

Jennifer Ada Assignment 1 2017-18 Advanced Web Development Report

by Jennifer Ada

FILE	3017_JENNIFER_ADA_JENNIFER_ADA_ASSIGNMENT_1_2017-18_ADVANCED_WEB_DEVELOPMENT_REPORT_93283_1685372856.DOCX (582.43K)		
TIME SUBMITTED	07-FEB-2018 11:23PM (UTC+0000)	WORD COUNT	2391
SUBMISSION ID	82922011	CHARACTER COUNT	16186

ADVANCED WEB DEVELOPMENT: ASSGNMENT 1 JENNIFER ADA

Contents

Introduction	3
The MVC pattern in web applications.....	4
The Model	4
The View.....	4
The Controller	4
MVC Example	5
Benefits and Disadvantages of MVC	6
Presentation-Abstraction-Control.....	7
Presentation	7
Abstraction	7
Control	7
PAC Example	8
Model-View-Presenter.....	9
Model: "What is my data?"	9
View: "How do I display my data"	9
Presenter: "How do I put it all together?"	9
Interactor: "How do events map into changes in my data?".....	9
Commands: "How do I change my data?"	9
Selections: "How do I specify my data?"	9
MVP Example	10
Benefits of PAC and MVP	11
Conclusion.....	12
References	13

Introduction

Design patterns are inherently good practice for web applications[1]. As web application functionality in terms of complexity and sophistication continue to increase, it only makes sense to abstract and codify practices for the economies of reuse and the insurance of proven practices.

As Moore's Law[2] continues to push hardware towards more distributed[3], multi-core[4] and asymmetric processing architecture[5] further challenges present themselves in terms of both programming these applications to mitigate wasted processing resources, and assurance that threads etc. do not fail due to concurrency issues. These can all be solved with architectural patterns[6].

The MVC pattern in web applications.

First introduced as a design pattern in the late 70's by Trygve Reenskaug[7]. Model-View-Controller (MVC) takes advantage of the separation of concerns to decouple the input, output and processing / storage of data into components which interface in a structured manner. As the name would imply, the pattern consists of three interdependent components, sometimes referred to as a triad. The components cannot be summarised more succinctly than Reenskaug[7]:

The Model

"The model manages the behaviour and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller)."

The View

"The view manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application."

The Controller

"The controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate."

See Figure 1 for a high-level diagram of the MVC pattern:

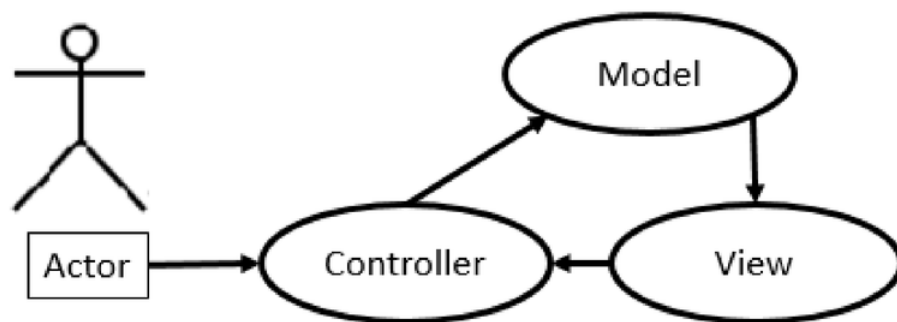


FIGURE 1 USE-CASE DIAGRAM FOR MVC[8].

MVC Example

It can be difficult to understand from a description so an example of its use as an analogy may help to comprehend it more effectively (See Figure 2 for illustration):

Assume that a website allows the user to rate their favourite critics a number from 0 to 10:

1. After selecting their rating from a dropdown box, they click a button to submit their selected rating.
 - The controller handles the user input event from the button.
 - The controller sends the data from the dropdown box to the model.
2. The model adds the data to a database of ratings for different critics.
 - It updates the state of the model.
3. The view inspects the rating data managed by the model.
 - Perhaps updating the new sum of ratings for the critic which was rated.

