Comparing Model-View-Controller (MVC) VS Model-View-Presenter (MVP) In Terms of Speed and Memory Usage

1st Alexander Dennis Ong
Pradita University
Information Technology 2022
Tangerang, Indonesia
alexander.dennis@student.pradita.ac.id

2nd Arnold Christopher Ghazali
Pradita University
Information Technology 2022
Tangerang, Indonesia
arnold.christopher@student.pradita.ac.id

3rd Okhama Siladata Devisepte
Pradita University
Information Technology 2022
Tangerang, Indonesia
okhama.siladata@student.pradita.ac.id

4th Brandon Josua Christian
Pradita University
Information Technology 2022
Tangerang, Indonesia
brandon.josua@student.pradita.ac.id

Abstract—make the abstract into 1 paragraph The choice of architectural patterns significantly influences the performance and memory efficiency of software systems. In this paper, we present a comprehensive comparative analysis between two widely used architectural patterns: Model-View-Controller (MVC) and Model-View-Presenter (MVP). Our study aims to provide insights into which architectural pattern is superior concerning speed and memory usage, crucial factors in determining the scalability and responsiveness of modern applications.

To conduct a fair evaluation, we implemented identical applications using both MVC and MVP architectures. We developed benchmarks to measure the speed of execution and memory footprint of each pattern under various scenarios, including different input sizes and complexities.

Furthermore, we discuss the implications of these findings on software development practices and provide recommendations for selecting the most suitable architectural pattern based on specific project requirements. By offering empirical evidence and insights into the performance attributes of MVC and MVP, this study contributes to informed decision-making in architectural design, ultimately enhancing the overall quality and efficiency of software systems.

(*BELOM SELESAI*)

I. Introduction

Introduction is too short, add one or more paragraf such as how MVP or MVC has been created and who created it, the origin of MVP or MVC, example of use the MVC or MVP which is lined with the introduction

Software development has undergone a significantly significant evolution in recent decades, with the emergence of various design patterns to build more structured and maintainable applications.

In software development, Model View Controller (MVC) and Model View Presenter (MVP) are two frequently debated design patterns. MVC, which emerged in the 1970s, organizes applications into Model (data), View (presentation), and Controller (logic) components. On the other hand, MVP is a variation of MVC that emerged in the early 2000s, introducing the Presenter as an intermediary between the Model and View to separate presentation logic from business logic. Comparing these two patterns is crucial for a better understanding of their strengths, weaknesses, and suitable applications in software development.

This research aims to delve into the comparison between the Model View Controller (MVC) and Model View Presenter (MVP) design patterns in software development. Through in-depth analysis of these two design patterns, it is hoped that the strengths, weaknesses, and optimal use cases for each pattern can be identified. Consequently, this study aims to provide valuable guidance for software developers in selecting the design pattern that best suits the needs of their projects.

II. Related Studies

The MVC Model

A. Understanding The MVC Architecture Model

Utilizing graphical user interface (GUI) libraries simplifies the implementation of the model-view-controller (MVC) design pattern. Initially proposed by Trygve Reenskaug, a Smalltalk developer at Xerox Palo Alto Research Center in 1979, MVC facilitates the separation of data access and business logic from the presentation to the user [Fig 1]. To elaborate, the following figure is the representation of how to MVC Works [1].

The Model Role is to act as an object that represent a data or a process of a program for example as a database and a process for the machines to act, application logic belong to this section [2]. a model functions as a digital

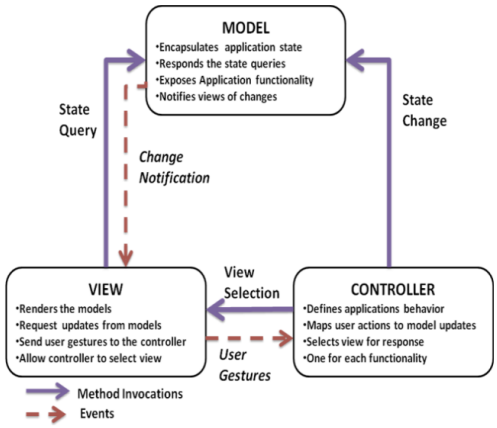

Fig. 1. MVC Model [1]

representation of real-world processes, embodying both the data and the regulations dictating its access and modification [1].

The View is a form of presentation of what the user will see when it request something, a user interface belong to this category [2].

The view displays the information contained within a model and dictates the precise manner in which this data should be shown. When there are alterations to the model data, the view adjusts its presentation accordingly. This adjustment can be accomplished through either a push model, where the view subscribes to the model for updates, or a pull model, where the view actively requests the latest data from the model [1].

The controller interprets the user's interactions with the view and translates them into operations that the model will execute. In a standalone graphical user interface (GUI) client, these interactions might include clicking buttons or selecting menu options, while in an enterprise web application, they typically manifest as GET and POST HTTP requests. Depending on the situation, the controller might also determine which view to present to the user next, such as a page displaying search results in a web application [1].

