SURVEY: DESIGN PATTERNS IN WEB BASED APPLICATIONS

Taranpreet Singh Saini[1], Priyanka Lokhande[2], Dipali Baviskar[3]
[1,2,3]Computer Engineering
Email: saini_13sf@yahoo.co.in[1], priyankalokhande05@gmail.com[2], dipali.shimpi@mitpune.edu.in[3]

Abstract
Design pattern is a general explanation to commonly occurring problems in software development. Any design issue can be solved by the developers by applying design pattern. Design patterns help designers learn a new design paradigm, communicate architectural knowledge, help people and help new developers avoid pitfalls that have been learned only by costly experiences. Pattern-based web applications have become popular since they promote reusability and consistency. Web application designers need to address many challenges during development in order to satisfy the quality of service requirements like speed, security and scalability. In this paper we will see how design patterns are connected with human computer interface (HCI) with respect to usability with the help of facebook case study.
Index Terms: Design pattern, web based applications, case study, MVC.

I. INTRODUCTION

In a field of software engineering, a design pattern is a repeatable answer to a usually occurring problem in software design. A design pattern isn't a complete design that can be changed directly into code. It is a description or pattern for how to answer a problem that can be used in several different conditions.

Uses of Design Patterns

Design patterns can enhance the development process by providing tested, verified development models. Effective software design need to consider matters that may be hidden until later in the implementation part. Recycling design patterns helps to avoid delicate issues that can cause major difficulties and advances code readability for coders and architects used to with the patterns.

Every so often, users only understand how to apply definite software design techniques to certain problems. These techniques are hard to apply to a wider range of difficulties. Design patterns offer general answers, documented in a format that does not need specifics tied to a specific problem.

Additionally, designs permit designers to communicate using familiar, well understood names for software interactions. Usual design patterns can be enhanced over-time, making them tougher than ad-hoc designs.

Creational Design Patterns

These design patterns are all about class actualization. This pattern can be further classified into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance efficiently in the actualization process, object-creation patterns use allocation efficiently to get the work done.

- Abstract Factory - Implements an instance of several families of classes.
- Builder - Splits object construction from its representation.
- Factory Method - Generates an instance of numerous derived classes.
- Object Pool - Avoid expensive gaining and release of resources by recycling objects that are no longer in use.
- Prototype - A fully initialized instance to be cloned.

- Singleton - A single instance of a class.

Structural design patterns

These design patterns defines Class and Object composition. Structural class-creation patterns use inheritance to constitute interfaces. Structural object-patterns describe ways to constitute objects to get new functionality.

- Adapter – Combine interfaces of different classes.
- Bridge - Splits an object's interface from its implementation.
- Composite - A tree structure of simple and composite objects.
- Decorator - Add duties to objects dynamically.
- Façade - A single class that signifies an entire subsystem.
- Flyweight - A fine-grained instance used for effective sharing.
- Private Class Data - Limits accessor/mutator access.
- Proxy - An object signifies another object.

Behavioral Design Patterns

Behavioral design patterns describe Class's objects communication. Behavioral patterns are those patterns that are most explicitly concerned with communication between objects.

- Chain of responsibility - A way of sending a request between a chains of objects.
- Command - Summarize a command request as an object.
- Interpreter - A way to include language essentials in a program.
- Iterator - Successively access the elements of a collection.
- Mediator - Describes simplified communication between classes.
- Memento - Capture and reinstate an object's internal state.
- Null Object - Intended to act as a default value of an object.
- Observer - A method of informing change to a number of classes.
- State - Modify an object's behavior when its state changes.
- Strategy - Summarizes an algorithm inside a class.
- Template method - Concede the exact steps of an algorithm to a subclass.
- Visitor - Describes a new operation to a class without change.

II. WEB BASED APPLICATIONS

A Web application is an application that can be retrieved by the users through a Web browser or a particular user agent. The browser generates HTTP requests for specific URLs that map to resources on a Web server. The server reduces and returns HTML pages to the client, which the browser can show. The basic of a Web application is its server-side reason. The application can comprise numerous distinct layers. The classic example is a three-layered architecture contained of presentation, business, and data layers. Figure 1 demonstrates a typical Web application architecture with mutual components grouped by different areas of concern.

Web based applications are the final way to take benefit of today's technology to improve your organizations output & efficiency. Web based application gives you chance to access your business data from anywhere in the world at anytime. It also eases you to save time & money and increase the interactivity with your customers and partners.

III. MODEL-VIEW-CONTROLLER (MVC)

For periods people have try to find ways to separate style from matter, good manners from good science, and user interface from application reason. The Model View Controller (MVC) design pattern is a way of unraveling the user-interface from the substance of the application.