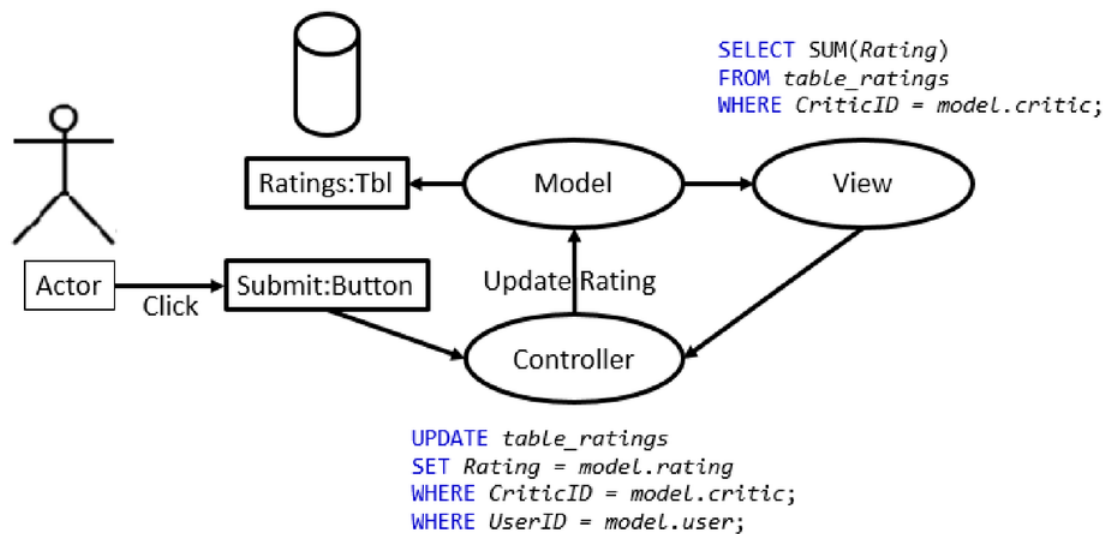


FIGURE 2 USE-CASE DIAGRAM FOR A CRITIC RATING WEB APPLICATION USING MVC.

Benefits and Disadvantages of MVC

One of the primary benefits to using the MVC design pattern properly is the adhesion to the DRY principle. DRY is an acronym for “Don’t Repeat Yourself” and encourages code reuse where feasible, barring edge cases where optimisation necessitates copies of code be in more than one place. This is not exclusive to PHP and also appears in Ruby on Rails amongst countless other web languages[9].

The DRY principle is also not exclusive to MVC, inheritance in object-oriented-programming (OOP) for example also adhere to DRY. This is most explicit in the MVC framework in the way in which inheritance is used to create instances of pairs of views and controller which can use the same model.

An example of this would be an authorisation interface for the critic rating website. Administrators that log in could be presented with a view and controller which allow them to add new critics to the database. In contrast, non-administrators would be presented with a view and controller which allows them to rate critics, effectively updating the critic table or associated table in the same model.

In addition and complementary to the DRY principle, orthogonality is also identified as an important benefit of the MVC pattern for web applications[10]. Whereas DRY avoids duplication of code, orthogonality avoids interdependency among code. These two principles combined make MVC “flexible, more understandable, and easier to debug, test, and maintain.”[10]

Unfortunately, MVC has been a victim of its own popularity[10]. Many frameworks which claim to use MVC or similar architectural patterns do not adhere to design principles the perusal of qualities espoused by MVC can also lead to unnecessary decoupling and overcomplication of the solution when abused[8].

Presentation-Abstraction-Control

An alternative to MVC is the Presentation-Abstraction-Control (PAC) pattern introduced by Joëlle Coutaz in 1987[11]. This architectural pattern also takes advantage of a structure that is suitable for web applications especially regarding interactivity and data handling like MVC, but it uses a hierarchy of agents in combination with the triad structure.

Each agent is responsible for the specific aspect of applications functionality so one agent may be responsible for a graph of the data in one client while another agent is responsible for a chart in another client[6]. As the name would imply, this pattern also consists of three interdependent components, sometimes referred to as a triad. They are the following[11]:

Presentation

“the input and output behaviour of the application as perceived by the user”

Abstraction

“implements the functions that the application is able to perform.”

Control

“maintains mapping and the consistency between the abstract entities (involved in the interaction and implemented in the Abstract part) and their presentation to the user. “

The inter-agent communication is better displayed visually as in Figure 3.

