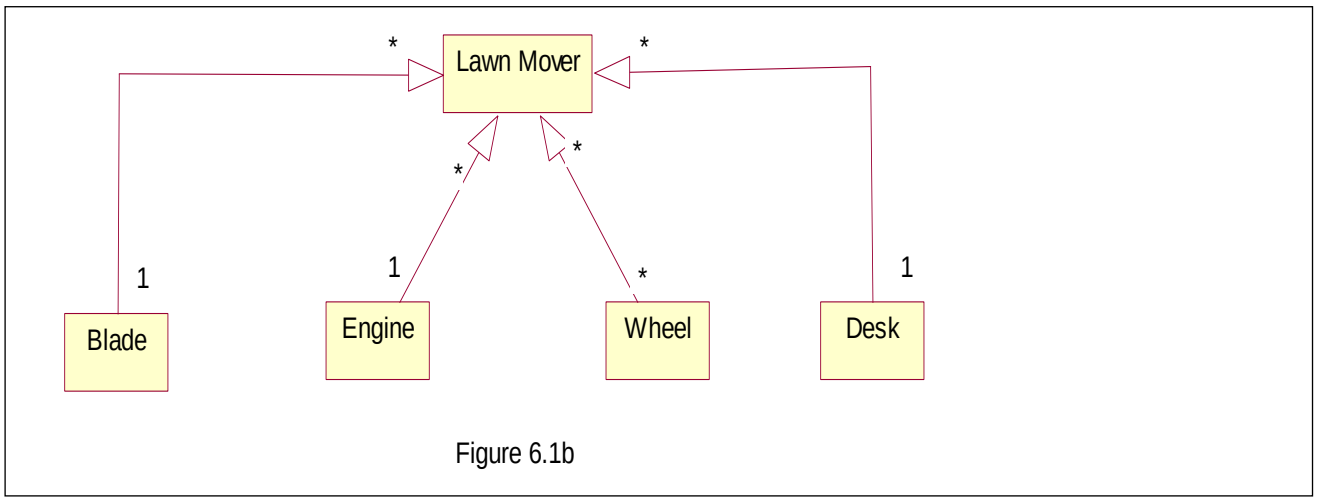

Figure 6.1b

2. Assigning responsibilities to objects by choosing system tend to understand, extend and reuse components in future applications. This problem can be solved using Information Expert design pattern to fulfill the responsibility. Consider the next generation POS application, some class need to know grand total of sales, for this partial domain model is shown in fig. 6.2 and another Bank application to get amount is shown in fig. 6.2a.