In modern years, MVC has become a widespread plan for developing websites. There are nowadays web-MVC frameworks accessible for many programming languages, for instance Struts for Java, Maypole for Perl and Rails for Ruby.

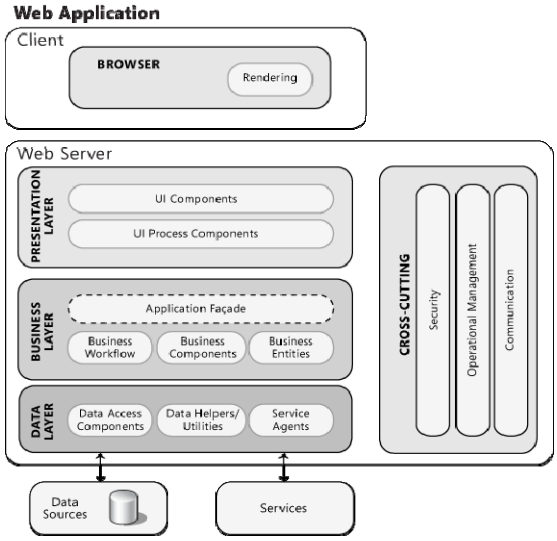

Figure 1

Since MVC was initially designed for traditional GUI applications, certain facts in original MVC pattern don't map well to web applications.

Model

The Model manages the state of the application. The state is nothing but what your application is *about*. If your application is a forum, your Model might comprise Class: DBI objects expressing threads, users and postings. The Model knows nothing about HTML, or web servers or anything like that. It just supplies methods to query the state, and ways to change that state.

View

The View is the illustration of the user interface. Generally there are many (possibly nested) Views in one application. A view can query the model, but it is not invented to alter the state. In web based MVC systems, a view can be applied using a template that reduces an HTML page. In our theoretical forum application, the Views would be the templates for rendering a full thread, the posting page, the login page etc.

Controller

User actions on the View are passing to Controller. In a web environment, this is generally done by having the Controller manage the incoming HTTP requests.

The Controller receives user requests, and converts them into activities that the Model should take. Then it selects the suitable View to manage the response.

It is allowed to have many Controllers, but most web application frameworks have used assume to have only one.

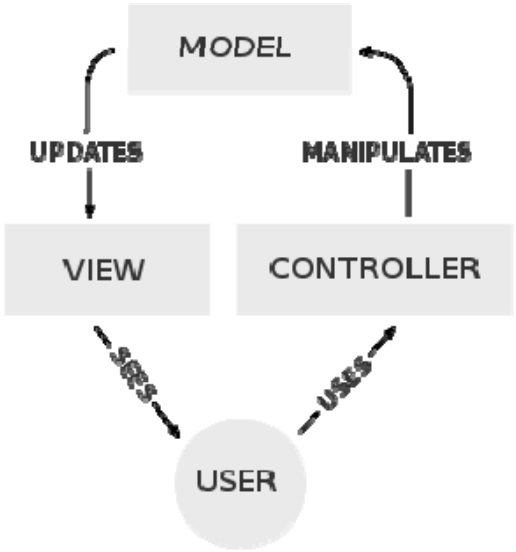

Figure 2 Model View Controller

IV. GENERAL DESIGN CONSIDERATIONS

When designing a Web application, the aim of the software architect is to reduce the complexity by separating tasks into different areas of concern while manipulating a secure, high performance application. Monitor these guidelines to confirm that your application meets your requirements, and performs effectively in situations common to Web applications:

- Partition your application logically. Use layering to logically partition your application into presentation, business, and data access layers. This supports you to generate sustainable code and allows you to display and enhance the performance of each layer individually. A clear logical separation also offers more options for scaling your application.

- Use abstraction to implement loose coupling between layers. This can be accomplished by describing interface components, such as a façade with known inputs and outputs that interprets requests into a format understood by components within the layer. Additionally, you can also use Interface types or abstract base classes to describe a shared abstraction that interface mechanisms must implement.

- Understand how components will communicate with each other. This needs an understanding of the deployment situations your application must support. You must define if communication across process boundaries or physical boundaries should be supported, or if all components will run within the same process.

- Consider caching to minimize server round trips. When designing a Web application, consider using techniques such as caching and output buffering to diminish round trips between the browser and the Web server, and among the Web server and downstream servers. A well designed caching strategy is possibly the single important performance associated design consideration. ASP.NET caching features contain output caching, partial page caching, and the Cache API. Design your application to take benefit of these features.