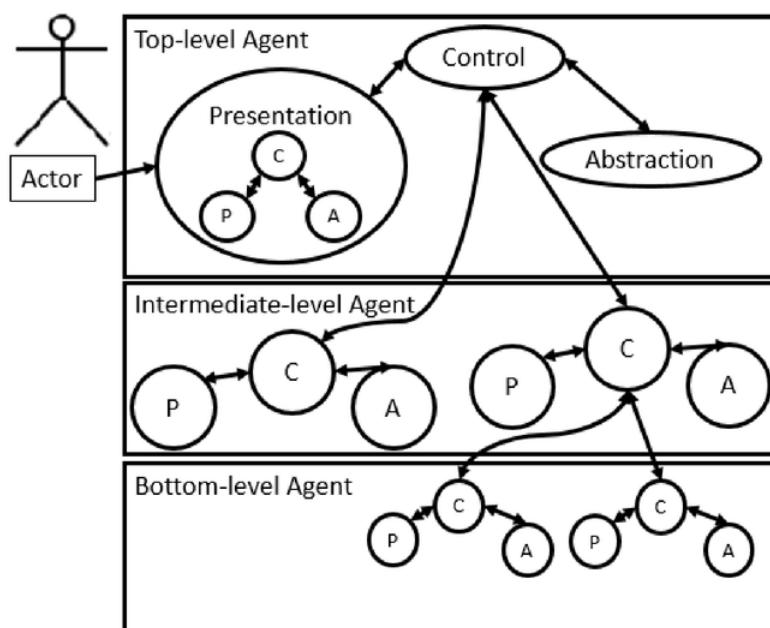


FIGURE 3 USE-CASE DIAGRAM FOR PAC ILLUSTRATING COMMUNICATION BETWEEN PAC COMPONENTS VIA THEIR CONTROL COMPONENTS IN THE HIERARCHY[12].

PAC Example

Using the previous example for MVC to illustrate PAC may aid in clarifying how it works (See Figure 4 for illustration):

Assume that a website allows the user to rate their favourite critics a number from 0 to 10:

1. After selecting their rating from a dropdown box, they click a button to submit their selected rating.
 - The bottom-level controller for the button handles the user input event from the button.
 - The intermediate-level controller sends the data from the dropdown box to the Top-level agent.
2. The top-level controller sends a signal to the abstraction component to add the data to a database of ratings for different critics.
 - The top-level abstraction component of the agent updates the state of the model.
3. A bottom-level agent presentation component inspects the rating data through the control component propagating calls up the hierarchy again through the control components.
 - Perhaps updating the new sum of ratings for the critic which was rated.

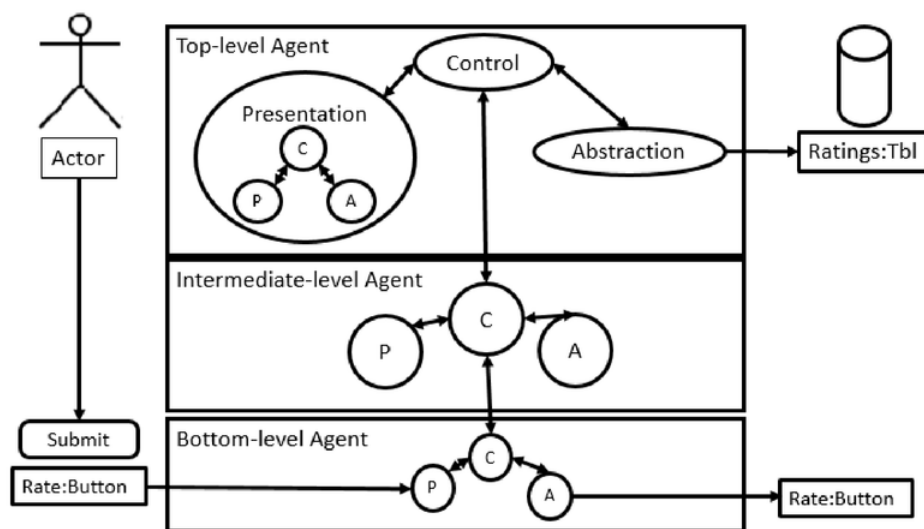


FIGURE 4 USE-CASE DIAGRAM FOR A CRITIC RATING WEB APPLICATION USING PAC.