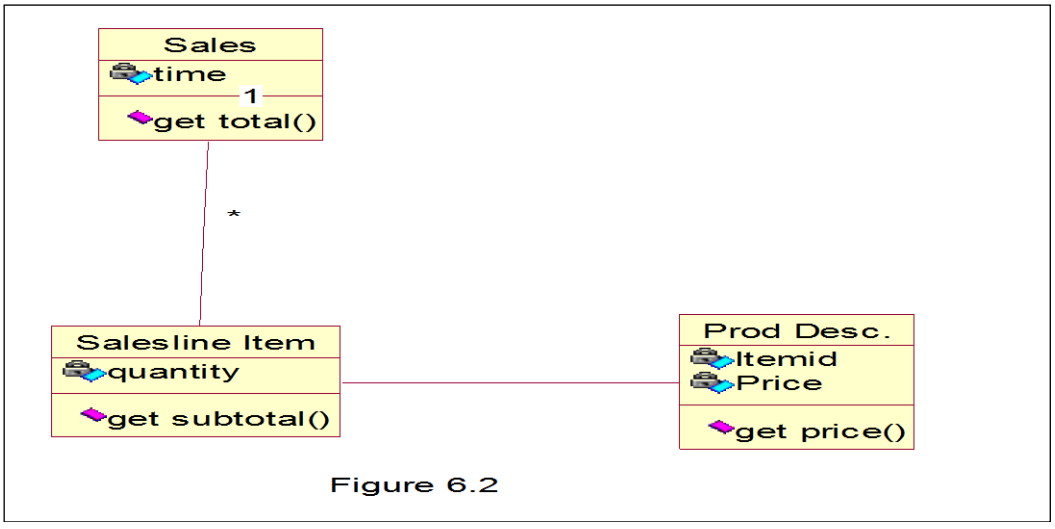

Figure 6.2

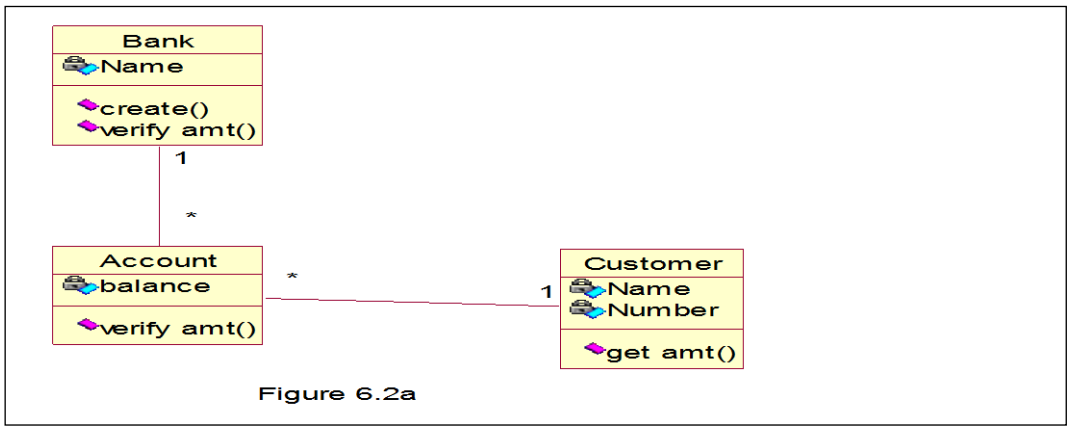

Figure 6.2a

3. To support low dependency and increased reuse facility. This problem can be solved by using Low Coupling design pattern to evaluate alternatives. Consider case study of Diagram Editor in which class diagram drawn in fig. 6.3 to create a collection instance and associated with Box class by using how coupling design pattern which will have low coupling.

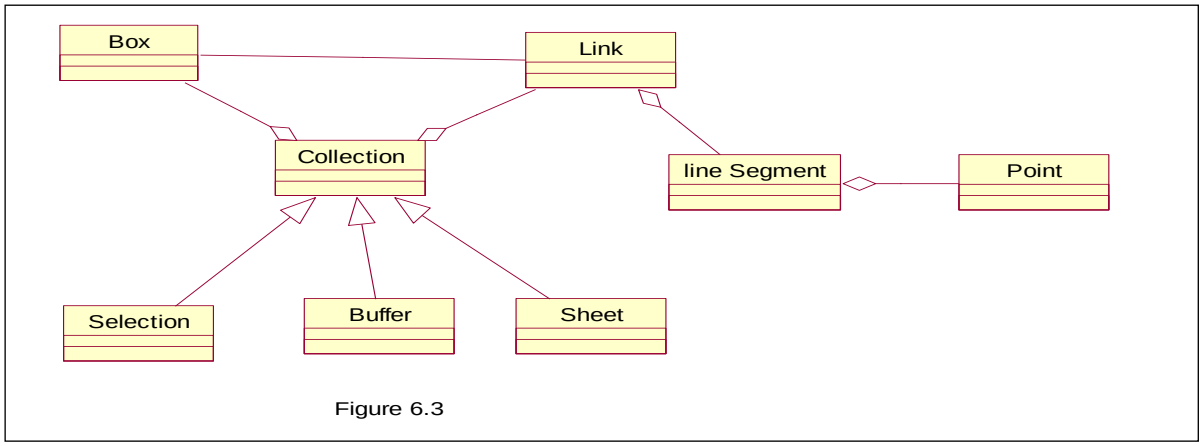

Figure 6.3

4. What first object beyond the UI layer receives and co-ordinates a system option? This problem can be solved by assigning responsibility to a class reporting by using Controller Design Pattern by representing the overall systems or by representing use case scenario or by UI framework or by MVC pattern. Consider for representation of class model for Desktop Publishing shown in fig. 6.4.

