Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 5, May 2016, pg.849 – 855

Model View Design Patterns with IOS

Gautham Krishna Ballal[1]
K.R. Udaya Kumar Reddy[2]
[1]Computer Science and Engineering, NMAMIT, Nitte, VTU, India
[2]Professor, Computer Science and Engineering, NMAMIT, Nitte, VTU, India
[1] gauthamballal@gmail.com, [2] krudaykumar@nitte.edu.in

Abstract— A Mobile Application which interacts with the user will need a user interface. Model - View - Controller (MVC) is the most popular design pattern to interact with the user. MVC separates the data (Model) which needs to be displayed to the user (View) and user interaction like tap of a button (Controller). But upon study of this design pattern we find out that there are two more design pattern which can be studies and of importance. Those are the Model - View - ViewModel (MVVM) and Model View Presenter (MVP). We implement these design patters on two languages and find out the performance variations and difference between them. The study shows that the selection of patterns should take into account the use cases and quality requirements at hand, and chosen technology. We illustrate the selection of a pattern with an example of the quiz application. The Study results in the performance analysis of each design pattern differentiated between the two languages under study. The Two languages are Objective - C and Swift.

Keywords— Model, View, Controller, ViewModel, Presenter

I. INTRODUCTION

A design pattern is a description of class and objects that communicate with each other and are customised to solve a general design problem. A design pattern is a customised answer to a problem in a context. Programs that use design patterns becomes more extensible, easier to change in the future and reusable. Also programs that follow a design pattern tend to be more efficient and elegant than the rest of the programs, as number of lines of code which needs to be written to accomplish the same task gets reduced.

The Design Patterns that are under consideration can be mainly divided into two types. One with separation of components and other without separation of components[1].

Widget-based User Interface: Widget-based User Interfaces is a naive way of creating an application with graphical components as provided by any development environments. Here the developer uses the user interface widgets which are already defined to display contents on the window, and writes code to handle these graphics user interface elements which is necessary for the working of the application. Model-View-Controller (MVC), MVC separates the user interface layer from application logic [2]. In MVC the controller plays a major role in receiving requests and working with model to display appropriate data on the view. Model-View-Presenter (MVP), MVP [3] is a UI based architectural design pattern created to improve the separation of concerns in presentation logic and enhance automated unit testing. Model-View-ViewModel (MVVM), In MVVM[4], The Viewmodel has access to the model, and the view model has logic which ends to be applied to the model so that

the request made from the view can be satisfied. MVVM is considered as an alteration to a general PM design pattern which was designed specifically for Microsoft's Silverlight platform.

We have picked Swift and Objective C because these are the only two languages on which you can develop a iOS application. and we can check for the performance difference with the same benchmark device.

Swift Vs Objective C [5][6], they can be differentiated on the basis of Readability, Maintainability, Safety, Memory Management, Code Quantity, Processing Power, Name collisions with open source projects, Supports for dynamic libraries are to name the few.

II. RELATED WORK

Smalltalk-80 System : Smalltalk-80 System uses Model-View-Controller (MVC) programming design pattern [7], a three-way factoring where the application's state display, user interaction and operations are taken over by the objects of different classes. Even though concepts dominating Smalltalk [8] is quite less, some unusual terminologies define it. Object-message orientation is incorporated in the system so the programming concepts that are required to grasp smalltalk is very less. Five keywords define the language smalltalk, they are object, instance, class, method and message. Smalltalk utilises hardware such as display screens supporting high-resolution and graphics pen or a mouse. These devices lets user to interact with the screen and view information on the screen. Every pattern in Smalltalk would be a blueprint of its dynamic system. A distinguishing amount of instructions can be achieved by dynamics of multiple access in case of static linear processes.

Web MVC Architecture : Browsers these days are powerful and incorporates excellent features, these are developed by using JavaScript [9]. Within a year the code size has increased by 45 % in a year when you look at the HTTP Archive. The client application is becoming more complex as the day progresses due to innovation in technologies and user expectation [10]. Development of Application requires synchronisation and collaboration with many developers. Writing reusable and maintainable code is crucial when it comes to developing any application. Even Chrome Application which has rich client-side features, is no exception. MVC helps in writing better organised and maintainable code when compared to standard JavaScript. This pattern has been used extensively and tested over multiple languages by generations of developers[11].