Model-View-Presenter

The final alternative to be examined in contrast to MVC and PAC is the Model-View-Presenter (MVP) architectural pattern introduced by Mike Potel in a paper for IBM in 1994[13]. This architectural pattern also takes advantage of a structure that is suitable for web applications based on a generalization of MVC targeted at SmallTalk.

MVP is a design methodology for application and component development tasks especially focused on the ability to adapt to multiple client/server and multi-tier application architectures like PAC but with a structure like MVC and emphasis on the separation of data and model that has been fundamental to previously discussed patterns. The components are as follows and are presented in Potels paper in a series of requirements questions:

Model: "What is my data?"

The model is responsible for the data referred to as "encapsulated data, read and write access methods, etc.".

View: "How do I display my data?"

Views display interactors, facets of the model but care must be taken to remember that they may also be used for non-visual elements of a model.

Presenter: "How do I put it all together?"

Presenters control which commands are activated when an interactor is activated by a user event.

Interactor: "How do events map into changes in my data?"

Interactors deal with user events such as mouse clicks and keyboard key presses.

Commands: "How do I change my data?"

Commands are activated by the presenter such as "delete, move, copy, change the values, save, or print" on selections of the model.

Selections: "How do I specify my data?"

Selections are "abstractions for specifying different subsets" in the model, E.g. data tables, columns and rows, etc.

This structure and the relationship between the expanded triad is better displayed visually as in Figure 5.

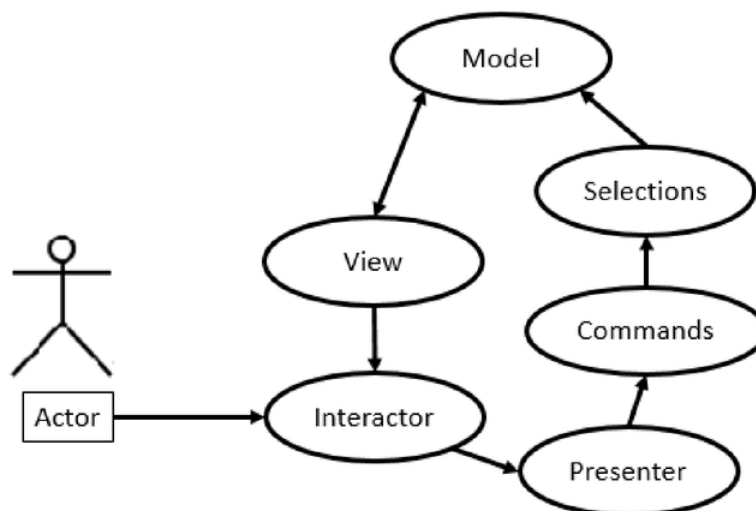


FIGURE 5 USE-CASE DIAGRAM FOR MVP [13].

MVP Example

Using the previous example for MVC and PAC may aid in clarifying how it works (See Figure 6 for illustration):

1. They click a button to submit their selected rating.
 - The interactor for the button handles the user input event from the button.
 - The interactor sends a signal to the presenter.
2. The presenter chooses the appropriate command and selection of the model to add the data to a database of ratings for different critics.
3. The view component is updated with data from the model.

