ABSTRACT FACTORY AND SINGLETON DESIGN PATTERNS TO CREATE DECORATOR PATTERN OBJECTS IN WEB APPLICATION

Vijay K Kerji

Department of Computer Science and Engineering, PDA College of Engineering,Gulbarga, India

vkkerji@gmail.com

ABSTRACT

Software Design Patterns are reusable designs providing common solutions to the similar kind of problems in software development. Creational patterns are that category of design patterns which aid in how objects are created, composed and represented. They abstract the creation of objects from clients thus making the application more adaptable to future requirements changes. In this work, it has been proposed and implemented the creation of objects involved in Decorator Design Pattern. (Decorator Pattern Adds Additional responsibilities to the individual objects dynamically and transparently without affecting other objects). This enhanced the reusability of the application design, made application more adaptable to future requirement changes and eased the system maintenance. Proposed DMS (Development Management System) web application is implemented using .NET framework, ASP.NET and C#.

KEYWORDS

Design Patterns, Abstract Factory, Singleton, Decorator, Reusability, Web Application

1. INTRODUCTION

Design Patterns refer to reusable or repeatable solutions that aim to solve similar design problems during development process. Various design patterns are available to support development process. Each pattern provides the main solution for particular problem. From developer's perspectives, design patterns produce a more maintainable design. While from user's perspectives, the solutions provided by design patterns will enhance the usability of web applications [1] [2].

Normally, developers use their own design for a given software requirement, which may be new each time they design for a similar requirement. Such method is time consuming and makes software maintenance more difficult. Adapting the software application to future requirements changes will be difficult if adequate care is not taken during design [3].

Adopting Design Pattern while designing web applications can promote reusability and consistency of the web application. Without the adoption of the design patterns or wrong selection of them will make the development of web application more complex and hard to be maintained. Thus knowledge in design patterns is essential in selecting the most appropriate

DOI : 10.5121/ijait.2011.1501

patterns in the web application. Unfortunately, the ability of the developers in applying the design pattern is undetermined. The good practice of using design patterns is encouraged and efforts have to be taken to enhance the knowledge on design patterns [1].

Creational design patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented. A class creational pattern uses inheritance to vary the class that's instantiated, where as an object creational pattern will delegate instantiation to other objects. Creational patterns become important as systems evolve to depend more on object composition than class inheritance. As that happens, emphasis shifts away from hard coding a fixed set of behaviors toward defining a smaller set of fundamental behaviors that can be composed into any number of more complex ones. Thus creating objects with particular behaviors requires more than simply instantiating class [4].

In case of web applications, it is sometimes necessary to add additional responsibilities to the web page dynamically and to cater to the changing requirements. Such additions should be transparent without affecting other objects. There can be possibility where in these responsibilities can be withdrawn. In such scenario, Decorator Pattern can be used to add the additional responsibilities to the default web page (whose some of the contents are always constant). For Example if the application has different users who can log into it (such as Admin, Program Manager and Stakeholder).Admin will have different user interface when compared to other users but some of the user interface elements may be similar across all users. In such scenario, we can use Decorator Pattern to decorate the changing user interface elements depending on the user type, keeping the remaining elements constant across all type of users [3].

In such applications, Decorator Pattern objects can be created using Abstract Factory creational pattern. In this work, concrete decorator objects are created with such Abstract Factory pattern without specifying their concrete classes. This encapsulates the object instantiation process from the client thus making the system more flexible to future changes in the requirements. Additional new users can be added to the system without applying changes to the client side code. Client side code is hidden from the type of current user instantiated by the Abstract Factory pattern thus making the system more reusable and easier to maintain. Since the application typically needs only one instance of a concrete factory in abstract factory pattern, we used Singleton design pattern to implement the concrete factory object.

We applied the required changes to the DMS application [3] to achieve the above mentioned objectives. The resulting application showed considerable improvement in performance when multiple users are added. It abstracts the client code from changes in concrete decorator object instantiation.