- Consider logging and instrumentation. You should check and log activities through the layers and tiers of your application. These logs can be used to detect doubtful activity, which often provides primary indications of an attack on the system. Keep in mind that it can be hard to log problems that occur with script code executing in the browser.

- Consider authenticating users across trust boundaries. You should design your application to validate users whenever they cross a trust boundary; for example, when accessing a remote business layer from the presentation layer.

- Do not pass sensitive data in plaintext across the network. Whenever you pass sensitive data such as a password or verification cookie across the network, consider signing and encoding the data or with the use of Secure Sockets Layer (SSL) encryption.

- Design your Web application to run using a least-privileged account. If an attacker handles to take control of a process, the process identity should have controlled access to the file system and other system resources in order to bind the possible damage.

V. USEFUL ABOUT MVC

Separation of requests and pages

Since the Controller is in control of handling the requirements and choosing a suitable page (View), there is no instant coupling between the demand made by the user and the resultant page.

This turns out to be very valuable if the page-flow in the application is multifaceted, but even for simple applications the Controller is a good place to manage common actions - verification and session management can be managed in the Controller, for instance.

Views are dumb

Since all code that does everything aside from building a nice page for the user is out of the View objects, altering the design does not include touching the logic of the application. Since the part of the application that varies the most during and later development is the layout, this means less chance of adding bugs.

Shielding of the Model implementation

Since all activities on the application state are touched by the Model, it is possible to alter the Model's execution without moving the user interface, as long as the Model's public API doesn't modify.

5.1 Problems and limitations

What goes where

Occasionally is just difficult to figure out where an exact piece of the application is supposed to go. Particularly separating the Model from the Controller can be difficult. As a rule of thumb, the Controller should be as nominal as possible - it is only responsible for decoding HTTP requests into Model actions and opt for the right View - the Model should offer all the behavior it can without managing the HTTP requests or output arranging details.

Coupling between View and Model

Difficulty with having the View and Controller querying the Model is that altering the Model's public API means you also have to adapt the Controller and Views that act on it. Adapting the Controller is typically not too much work, but changing a big number of Views will be annoying.

The Model View Controller pattern tries to decrease the impact of these alterations by using two Models: a Domain Model and an Application Model - the View only queries the Application Model, and the Application model

can query the Domain Model. The Application Model typically partly generated by the GUI design tools. I haven't used this pattern at all, so I don't know how useful it is for web applications.

Lots of objects

Creating an MVC application can result in additional classes and objects than a "page-based" system. That means more design up front. On the other hand, a well-designed MVC structure will be easier to adapt and enlarge, because the code will be parted well.

Should I use it

Perhaps, I think that any web-based application that uses more than a handful of patterns, or has a compound communication between pages would be a good candidate for the MVC pattern. There are replacements, of course.

VI. ROLE OF MVC IN WEB APPLICATIONS

Although initially advanced for desktop computing, model–view–controller has been extensively adopted as architecture for World Wide Web applications in main programming languages. Several marketable and noncommercial web frameworks have been shaped that enforce the pattern. These software frameworks differ in their clarifications, mainly in the way that the MVC tasks are separated between the client and server.

Timely web MVC frameworks took a thin client method that placed almost the entire model, view and controller logic on the server. This is still imitated in widespread frameworks such as Ruby on Rails, Django, ASP.NET MVC and Express. In this method, the client sends either hyperlink requests or form input to the controller and then obtains a complete and updated web page (or other document) from the view; the model exists completely on the server. As client technologies have developed, frameworks such as AngularJS, EmberJS, JavaScriptMVC and Backbone have been formed that allow the MVC components to implement partly on the client.

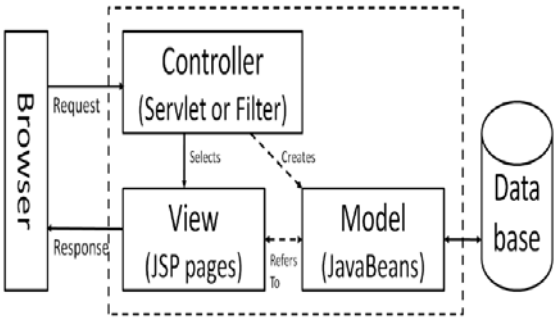

Figure 4 MVC with Java

VII. CASE STUDY – FACEBOOK

The figure 5 shows the facebook user data, which is having attributes for every major aspect. There are four main classes user, page, connect and list. UserClass contains different attributes as UserID, FirstName, Email, EmailVerified, Password, Sex, Birthdate, Hometown, Relationship, Political, Religious and all these values are the user personnel information for creating the facebook id. If user administers, likes or own a page then the class is PageClass and its attributes are different as PageID, Name, Description, Founded, Address, Mission, Published, PublishDate. The third class is about connectivity which is ConnectClass, which shows user's friend list and also gives the add friend option. In ListClass, list is created which are having attributes as newList, addFriends, noOfFriends and showFriends.