B. An example of how the MVC Works

This is an example of a web browser about a movie. The user in this example want to see list of movies available on the web and this is how the MVC works[fig 2].

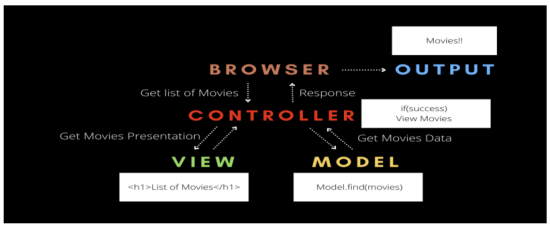

Fig. 2. MVC flow example [3]

So in this case the user input the data about a list of movies this is what happens. In the process of retrieving a list of movies, the browser initiates a request to the Controller, which subsequently communicates with the Model to fetch the desired data from the primary database. Once the Model retrieves the list of movies, it delivers it to the Controller. If the Controller successfully obtains the list, it directs the View to render the movies in HTML format. The View then generates the HTML representation of the movie list, which is returned to the Controller. Finally, the Controller presents this HTML to the user, resulting in the display of the list of movies [3].

C. MVC utilization

- MVC or model view controller is highlighted for its ability to streamline horizontal development and maintenance of large scale distributed web application across three frameworks: Big Blob, MVC architecture, and modified MVC Design.
- The Model-View-Controller (MVC) architecture is easier to implement graphical user interface (GUI) libraries because it provides a true decoupling of each part. So it is easier to change the odel part and view will notify the update the model does not carry a refenrence to the view but instead uses an event-notification model to notify interested parties of a change. One of the problem using this powerful design is that the many views can have the same underlying model. When a change in the data model occurs, each view is notified by a property change event and can update itself accordingly.
- The Modifying MVC architecture uses Cocoa Touch frameworks that power Mac OS X and iOS are tightly integrated into the Xcode development experience. Cocoa's high-level APIs make it easy to add animation, networking, and the native platform appearance and behavior to your application with only a few lines of code. In this Structure models encapsulate application data, Views display and edit that data, and Controllers mediate the logic between the two. By separating responsibilities in this manner, you end up with an application that is easier to design, implement, and maintain
- The MVC architecture was able to solve some of the problem of web and internet programming but still there were a lot of things missing from it. It was centred on the navigation of the JSP pages so there was the scope of the further development in the architecture point of view. During this process the next development was the Model 2 architecture. This problem was solved using the Servlet and JSP together. The Servlet handles the Initial request and partially process the data. It set up the beans then forward the result to the one of the JSP page. The Servlet decide the one of the page to be displayed from the list of pages

D. Advantages and Disadvantages of MVC

1) Advantages of MVC
   - Separation of Concerns: MVC enforces a clear separation between the data (Model), the presentation logic (View), and the application logic (Controller). This separation makes the codebase easier to understand, maintain, and extend.
   - Modularity: Each component in MVC can be developed independently of the others. This modularity allows developers to work on different parts of the application simultaneously without interfering with each other's work.
   - Reusability: Components in MVC can often be reused across different parts of the application or even in entirely different applications. For example, a controller logic that handles user authentication can be reused in multiple views.
   - Testability: Because of its separation of concerns, each component in MVC can be unit tested independently, which makes it easier to write automated tests for the application.
   - Scalability: MVC promotes a structure that scales well with the size and complexity of the application. As the application grows, developers can add more models, views, and controllers without significantly impacting existing code.
2) Disadvantages of MVC
   - Complexity: Implementing MVC requires a good understanding of the pattern and its principles. For developers unfamiliar with MVC, there may be a learning curve associated with adopting this architectural pattern.
   - Overhead: MVC can introduce some additional overhead compared to simpler architectures, particularly in terms of initial setup and configuration. In some cases, MVC might be overkill for small or simple applications.
   - Tight Coupling: While MVC aims to reduce dependencies between components, it's still possible to introduce tight coupling between the components, especially if not implemented properly. Tight coupling can make the code harder to maintain and refactor.
   - Potential for Code Duplication: Without proper discipline, developers might duplicate code across different components of the MVC architecture, leading to maintenance issues and inconsistencies.
   - Increased Complexity in Asynchronous Applications: Asynchronous programming and handling asynchronous requests in MVC can sometimes introduce additional complexity, especially in scenarios where the flow of control is not linear.

The MVP Model

E. Understanding The MVP Architecture Model

The MVP is designed around two questions: data management and user interface. It's about how data is managed and how users interact with the data. There are six components that recognized in MVP architecture: model, selections, commands, presenter, interactor and view. The model component is like the concept in MVC that indicates what data is present in this application. The Selection element specifies the subset of the data to be used. The Command component presents a list of actions that can be performed. The View component is the same in MVC. The Interactor component indicates which events should be triggered by the user, e.g. keyboard, mouse movements and clicks, scrolling. The Presenter component is like a controller in MVC that is intended to organize and coordinate all the intermediary components – interactors, options, and commands. The first three components are for data management and the last three part is for the user interface.

