www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 4 Issue 12 Dec 2015, Page No. 15114-15116

Design Patterns' Model for Application Development in Object Oriented Languages

Gaurav Pradip Pande ,Akshay Janardan Mendki
Dept. of Computer EngineeringVidyalankar Institute of TechnologyMumbai, India
gaurav.pande@vit.com
Senior software engineer, Capgemini India Pvt Ltd.Mumbai, India.
akshay_mendki@hotmail.com

ABSTRACT

Design patterns are one of the most efficient ways of application development. Many developers are still reluctant to the use of design patterns and development is carried out with not enough attention paid to the reusability and extensibility of the code. Patterns are not to be restricted in the building of integrated development environments and language libraries.
Proposed work presents a model which use combination of multiple design patterns for application development in object oriented programming. The model suggested in this proposed work uses hierarchical abstraction of objects created using factory pattern. Façade pattern is also used which makes functional encapsulation possible. Proposed work also presents the scenarios in which use of patterns is likely to improve the code efficiency and reusability. Along with the factory pattern, singleton pattern may also to be used together in the proposed model for their respective benefits.
Keywords:-Index Terms— design pattern, façade pattern, singleton pattern, factory pattern,

INTRODUCTION

The application development is evolved over the period of time remarkably. The coding used to be done in 'spaghetti' manner in the earlier times. The word spaghetti suggests the linear flow of coding which seldom reused codes previously written and entire focus was given to instantaneous problem solving. [1]

When personal computers were still in their infant phase, functional coding bloomed by means of languages which supported routines to be written and called whenever needed. This considerably reduced the rewriting of same logic in the code and L.O.C was strongly improved. Later structural and object oriented programming came into the picture by which independent coding was actually possible. Writing different objects and then coupling them together is now well accepted way of modular coding.

Application development in object oriented environment is always hard. Moreover if code is to be written in order to keep a way to reuse it later then coding becomes even harder.
Solution designs need not be started from scratch by use of design patterns in which we map our problem to the one previously faced and try to reuse the similar solution partially or completely. [2]

I. ELEMENTS OF DESIGN PATTERN

i. *Pattern Name* – Handle to refer and describe pattern.
ii. *Problem* – Describes context in which pattern is to be applied.
iii. *Solution* – Elements that make up the design and their inter-relationships.
iv. *Consequences* – Results which can be used to evaluate the benefits and cost of applying design pattern.

II. TYPES OF DESIGN PATTERN

i. *Creational Patterns* - abstract the instantiation process
ii. *Structural Patterns* - concerned with ***how*** classes and objects are composed
iii. *Behavioral Pattern* – Describes pattern and the pattern of communication between them. [1]

FACTORY PATTERN

In *factory pattern* concrete classes are not specified, instead of that, families of dependent or similar objects are created. That makes it a *creational pattern*. In object oriented languages, non-concrete objects could be interfaces. Inheritance can be used to get structure and methods of those interfaces into real concrete objects.[3]

III. ABSTRACT FACTORY PATTERN

Abstract factory pattern takes the concept of factory to the next level and has one or more factories which we can call 'parent factories', these factory objects create multiple factories, which later on create concrete objects for the application.[3]

IV. FAÇADE PATTERN.

It can be seen as a higher class, or more aptly, a 'client' class will try to get its work done using some few classes only. These few 'façade' classes may then use many more

DOI: 10.18535/Ijecs/v4i12.3

classes which have different functionalities, as per the requirement.[4]