MVVM Architecture : MVVM is one of the popular design pattern for creating design workflow. Many design patterns were introduced to create software user interfaces, like Model-View-Presenter (MVP) which is a variation of Model-View-Controller pattern (MVC). In Model-View-Presenter design pattern, view is used to display data on the user interface, models are used to wrap the data to be displayed, while the presenters acts as communicator between the view and the model. The Presenter carries the task of populating and validating model data. For more information about the Model View Presenter, [12] column in August 2006 Design Patterns written by Jean-Paul Boodhoo's is a good read[13]. in 2005, Model-View-ViewModel pattern was published by John Gossman. MVVM also features an abstraction of a view similar to Presentation Model (PM). MVVM was introduced to simplify basic features of WTF in creating user interfaces. The ViewModel in MVVM and Presentation Model in PM perform the same task of creating an abstraction of a view. In MVVM, a ModelView does not hold any connection or reference to the view [14]. The ModelView does not take part by holding a connection. The views binds itself to the ModelView and get access to the model data. The data binding facilitates in keeping the view up to date, by populating the latest updated data in view in case if the data gets restored.

MVP Architecture : MVP is a architectural design pattern which helps in separation of concerns in presentation logic. Model-view-presenter (MVP) is similar to MVC design pattern focusing mainly on presentation logic improvement. MVP was developed in early 1990s by a company named Taligent when their team was working for a C++ model environment. Both MVP and MVC concentrate on separation of concerns in circumstances having more than one component. In MVP design pattern, the M stands for the term Model, V stands for View and P stands for the Presenter. The data to be displayed upon user interface is encapsulated in a model. The logic for presenting in a view is contained in the presenter. The interactions in view are passed on to the presenter as the view interacts through an interface. The decoupling of views from interactions allows developer/designer to mock views at any point of time. In an ideal case, the view does not contain any business logic, hence acting as a passive view. The model is similar to MVC model, encapsulating data to be displayed. The presenter performs the role of moderator between model and view which are isolated. Hence it is the presenter which binds the components together and act as the heart of MVP design pattern.

III. DESIGN PATTERNS

Lets consider our problem statement, The user once opens up a design pattern to start his his, has options of tests the application offers, once a test is selected. Questions pertaining to the test is displayed for the user. Also a timer starts and once the timer starts it runs for a predetermined amount of time. If timer expires the user is moved to results page where results are displayed to him. In the questions view, user can answer all questions related to the test he had selected previously, user can also check the hint which helps in take better decisions while answering the question. Once the test is completed the user moves to results page where his scores are displayed to him. For this page he can check all the answers for the questions he had answered. This gives him the knowledge about the test he wants to take next time. From this page, user can go back to the page where different design patterns are displayed. The Same problem statement is designed in three different ways, MVC, MVVM and MVP. All three design patterns use the same common datasource which is provided by the admin in the form of a .csv.

A data flow diagram (DFD) [15] illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

Data flow diagram for our problem statement can be broken down into two levels, level 0 and level 1. Here the admin takes the responsibility of proving the application with the list of all questions which needs to be displayed to the user in the game. Level 0 : There are two external entities, The User and the Admin. The User provides his input to the application by answering the questions and tapping on different UI elements. He is the one using this application. Another external entity, Admin provides the app with the question file, which contains all the questions in the csv format. The process, which is the quiz application, takes the input from the user and the csv file from the admin to run the game and successfully execute different operation. Level 1: (Fig 1)Level 1 breaks down the single process from Level 0 into subprocesses, that is the Home, Test, Game, Result and All Answer. each of the process interacts with the datasource and user to proceed from one process to another. The datasource provides all tests and also save users answers in the datasource. The data source provides all test information, all answers related to a specific test, saves user answer related to a specific question, get all the questions answered by the user. After user has completed the test all answers are removed from the questions answered by the user and he can continue to take test again.