Most of the time MVP architecture is described as consisting 3 components: model, view and presenter. Interactors, controls, and selections are placed in the presenter component. With MVP, the coupling between view and model is eliminated. The model knows nothing about the presenter and vice versa [4]. MVP model interaction are desribed in Figure 3.

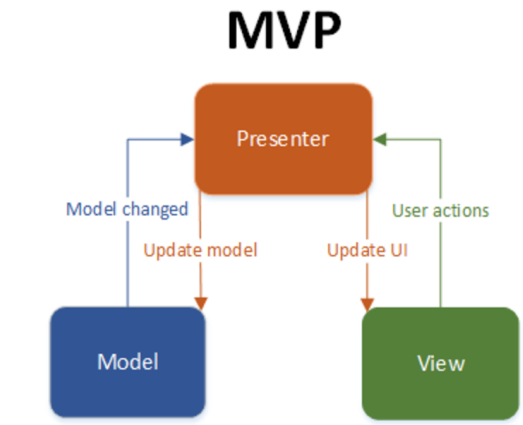

Fig. 3. MVP Model

The flow of MVP starts from view component. View capture user actions and send them to presenter. For a simple action, presenter makes decision and update the view. If this a complex action, the presenter will send message to the model to get the relevant data and then update the view. Sometimes user action impacts model data. In this case the View will not directly affect the model but requires the presenter to update the model [4].

F. Advantages and Disadvantages of MVP

1) Advantages of MVP

- MVP aims to reinforce the separation between the main components of the application: the model, view, and presenter. With clear separation, developers can more easily manage and maintain code as each component has well-defined responsibilities. This helps avoid complex code stacks and facilitates additional development or changes in the future [5].
- The MVP is its ability to support automated testing of the user interface. By separating presentation logic from the view, unit testing can focus on the presenter without directly involving the view. This facilitates the testing process and allows developers to effectively validate application behavior automatically [5].
- MVP variations, especially in the form of Passive View, offer significant testing convenience. By separating presentation logic from the view, unit testing can be done in isolation for the presenter and model. This reduces dependency on the view and enables more focused and efficient testing, helping to identify and fix bugs more quickly [5].
- Both Supervising Controller and Passive View patterns in MVP have a significant testing footprint. This means that applications built using these patterns have broader testing capabilities, allowing developers to conduct comprehensive testing and better maintain applications [5].
- MVP is a design pattern that separates the presenter, view, and model in an application, providing greater flexibility in dividing responsibilities within it. Each component has well-defined roles, allowing developers to efficiently manage the application logic and modify or extend functionality without disrupting other components. This helps strengthen the application architecture and makes it easier to organize and maintain over time [5].

2) Disadvantages of MVP

- The drawback of MVP is the potential for low re-usability because the presenter is tasked with updating the view. This could result in code that is challenging to reuse across various sections of the application [5].
- A disadvantage of MVP is the likelihood of low separation of concerns, particularly in the Passive View pattern, where the presenter is responsible for view updates. This may lead to increased code complexity and impede application maintenance [5].
- An issue with MVP is the difficulty in maintaining the application, especially in the Passive View pattern, due to potential code complexity. This challenge arises primarily when the presenter is responsible for updating the view [5].

G. Difference Between MVC and MVP Architecture

In MVP (Model-View-Presenter), the view directly communicates with the presenter, making it easy to supply user information for a specific model. The role of the view is strictly for displaying data and does not implement any business logic. Multiple views can be associated with the same presenter, allowing for flexibility in the user interface [6].

In contrast, MVC (Model-View-Controller) allows for multiple controllers to handle views, especially in web applications where different events like mouse clicks or keyboard inputs may trigger actions. The controller manages the interaction between the model and the view [6].

The presenter in MVP acts as an intermediary between the view and the model, reading and retrieving data from the model through the view interface. This separation ensures that business logic remains within the presenter and model layers [6].

The model's role is consistent across both MVP and MVC, containing the business data and functionality associated with it. However, in MVP, the model does not have a direct reference to the presenter, leading to the need for page refreshes to update data, unlike MVC where updates can occur without refreshing the page [6].

References

[1] T. Dey, "A comparative analysis on modeling and implementing with mvc architecture," International Journal of Computer Applications, vol. 1, pp. 44–49, 2011.

[2] R. N. Thakur and U. Pandey, "The role of model-view controller in object oriented software development," Nepal Journal of Multidisciplinary Research, vol. 2, no. 2, pp. 1–6, 2019.

[3] P. Rustagi and Y. Kumar, "Mvc architecture and its application," 2022.

[4] T. Lou et al., "A comparison of android native app architecture mvc, mvp and mvvm," Eindhoven University of Technology, 2016.

[5] S. Lappalainen, "A pattern language for mvc derivatives," 2017.

[6] M. Qureshi and F. Sabir, "A comparison of model view controller and model view presenter," arXiv preprint arXiv:1408.5786, 2014.