The rest of the paper is organized as follows. Section 2 mentions about the previous study conducted on this subject. Section 3 explains the motivation of this proposed work Section 4 describes Abstract Factory and Singleton Patterns and their applicability. Section 5 briefs about DMS web based application. Implementation and sample code is listed in section 6. Finally conclusion is presented in section 7.

2. PREVIOUS STUDY

In [1], a set of design patterns commonly used in web based applications has been proposed as a case study. These are the patterns which frequently occur in a web based applications which are discussed in this work.

In [3], Application of Decorator Design Pattern to the web based application has been proposed and implemented. Decorator added additional responsibilities to the default user page thus

making the application more adaptable to the future requirements changes. New user types in the application can be added without changing the client code by using the Decorator Design Pattern. In[6], It is briefly introduced the concept of software design patterns and given a research on some design patterns including Observer Pattern, Decorator Pattern, Factory Method Pattern and Abstract Factory Pattern.

In [7], Research and application of Design Patterns on Shopping Mall component design has been proposed. Strategy, bridge and abstract factory patterns are being used in this work.

3. MOTIVATION

In [4], Client code is exposed to the creation of decorator pattern objects and their composition and representation. Whenever new user is added, client code is changed since it is exposed to the decorator objects creation. Inclusion of Abstract Factory design pattern will ensure the following advantages:

1. Abstraction of Decorator Object creation from Client code thus making the application more adaptable to future changes such as addition of new user type.
2. Easy to understand and maintain the application
3. Design can be reused because of the use of use of additional design patterns.
4. Provides single instance of the abstract factory object by using Singleton creational pattern.

Abstract Factory and Singleton Patterns together incorporate the aforementioned objectives in the proposed implementation.

4. DECORATOR PATTERN STRUCTURE

Decorator pattern adds responsibilities to individual objects dynamically and transparently, that is, without affecting other objects. Such additional responsibilities can be withdrawn. Also when extension by sub classing is impractical.

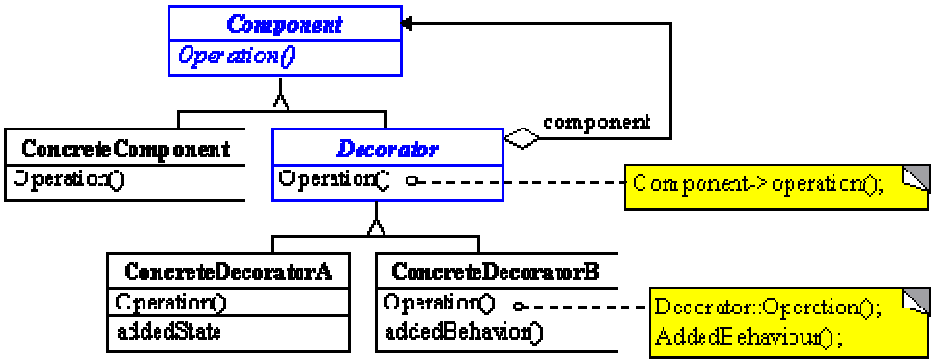

Figure 1. Structure of the Decorator Pattern

Component defines the interface for the objects that can have responsibilities added to them dynamically. Concrete Component defines an object to which additional responsibilities can be attached. Decorator maintains a reference to a component object and defines an interface that conforms to the component interface. Concrete Decorator adds responsibilities to the component.

5. ABSTRACT FACTORY AND SONGLETON PATTERN STRUCTURE

Abstract factory provide an interface for creating families of related or dependent objects without specifying their concrete classes. It helps to control the classes of objects that an application creates. Because a factory encapsulates the responsibility and the process of creating product objects, it isolates clients from implementation classes. Clients manipulate instances though their abstract interfaces. Product class names are isolated in the implementation of the concrete factory; they do not appear in client code [4].