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

V. PROPOSED WORK

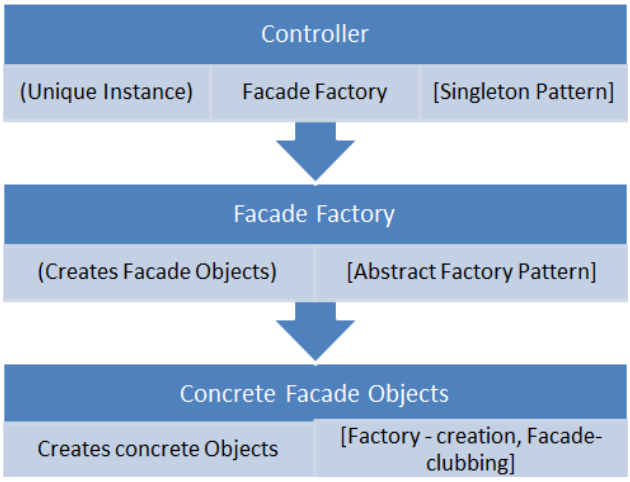
Design patterns have come a long way since the serious research was spawned by Erich Gamma's research paper in 1994. The use of design patterns is mainly done in the framework development or integrated development environments. Typical combinations of the patterns are repeatedly used**.** Small scale applications, especially mobile applications avoid using of design patterns in the development due to their apparent complexity in the implementation.

Our research focused on the use of creational, structural and behavioral design patterns to develop a real life application using integrated development environment.

Factory pattern is to be used for hierarchical development structure to support scalability. Singleton pattern to be used for unique control and façade pattern is to be used to simplify the development. Design patterns provide discipline to the coding by means of naming consistency.

The research presents combination of multiple design patterns. The application is suggested to be developed with hierarchical abstraction structure using factory pattern. The Research also presents the scenarios in which use of patterns is likely to improve the code efficiency and reusability.[5] Along with the factory pattern, singleton pattern and façade pattern are also to be used together for their respective benefits like, better coupling and code reuse, single instance of creation , modular coding.

A. MODEL



Suggested Model

B. MODEL DESCRIPTION

Tier 1

Whenever an application is developed eyeing traditional MVC pattern in which application logic is in the *model* part, user interface is in the *view* part and both these parts are controlled by means of a *controller*, we need to ensure that model and view will interact with each other only by means of the controller and there will be only one instance of the controller.

To ensure that only one controller exists, *singleton pattern* is used. By this, the suggested model will allow only one controller object to be created and thus data integrity will be maintained. Implementation of such single object creation is a crucial task. As this research focuses on the object oriented programming, it would be beneficial to make use of the salient object oriented features of instantiation like, *constructors*.

Tier 2

Façade objects are the one which hide internal working of the application from the user.

In our model level 2 will create abstract façade objects which will simplify the object creation of concrete objects in the next level, ~~in level 2~~ we would create abstract objects which will divide application in according to different functionalities.

For example, abstract objects are to be created like, *text*, *web* and *graphic* in level 2. Now these objects can be used for different purposes in different scenarios**.** User need not be concerned with how different conditions are handles and which concrete objects are to be used. From user's point of view, the interface is made simpler.

Tier 3

Concrete objects are the one which are to be actually used for the work logic. These concrete objects will be using both façade and factory pattern for their implementation.

Creation or instantiation of the objects will be done using factory pattern. The task assignment and getting work done by those objects will be done by means of façade pattern.

E.g. *Text* object can create 'text editor', 'text messenger' etc. using factory pattern.

But this 'text editor' will be used like a façade object. The editor will work for multiple coding languages or for multiple text formats according to the requirements. Creation and method calling of objects like, xyz font, abc layout will be done by this text editor object itself. Internally it will make use of the different classes made for each of them; without giving a notion to the user.

VI. ACKNOWLEDGMENT

We thank Prof. Umesh Kulkarni for his active guidance and support in this research.

VII. REFERENCES

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
"Design Patterns: Elements of Reusable Object Oriented Software". Addison-Wesley Second ISE reprint edition 1999
2. Rachel Cardell-Oliver. "Evaluating the application and understanding of elementary programming pattern" 22nd Australian conference on software engineering, pp 61-66, 2013.
3. DesignPatterns
http://www.tutorialspoint/design_patterns.

DOI: 10.18535/Ijecs/v4i12.3

4. Freeman, Eric T; Elisabeth Robson; Bert Bates; Kathy Sierra (2004). "*Head First Design Pattern, Second Edition, Oct 2004"*

A Xiaoyue Wang, Bin Xu, Rui Gu, "The application of code reuse tec