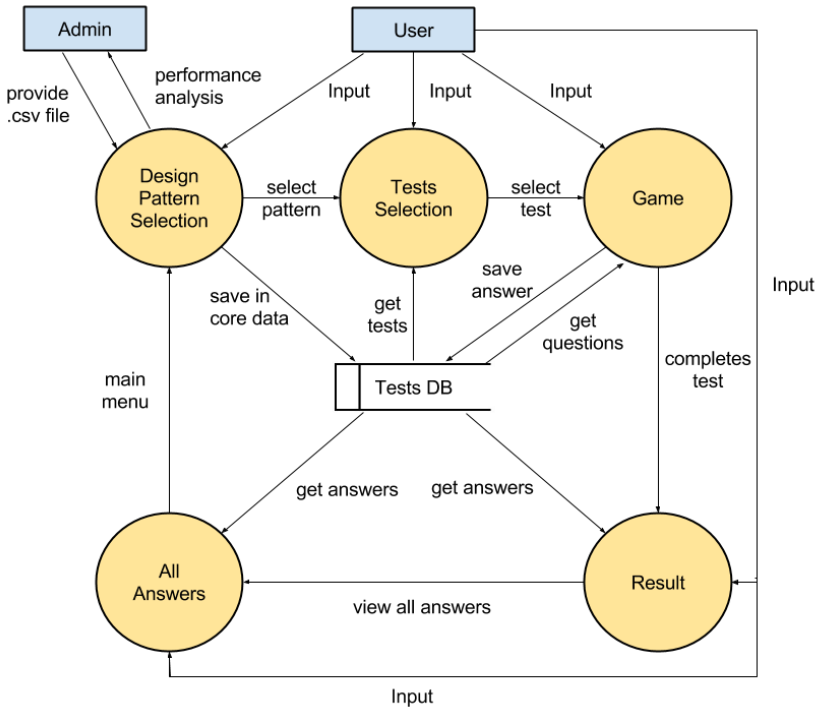

Fig. 1 Data Flow Diagram Level 1

MVC : When implementing problem statement with MVC, all the calculation takes place at the controller. Controller is responsible for fetching details about tests and question, also displaying these details are taken care via the View with the help of the controller. In our example PackageListingViewController displays the tests offered by the application, when user taps on any of the test user is navigated to GamePlayViewController, where all the game logic is implemented. After the test completes user is displayed the results in ResultViewController, And if user wants to check correct answers he will be navigated to AllAnswersViewController. All the View controller has direct access to the data or Model present in the application. Due to which it is hard bounded to the data and view. View controller has access to the components of the UIKit and the Model, so it fetches the data from the database and assigns it to the UI Components. Whenever there needs to be synchronisation it is done by flow synchronisation. Any Event from the user is handled by the view controller, when a button is tapped the button sends the control to the view controller and the view controllers takes action as it sees fit. Also any data which needs to be updated is handled by the controller. The Model just has the basic data and does not perform any calculations. All calculations and validation are done in the view controller.

MVVM : When implementing problem statement with MVVM, there are few design decisions which needs to be taken care. All the details needed by the view needs to be made available to it by the view model, the view does not have direct access to the model. In our example the PackageListingView uses PackageListingViewModel for all the details needed, and PackageListingViewModel in turn uses Test Object to fetch information stored in the model. After user selects a test, they are taken to GamePlayView which uses GamePlayViewModel for all the details needed, and GamePlayViewModel in turn uses Test Object to fetch information stored in the model. MVVM follows strict rule that the View does not have any knowledge about the model. it need to query ViewModel for any details necessary to display data. This helps in reusing the View in multiple places. The View defines the interface which the the View Model needs to satisfy. If the View is used by another ViewModel all it needs to do is implement these methods provided in the interface and the app will work seamlessly.