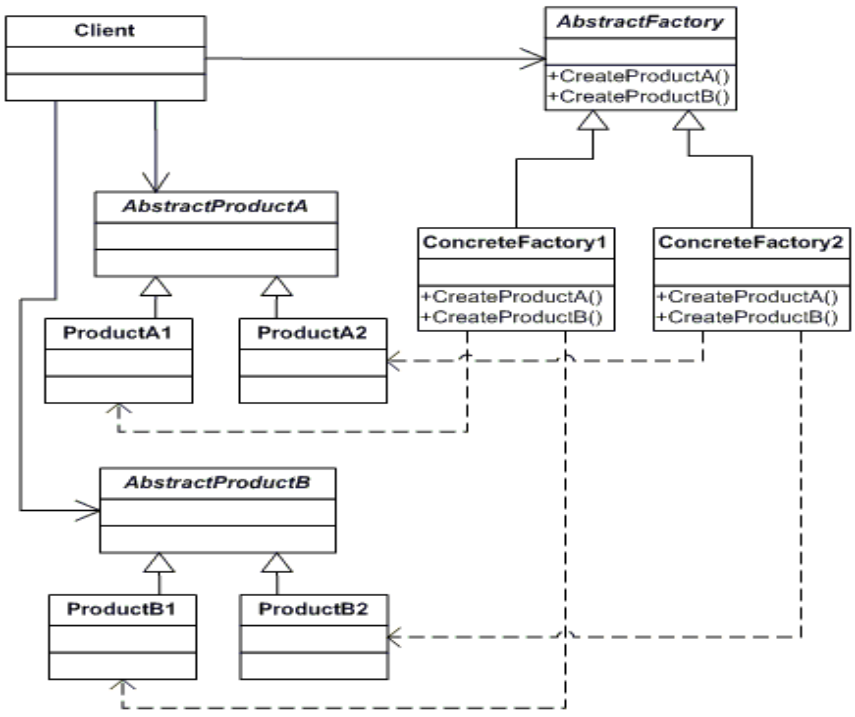

Figure 1. Structure of Abstract Factory Pattern.

Abstract Factory declares an interface for operations that create abstract product objects. Concrete Factory implements the operations to create concrete product objects. Abstract Product declares an interface for a type of product object. Concrete Product defines a product object to be created by the corresponding concrete factory. It implements the Abstract Product interface. Client uses only interfaces declared by Abstract factory and Abstract Product classes.

The factory completely abstracts the creation and initialization of the product from the client. This indirection enables the client to focus on its discrete role in the application without concerning itself with the details of how the product is created. Thus, as the product implementation changes over time, the client remains unchanged. The most important aspect of this pattern is the fact that the client is abstracted from both the type of product and the type of factory used to create the product. Furthermore, the entire factory along with the associated products it creates can be replaced in a wholesale fashion. Modifications can occur without any changes to the client[5].

In our work, Abstract Product is that of Decorator object of the DMS application. Concrete products are different types of users (Admin, Program Manager and Stake Holder). Essentially,

Decorator Pattern objects are made as that of Abstract Product and its Concrete Product objects. Individual Concrete Factory objects are created for each of the user types (Concrete Factory1 for user type Admin, Concrete Factory2 for user type Program Manager and Concrete Factory3 for user type Stake Holder). In this design, Client is unaware of which concrete product and concrete factory classes are created since it access the required objects embedded in abstract class pointer. Hence the client code is detached from underlying concrete object creation and making the system more adaptable to future requirement changes and easier to maintain it.

Singleton Pattern ensures a class has only one instance and provides global point of access to it. Since there is only one instance of the Concrete Factory class, it is implemented using Singleton Pattern so that client can have access to it from well known access point.

6. DMS APPLICATION

Development Management System is a web based application which allows different types of user (Admin, Stake Holder and Program Manager) to login and operate the DMS application. User page has certain functionality similar across all type of users. Changed functionality is implemented using corresponding Decorator objects, using XML input data queried through XPath [3].