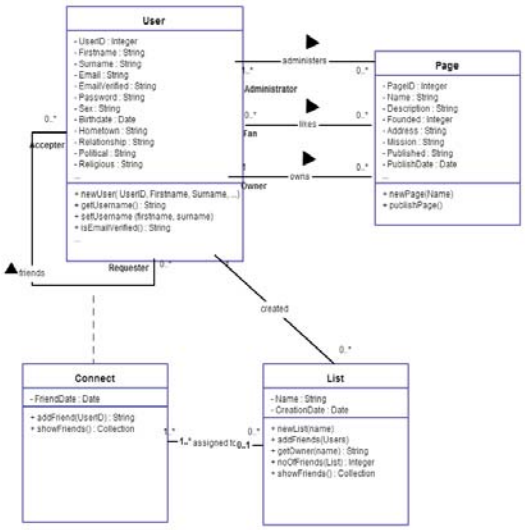

Figure 5 Class Diagram for Facebook User Data

A data flow diagram (DFD) is a graphical demonstration of the data "flow" through a data system, modelling its process aspects. A DFD is often used as an initial step to create an indication

of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the information will come from and go to, and where the information will be stored. It does not show information about the timing of process or information about whether processes will operate in order or in similar. In figure 6 the data flow diagram of facebook is shown with respect to user. Data flow diagram shows the process for new user and registered user.

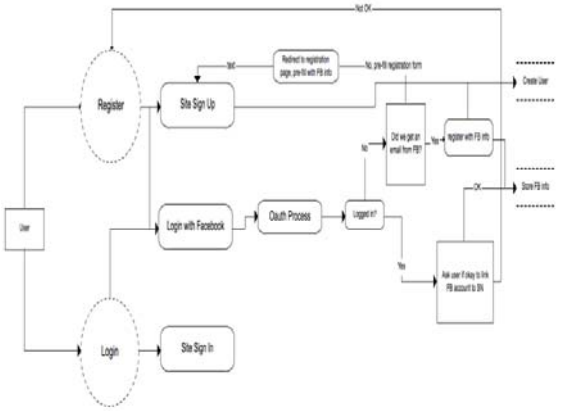

Figure 6 Facebook Data Flow Diagram

Facebook uses OAuth 2.0 protocol framework in figure 7, which allows web application (called "client"), which is typically not the FB resource owner but is acting on the FB user's behalf, to request access to assets controlled by the FB user and hosted by the FB server. In its place of using the FB user credentials to access protected resources, the web application obtains an access token.

Web application should be recorded by Facebook to have an application ID (client_id) and secret (client_secret). When request to some threatened Facebook incomes is received, web browser ("user agent") is transmitted to Facebook's support server page with application ID and the URL the user should be communicated back to after the approval process.

User obtains back Request for Authorization form. If the user agrees the application to get his/her data, Facebook approval server redirects back to the URI that was stated before together with permission code ("verification string"). The approval code can be swapped by web application for an OAuth admission token.

If web application gets the admittance token for a FB user, it can attain official requests on behalf of that FB user by counting the right of entry token in the Facebook Graph API requests. If the user did not approve web application, Facebook issues redirect request to the URI specified earlier, and adds the error_reason parameter to notify the web application that approval request was denied.

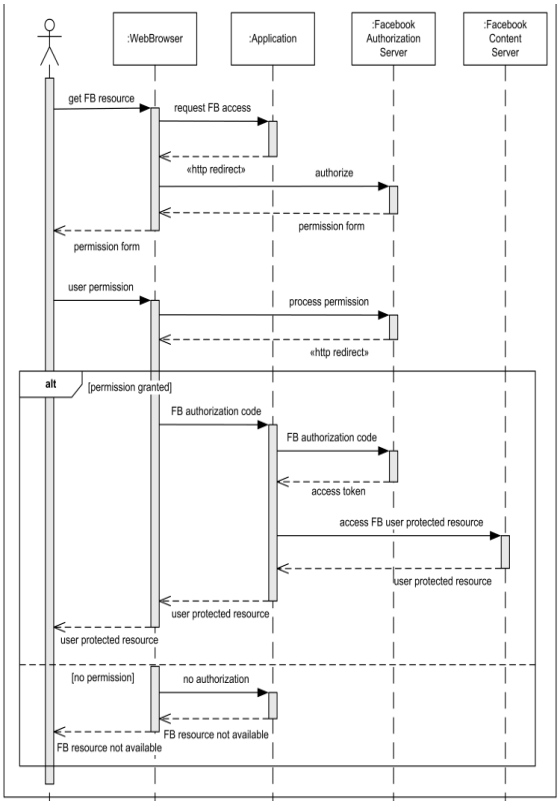

Figure 7 UML Sequence Diagram for Facebook

VIII. CONCLUSION

In this paper, it is concluded that MVC decouples views and models by establishing a subscribe/notify protocol between them. Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself. This approach lets you attach multiple views to a model to provide different presentations. You can also create new views for a model without