MVP : When implementing problem statement with MVVM, there are a lot of things which needs to be considered, for example every View needs to have a interface which will help presenter to interact with the view. Also the Presenter will have a interactor which will help in interactor with the model and incase there are details witch needs to be downloaded from server, this is the place to do it. In our example the PackageListingPresenter has PackageListingView and PackageListingInteractor, to fetch and display details of all the Tests offered by the application, The interactor in turn uses Test Model object. GamePlayPresenter has access to GamePlayView and GamePlayInteractor, with the help of these the presenter executes game logic. After the game ends we navigate to results page, GamePlayPresenter has access to GamePlayView and GamePlayInteractor, with the help of these the presenter executes game logic. After the game ends we navigate to results page, ResultsPresenter has access to ResultView and ResultsInteractor, Interactor provides set all all questions answered by the user and it calculates the correct answers and displays it to the user with the help of ResultsView. If users wants to check all his answers he will be navigated to AllAnswersPresenter, AllAnswersPresenter has access to AnswerInteractor and AnswersView, with these two correct answers to all the questions in that test is displayed to the user.

IV. IMPLEMENTATION

The application is implemented using xCode on a Mac OS. And the performance analysis is done with the help of iPhone 5c. User interaction is automated with the help of automation tools provided by apple.

Hardware Requirements

- MacBook Pro / iMac / Mac Mini
- Processor: 2.2 GHz Intel Core i7
- HardDisk Space: 500 GB
- RAM Memory: 8 GB
- Apple iPad / iPhone
- Processor : Dual-core 1.0 GHz Cortex-A9 or higher
- Hard Disk Space : 16/32/64 GB
- RAM Memory : 512 MB RAM or Higher

Software Requirements

- Operating System: MAC
- SDK: 10.10.5 (OS X Yosimite)
- Language Used: Objective - C and Swift

V. RESULTS

MV* design patterns provide reusable solutions to the recurring problem of synchronizing user interfaces with domain data. we have provided a overview of the major MV* pattern families and discussed concrete patterns of each family. We have Implemented our problem in Objective - C and Swift, with Model - View - Controller, Model - View - ViewModel and Model - View - Presenter. UI Automation is performed on the application. The memory and CPU is profiled with the help of allocation and leaks instruments.

MVC : CPU Usage : Fig 2 has the details of the cpu usage testing performed on the problem implemented on Objective C and Swift. Objective C has 3 % higher usage of CPU when compared to Swift. So Swift performs better when implemented in MVC. This is mainly due to the lack of complexity in MVC when compared to other design patterns

Memory Usage : Fig 3 displays details of memory usage of the application in both Objective C and Swift. On An average the swift programming language uses 6-8% less memory when compared to objective C for the problem statement designed in MVC.

On an average the Swift programming language performs better for MVC design pattern. This is specific to the problem under study. The difference in performance will be similar when the problem statement is changed. The Data flow between the view components is least here as everything is present in Controller and the view is directly referenced from the controller. Also the Model is created in the controller and passed on from one controller to another when navigation takes place.

MVVM : The MVVM design pattern has the ViewModel which requires more classes and class interaction between the view and the model. Due to which a lot of repetition needs to be performed. This causes more CPU and memory to be consumed in Swift because many constants in objective C needs to be made a variable because of Swift's error handling capability.