With reference to Decorator Pattern, DefaultUserView class is of the type ConcreteComponent which is sub classed from Component base class. Decorator class is sub classed from Component class which is a base class for Concrete Decorators. Concrete decorators can be Admin, Stakeholder or Program Manager decorator classes. Dynamic user page is decorated with any of the concrete decorators depending upon the type of user logged into the system. Also, the contents of the decorator are read from XML using X-path query [3].

In our work, changes are incorporated in Decorator Pattern object instantiation process. It is achieved through Abstract Factory pattern which abstract clients from how the objects are represented, composed and instantiated. Admin, Stake Holder and Program Manager Objects of Decorator design pattern are now become part of the Abstract Factory pattern. Since there can be only one instance of the Concrete Factory object for a particular logged in user, Singleton design pattern is used so that client can access this one and only instance for further usage.

6. IMPLEMENTATION AND SAMPLE CODE

In this work, we applied changes to a web based application called Development Management System. This application allows different types of user (Admin, Stake Holder and Program Manager) to login and operate the DMS application. User page has certain functionality similar across all type of users. Changed functionality is implemented using corresponding Decorator objects, using XML input data queried through XPath [3].

AbstractFactory class(Listing1) is a Singleton Design Pattern class which creates the required Concrete Factory object depending the type of user logged into the system. Instance is a static function which returns the single instance of the concrete factory object. getInstance is a helper function which returns the appropriate concrete factory object depending the user type. User type can be read from either configuration file or registry. Abstract factory also has an important abstract function, CreateUser which will be implemented by the concrete factory objects to create the respective user type object.

```
abstract class AbstractFactory
{
//one and only singleton instance
private static AbstractFactory instance = null;

        //function to access the single instance object
        public static AbstractFactory Instance()
        {
                if(instance == null)
                {
                        instance = getInstance();
                }
                return instance;
        }
        //helper function to create proper factory object
        Private static AbstractFactory getInstance()
        {
                string userType; //get the user type from //.ini file or registry
                //Create factory object depending on user //type
                Switch(userType)
                {
                Case “Admin”:
                        return new ConcreteFactory1();      Case “StakeHolder”:
                        return new ConcreteFactory2();
                Case “ProgramManager”:
                        return new ConcreteFactory3();
                }

        }
        //abstract class & function to create decorator //object
                Public abstract Decorator CreateUser();
}
```

Listing1: Abstract Factory Singleton class

ConcreteFactory1 class (Listing 2) is a sub class of AbstractFactory class which is responsible for creating AdminDecorator object corresponding to Admin user type. AdminDecorator takes DefaultUserView as an input argument to its constructor and AdminDecorator object is returned to the client which has called CreateUser function.

```
class ConcreteFactory1 : AbstractFactory
{
        Public override Decorator CreateUser()
        {
                //Default user view as an input to decorator
                DefaultUserView duv = new DefaultUserView();       //create&return Admin Decorator
        object
                return new AdminDecorator(duv);
        }
}
```

Listing2: ConcreteFactory1 class for Admin Decorator

ConcreteFactory2 class (Listing 3) is responsible for creating stake holder decorator object. ConcreteFactory is a subclass of AbstractFactory class and implements CreateUser function

declared in AbstractFactory base class. Again the StakeHolderDecorator object is passed to the client as Decorator object pointer, without revealing the underlying object in it.

```
class ConcreteFactory2 : AbstractFactory
{
        Public override Decorator CreateUser()
        {
                DefaultUserView duv = new DefaultUserView();            //create&return Stake
Holder Decorator object
                return new StakeHolderDecorator(duv);
        }
}
```

Listing3: ConcreteFactory2 class for Stake Holder Decorator

ConcreteFactory3 class (Listing 4) is responsible for creating ProgramManager Decorator object. ConcreteFactory is a subclass of AbstractFactory class and implements CreateUser function declared in AbstractFactory base class. Again the ProgramManagerDecorator object is passed to the client as Decorator object pointer, without revealing the underlying object in it.

```
class ConcreteFactory3 : AbstractFactory
{
        Public override Decorator CreateUser()
        {
                DefaultUserView duv = new DefaultUserView();            //create&return Program
Mgr Decorator object
                return new ProgramManagerDecorator(duv);
        }
}
```

