Department of Information Technology, State Polytechnic of Malang

OBJECT ORIENTED ANALYSIS AND DESIGN

GRASP PATTERN RESUME

NAME : MUHAMMAD ALIF ANANDA

NIM : 1941720078

CLASS : TI – 2H

GRASP PATTERN

What is GRASP Patterns ?

General Responsibility Assignment Software Patterns (or Principles), abbreviated GRASP, is a set of "nine fundamental principles in object design and responsibility assignment. The different patterns and principles used in GRASP are controller, creator, indirection, information expert, low coupling, high cohesion, polymorphism, protected variations, and pure fabrication. These patterns solve some software problem common to many software development projects. These techniques have not been invented to create a new ways of working, but to being better document and standardize old, tried and tested programming principles in object oriented design

Whats the purpose of GRASP Patterns ?

As a tool for software develpers, GRASP provides a means to solve organizational problems and offers a common way to speak abut abstract concepts. The design pattern sets responsibilities for objects and classes in object-oriented program design.

How many Principles in GRASP Patterns ?

With these problems and solutions well defined, they can be applied in other similar instances. GRASP assigns seven types of roles to classes and objects in order to make for clear delineation of responsibilities. These roles are:

- Controller
- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Polymorphism
- Protected Variations
- Indirection
- Pure Fabrication

The Controller

The Controller is responsible for handling the requests of actors. The Controller is the middle-man between your user clicking “Send” and your back-end making that happen. The Controller knows how to interpret the actions of user-interfaces, and how to connect those actions to behaviours in your system. We can imagine a controller as the steering wheel in a car, it connects the intentions of a driver to the actions of the vehicle.

Information Expert

As our systems grow, we might find that we are putting too much logic into our controllers. This results in what we call “bloated controllers”. Bloated controllers imply tight coupling in our system, which is bad. The Expert Pattern solves this by encapsulating information about a task into a distinct class.
This module knows precisely how to authenticate a user, and the Controller need only delegate the authentication request to this module to know that authentication will be handled correctly.

Creator

The Creator takes the responsibility of creating certain other objects. This pattern is probably the easiest to understand conceptually. There are a number of reasons why a class may take the responsibility of creating another. These decisions will be largely established in the initial design of the system, and other documents such as UML diagrams will guide and inform the Creator pattern.
An example of this is a printing-press. The Creator Pattern allows for a lot of other best-practices to fall into place, such as dependency injection and low-

coupling. The Creator Pattern can be used to enforce the logical design of a system.

High Cohesion

High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of low coupling. High cohesion means that the responsibilities of a given element are strongly related and highly focused. Breaking programs into classes and subsystems is an example of activities that increase the cohesive properties of a system. Alternatively, low cohesion is a situation in which a given element has too many unrelated responsibilities. Elements with low cohesion often suffer from being hard to comprehend, reuse, maintain and change.

Low Coupling

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. Low coupling is an evaluative pattern that dictates how to assign responsibilities for the following benefits:

- lower dependency between the classes,
- change in one class having a lower impact on other classes,
- higher reuse potential.

Polymorphism

According to the polymorphism principle, responsibility for defining the variation of behaviors based on type is assigned to the type for which this variation happens. This is achieved using polymorphic operations. The user of the type should use polymorphic operations instead of explicit branching based on type. Problem: How to handle alternatives based on type? How to create pluggable software components? Solution: When related alternatives or behaviors vary by type (class), assign responsibility for the behavior—using polymorphic operations—to the types for which the behavior varies. (Polymorphism has several related meanings. In this context, it means "giving the same name to services in different objects".)

Protected Variations

It is necessary to design the system so that changes in some of its elements do not affect others. As a solution, it is proposed to identify points of possible changes or instability and assign responsibilities in such a way as to ensure the stable operation of the system. In fact, this is not a pattern, but a goal achieved by following the rest of the patterns.

Indirection

The indirection pattern supports low coupling and reuses potential between two elements by assigning the responsibility of mediation between them to an intermediate object. An example of this is the introduction of a controller component for mediation between data (model) and its representation (view) in the model-view control pattern. This ensures that coupling between them remains low.

Problem: Where to assign responsibility, to avoid direct coupling between two (or more) things? How to de-couple objects so that low coupling is supported and reuse potential remains higher?

Solution: Assign the responsibility to an intermediate object to mediate between other components or services so that they are not directly coupled. The intermediary creates an indirection between the other components.

Pure Fabrication

Low coupling and high cohesion must be ensured. For this purpose, it may be necessary to synthesize an artificial essence. The Pure Fabrication pattern suggests that you shouldn’t hesitate to do this. As an example, consider the facade to the database. This is a purely artificial object that has no analogues in the subject area. In general, any facade is Pure Fabrication (unless it is, of course, an architectural facade in the corresponding application).