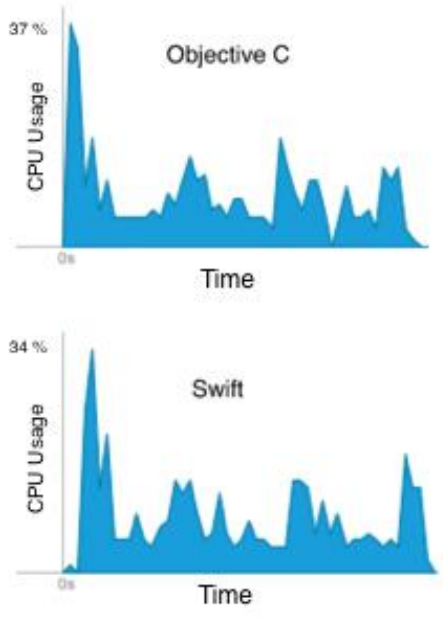

Fig. 2 CPU Analysis for MVC

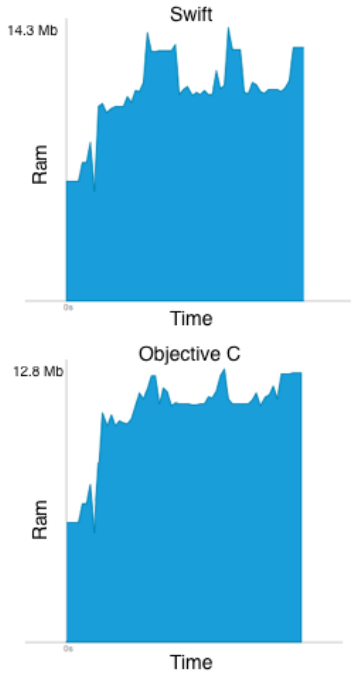

Fig. 3 Memory Analysis for MVC

Objective C has 2 % lower usage of CPU when compared to Swift. So Objective C is better when implemented in MVVM. This is mainly due to the higher complexity between view and model in MVVM. On An average the Objective C programming language uses 5-7% less memory when compared to Swift for the problem statement designed in MVVM. The Objective programming language performs better for MVVM design pattern. This is specific to the problem under study. The Data flow between the view components is more here as there needs to be communication between view and view model for each and every data element which the view requires needs to be passed from the view model.

MVP : In MVP design pattern the data is passed to view with the help of presenter. Also the data is requested by the presenter with the help of interactor. But this drawback is compensated by the fact that there is no strict binding of model to presenter. Objective C has 2 % lower usage of CPU when compared to Swift. So Objective C is better when implemented in MVVM. This is mainly due to the higher complexity between view and model and the interactor in MVP. On An average the Objective C programming language uses 5-7% less memory when compared to Swift for the problem statement designed in MVP.

TABLE I
CPU USAGE

Language	**Design Pattern**		
	MVC	**MVVM**	**MVP**
Objective - C	37 %	14%	15 %
Swift	34 %	16 %	17 %

TABLE 2
MEMORY USAGE

Language	**Design Pattern**		
	MVC	**MVVM**	**MVP**
Objective - C	14.3 Mb	11.5 Mb	11.6 Mb
Swift	12.8 Mb	12.1 Mb	12.2 Mb

VI. CONCLUSION

We can conclude that Swift performs better if the complexity of code is less and the data flow among the components is less. This is mainly because Swift is still a language under development and the compiler being used for the development is still not completely optimised for swift language. Apple is improving the language and going forward swift might be as good or better than objective c even for projects with higher complexity.

REFERENCES

1. Artem Syromiatnikov and Danny Weyns, A Journey Through the Land of Model-View-* Design Patterns, 2014 IEEE/IFIP Conference on Software Architecture
2. Wikipedia - https://en.wikipedia.org/wiki/Model-view-controller
3. Visual Works Tutorials - http://www.cincomsmalltalk.com/main/products/visualworks/visualworks-tutorials/
4. Martin Fowler, Patterns of Enterprise Application Architecture, http://martinfowler.com/eaaDev/PresentationModel.html
5. The Swift Programming Language (Swift 2.2) - https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/
6. Programming with Objective-C - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/Programming WithObjectiveC/Introduction/Introduction.html
7. Adele Goldberg and David Robson, Smalltalk-80 - The Language and its Implementation, Addison-Wesley Publishing Company.

8. S. Burbeck, Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc), Smalltalk-80 v2. 5. ParcPlace, 1992.
9. Web MVC framework: http://docs.spring.io/autorepo/docs/spring/3.2.x/spring-framework-reference/html/mvc.html
10. iOS App Architecture - https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html
11. An explanation of the MVC pattern in traditional GUI applications - http://csis.pace.edu/~bergin/mvc/mvcgui.html
12. Patterns - WPF Apps With The Model-View-ViewModel Design Pattern by John Smith, February 2009
13. Jean-Paul Boodhoo, Design Patterns column, http://blog.developwithpassion.com/2006/07/msdn-article-model-view-presenter
14. Glenn Block, Prism: Patterns for Building Composite Applications with WPF, https://msdn.microsoft.com/en-us/library/ff648612.aspx
15. Data Flow Diagram - https://en.wikipedia.org/wiki/Data_flow_diagram