rewriting it. This is how design patterns are connected with human computer interface (HCI) with respect to usability. Model view controller (MVC) design pattern is helpful in web based applications and have a great future scope.

IX. ACKNOWLEDGEMENT

We would like to thank our Ass. Prof. Dipali Baviskar (Software Design Architecture) for excellent supervision and good advices.

REFERENCES

[1]"facebook-authentication-uml-sequence-diagram-example@ www.uml-diagrams.org." .
[2] "Facebook Login Activity Diagram @ creately.com." .
[3] "Facebook @ creately.com." .
[4] "Data_flow_diagram @ en.wikipedia.org." .
[5]"b4b12dcdff87e3ae83d0042427ecada0bcb57a4b@ blog.iandavis.com." .
[6] D. Bindal, "International Journal of Advanced Research in Computer Science and Software Engineering," vol. 3, no. 6, pp. 426–433, 2013.
[7] J. Wang, Y. T. Song, and L. Chung, "Analysis of secure design patterns: A case study in E-commerce system," Proc. - Third ACIS Int. Conf. Softw. Eng. Res. Manag. Appl. SERA 2005, vol. 2005, pp. 174–181, 2005.
[8] J. Wang, Y.-T. Song, and L. Chung, "From software architecture to design patterns: a case study of an NFR approach," Softw. Eng. Artif. Intell. Netw. Parallel/Distributed Comput. 2005 First ACIS Int. Work. Self-Assembling Wirel. Networks. SNPD/SAWN 2005. Sixth Int. Conf., pp. 170–177, 2005.
[9] F. A. Vinisha and R. Selvarani, "Study of Architectural Design Patterns in Concurrence," pp. 393–402.
[10]J. Shuai and M. Huaxin, "Design Patterns in Object Oriented Analysis and Design," IEEE xplore, pp. 326–329, 2011.
[11]E. M. Sahly and O. M. Sallabi, "Design pattern selection: A solution strategy method," 2012 Int. Conf. Comput. Syst. Ind. Informatics, ICCSII 2012, 2012.
[12]J. S. Fant, H. Gomaa, and R. G. Pettit, "Architectural design patterns for flight software," Proc. - 2011 14th IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput. Work. ISORCW 2011, pp. 97–101, 2011.
[13]D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," ACM SIGSOFT Softw. Eng. Notes, vol. 17, no. 4, pp. 40–52, 1992.
[14]H. Mu and S. Jiang, "Design patterns in software development," 2011 IEEE 2nd Int. Conf. Softw. Eng. Serv. Sci., pp. 322–325, 2011.
[15]N. Heidke, J. Morrison, M. Morrison, and E. Claire, "Assessing the Effectiveness of the Model View Controller Architecture for Creating Web Applications," Science (80).
[16]"IJCSE10-02-04-48.pdf." .
[17]A. Paikens and G. Arnicans, "Use of Design Patterns in PHP-Based Web Application Frameworks," Sci. Pap. Univ. Latv. Comput. Sci. Inf. Technol., vol. 733, pp. 53–71, 2008.
[18]F. Montero and M. Lozano, "A first approach to design web sites by using patterns," First Nord. Conf. …, 2002.
[19]I. Wentzlaff and M. Specker, "Pattern-based development of user-friendly web applications," Work. Proc. sixth Int. Conf. Web Eng. - ICWE '06, p. 2, 2006.
[20]D. Bonura, R. Culmone, and E. Merelli, "Patterns for web applications," Proc. 14th Int. Conf. Softw. Eng. Knowl. Eng. - SEKE '02, p. 739, 2002.
[21]V. K. Kerji, "Creational patterns to create decorator pattern objects in web application," Commun. Comput. Inf. Sci., vol. 204 CCIS, no. 5, pp. 225–232, 2011.
[22]C. Richardson, "A Pattern Language for J2EE Web Component Development 1," Development, pp. 1–34, 2001.
[23]S. S. Thabasum and U. T. M. Sundar, "A Survey on Software Design Pattern Tools for Pattern Selection and Implementation," vol. 2, no. 4, pp. 496–500.