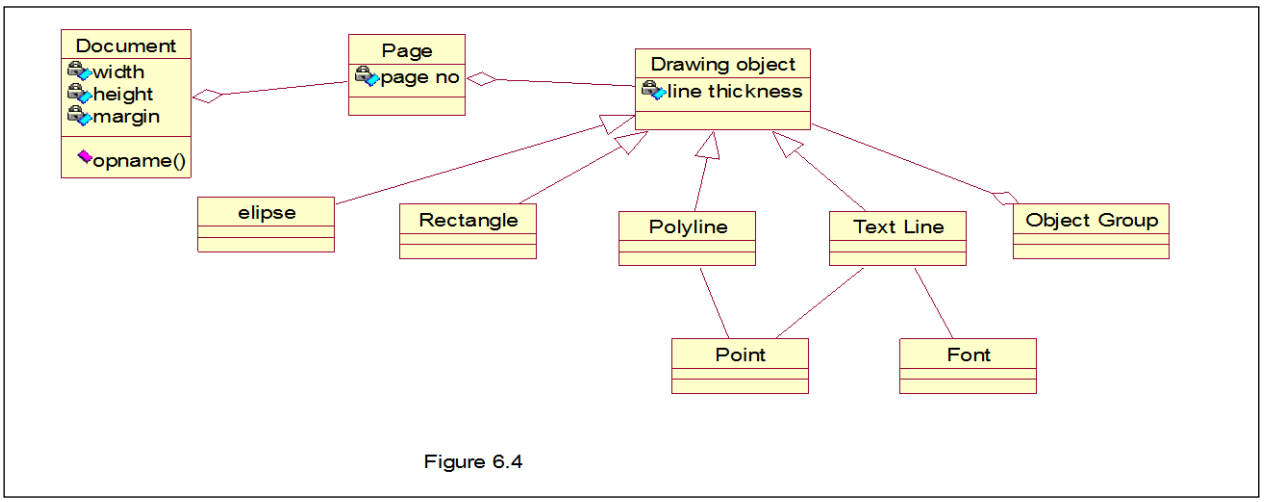

Figure 6.4

5. How to handle alternatives based on type or how to create pluggable software components. This problem can be solved by using Polymorphism Design Pattern, by assigning responsibility for the behavior. Consider Fund Transfer application that must be supported by using ATM and Bank Counter; the system need to be able to integrate with different technique i.e. Net Banking. Each mode has different interface represented in fig. 6.5