Listing4: ConcreteFactory3 class for Program Manager Decorator

AdminDecorator is a ConcreteDecorator (Listing 5) which adds the responsibility of rendering the Admin related XHTML to the web browser. AdminDecorator will copy the visual component object to its base class visual component member variable. It implements the Render function which in turn reads the XML file to read its required contents and develops the XHTML as per the XML data [3].

```
public class AdminDecorator : Decorator
{
        //constructor for Admin Decorator
        Public AdminDecorator(VisualComponent vc)
        {
                base.vc = vc;
        }
        …
}
```

Listing5: Admin Decorator class derived from Decorator

```
AbstractFactory  af = AbstractFactory.Instance();
Decorator dc = af.CreateUser();

//Client unaware of the type of user
string totalXHTML = dc.Render();
//write the XHTML as string to the browser
Response.Write(totalXHTML);
```

Listing 6: Client code to render the XHTML

Client Code (Listing 6) shows how the final XHTML contents are retrieved from Decorator object. Client code has been simplified and is now unaware of the user type logged into the system. It gets single instance of concrete AbstractFactory object and in turn created the proper Decorator Object which will render the XHTML code to the browser.

Above code snippet reveals us that the client code is hidden from the instantiation process of concrete decorator objects. AbstractFactory object creation is implemented with the Singleton design pattern to ensure that only one instance of the concrete factory object is created in the application to provide the global point of access. Hence the above method of implementation makes the maintenance of software easier and any changes in the user requirements can be handled without changing the client part of the code. It also improved the performance of the application when more number of users is added to the system.

7. CONCLUSION

Software Design Patterns are proven design methods which when used carefully in the implementation of software development will lead to several advantages. Reusability, Encapsulation and Ease of system maintenance are some of the benefits one can incorporate into software systems if these proven design patterns are used in the system. Creational patterns belong to that category of design pattern which will help in abstraction of object instantiation process. They help make a system independent of how objects are created, composed, and represented. Abstract Factory and Singleton patterns belong to creational pattern category, which are used in this work to encapsulate the Decorator Pattern object instantiation process. Concrete Decorator objects are created by respective Concrete Factory objects depending on the type of user logged into the system. This resulted in improved application performance when additional user types are added to the system and easier to maintain the application code for future requirement changes.

REFERENCES

[1] Phek Lan Thung, Chu Jian Ng, Swee Jing Thung, Shahida Sulaiman, “Improving a Web Application Using Design Patterns: A Case Study”

[2] V.Pawan, “Web Application Design Patterns”, CA Morgan Kauffman, 2009.

[3] Vijay K Kerji, “Decorator Pattern with XML in web based application”, ICNCS 2011 Kanyakumari, India.

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns Elements of Reusable Object Oriented Software”,Addison Wessley

[5] Danijel Matic, Dino Butorac, Hrvoje Kegalj, “Data Access Architecture in object oriented application using Design Patterns”

[6] Mu Huaxin, Jiang Shuai, "Design Patterns in Software Development", IEEE 2011

[7] Jing Gang Chu, Jia Chen, "Research and Application of Design Patterns on Shopping Mall Design Component"

Authors

Mr. Vijay K Kerji completed his B.E in 1993 in E&CE and M.Tech in CSE in 2007. He worked for W.S Telesystems Ltd for two years, CMC Limited Hyderabad for four years, and Siemens Corporate Research New Jersey for five years and Intel India for two years as software engineer/senior software engineer. He has three international publications including IEEE and Springer in the area of software design patterns and computer networks. Currently he is serving educational institutions as a mentor in the area of c omputer science engineering and electronics and communication engineering. His area of interest include software design patterns, object oriented analysis and design, web development, operations research, computer networks and electronic circuits.