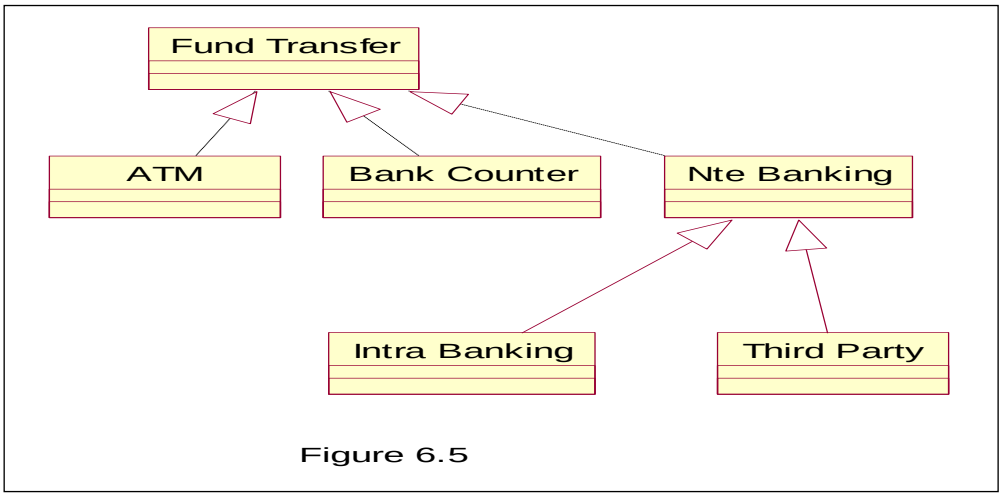

Figure 6.5

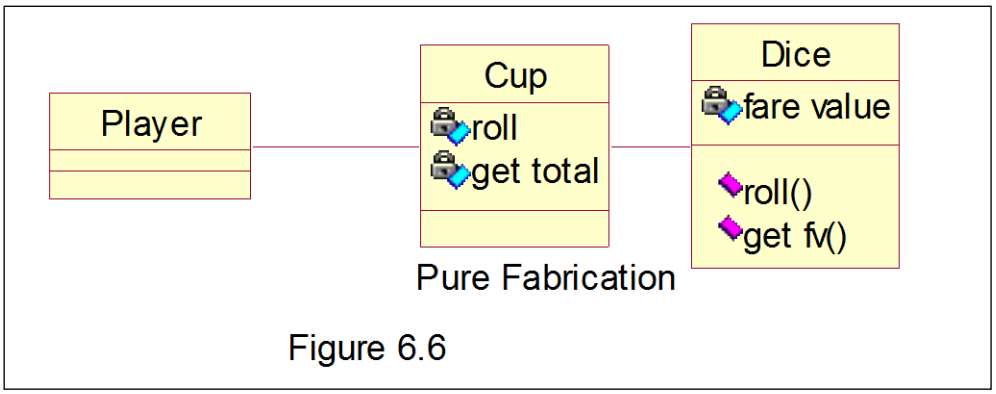

Figure 6.6

6. Not to violate High Cohesion and Low coupling, but solution offered by expert are not appropriate. This problem can be solved by using Pure Fabrication Design Pattern, which assigns a highly cohesive set of responsibilities to an artificial class. Consider Monopoly Dice game in which there is no cup for the dice in Monopoly, many games use a dice cup. In this a Pure Fabrication Pattern Cup is used which is represented in fig. 6.6.
7. To avoid direct coupling between two or more things or to decouple objects. This problem can be solved by using Indirection design pattern by assigning responsibility to immediate object. The intermediary creates an Indirection between the other components. Consider decoupling the Sale from Relational database services through Indirection of Persistence Storage Class, acts as intermediary between the Sale and Databases represented in fig. 6.7

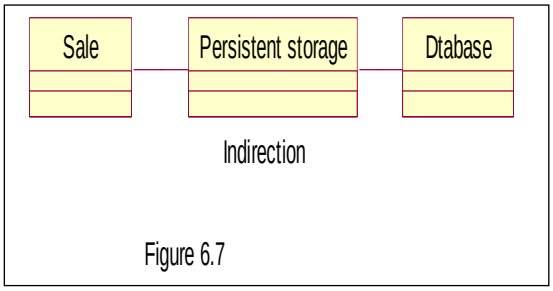

Figure 6.7

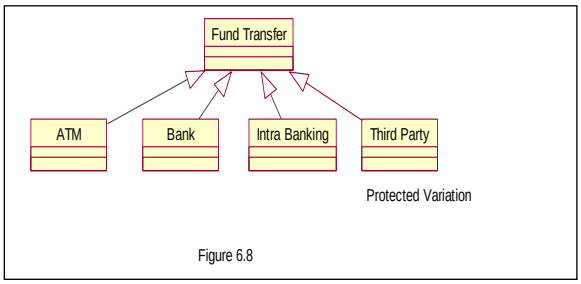

Figure 6.8

8. To design objects, subsystems and system that variation does not have an undesirable input on other elements. This problem can be solved by using Protected Variation design pattern which identify points of predicted variation and assign responsibilities to create a stable instance. Consider a case of Fund Transfer by Third Party in future is different APIs of other types of fund transfer which is represented in fig. 6.8

7. CONCLUSION

The purpose of this paper is to study utilization of GRASP and GOF Design Patterns to solve various design problems of Object Oriented Design. Some of them are identified and proposed solution suggested with relevant design patterns. Some case studies in designing real systems are considered with some design problems and solutions proposed by using GRASP design patterns are represented with UML diagrams by using Rational Rose software. Relationships between GRASP and GOF design patterns also presented.

REFERENCES

[1] European Conference on Object Oriented Programming, pages 21-35, Kaiserlautern, Germany, July1993.

[2] European Conference on Object Oriented Programming, pages 139-149, Bologna, Italy, July1994.

[3] Object Oriented Analysis and Design with Applications, Benjamin/Cummings, 1994, Second Edition.

[4] ACM Transactions on Programming Languages and Systems, pages 343-387, October 1981.

[5] An Object Oriented System in C++ Communications of the ACM, pages 117-126, September 1993.

[6] ACM User Interface Software Technologies Conference, pages 92-101, Snowbird, UT, October 1990.

[7] Software Reusability, Volume II: Applications and Experience, pages 269-287, Reading MA, 1989.

[8] Object Oriented Programming Systems, Languages and Applications Conferences Proceedings, pages 214-223, Portland, November 1986.

[9] Proceedings of the USENIXC++ Conference, pages 51-63, Washington, April 1991.

[10] Object Oriented Design Heuristics by Reil.

[11] Object Models by Coad.

[12] Designing Object Oriented Software by Brock.

[13] Using Pattern Languages for Object Oriented Programs, AFIPS Conference Proceedings, Pages 43-47,Beck K and Cunningham W 1987.

[14] Introduction and overview of the Multics Systems AFIPS Conference Proceedings 27, pages 185 – 196, Carbato F and Vyssotesky V

[15] The Unified Modeling Language Reference Manual 2e, Addison-Wesley, Rumbaugh J , Jacobson and Booch G 2004.