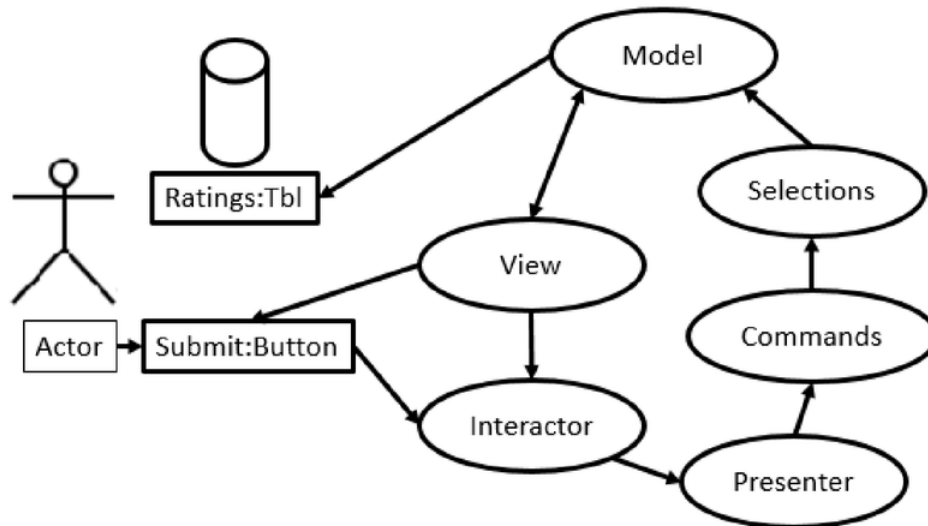


FIGURE 6 USE-CASE DIAGRAM FOR A CRITIC RATING WEB APPLICATION USING MVP.

Benefits of PAC and MVP

One of the primary strengths of PAC is that user interface components such as data displays and controls on forms are separated from the agents' function and communication with other agents in the hierarchy. Different views and behaviour can be slotted in where necessary and changes to the abstraction and presentation or even control component of an individual agent will have little to no effect on other agents.

PAC is also designed to support multi-tasking. The architecture of components allows PAC agents to be distributed. Different use-cases for distributed architecture include threads or processes amongst others and this makes it attractive for site back-ends that run on server farms which have scalable scope for parallelisation resources. Multi-tasking also benefits the front-end as multiple user can share different views of an application and interactivity is also supported due to different abstractions processing data and different presentations handling input and output concurrently.

MVP's strength can be broken down per component of the pattern but can be summarised as including flexibility and reusability of code due to generalisation of commands and independent components much like MVC.

In summary, there are similar characteristics between MVC and PAC in terms of their problem domain and therefore design as architectural patterns. The main difference between PAC and MVC is the allocation of input and output responsibilities to different components. For example, in MVC model, view and controller components are separate and work independently providing orthogonality and flexibility. In contrast, in PAC, input and output is combined in the presentation component[14].

In addition, while both models can be difficult to comprehend at first, the MVC model is simpler than the PAC model because the "relationships between objects can be interpreted as different views of the model"[14] while PAC does not. PAC model uses "hierarchy of control components that leads to more complex specification and implementation" [5]. Although less widely adopted IBM's MVP lends further granularity to the MVC model which may ease comprehension as it describes the interactions between components more explicitly.

PAC sacrifices the simplicity of MVC to facilitate parallelization and further modularisation of components. It is also worth noting that there is no inherent argument for MVC not being run in a parallelised manner although it could be argued that PAC and MVP are inherently designed for parallelisation.

Conclusion

Architectural patterns such as MVC, PAC and MVP can be used to solve problems inherent to web applications. The use of architectural patterns designed for handling the demands of interactive and multifunctional programs make much of the functionality possible to realise through modularisation and communication as part of their inherent design. As more distributed, multi-core and asymmetric processing architectures are developed, further challenges will present themselves in terms of both programming these applications to mitigate wasted processing resources, and assurance that threads etc. do not fail due to concurrency issues. As web application functionality in terms of complexity and sophistication continue to increase, it only makes sense to further develop practices for design patterns to ease the development and use of web applications. There are variations even among MVP, PAC and especially MVC but it is prudent to examine whether a solution is what it claims to be.

References

- [1] A. Ebneenasir and R. Beik, "Developing parallel programs: A design-oriented perspective," in *Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering, IWMSE 2009*, 2009.
- [2] G. E. Moore, "Cramming more components onto integrated circuits.," *Electronics*, vol. 38, no. 8, 1965.
- [3] T. Bashford and K. Donne, "PARALLELISATION TECHNIQUES FOR THE DUAL RECIPROCITY AND TIME-DEPENDENT BOUNDARY ELEMENT METHOD ALGORITHMS," *Int. J. Comp. Meth. Exp. Meas*, vol. 5, no. 3, pp. 395–403, 2017.
- [4] B. Venu, "Multi-core processors – An overview," p. 6, 2011.
- [5] S. Mittal and Sparsh, "A Survey of Techniques for Architecting and Managing Asymmetric Multicore Processors," *ACM Comput. Surv.*, vol. 48, no. 3, pp. 1–38, Feb. 2016.
- [6] G. Calvary, J. Coutaz, and L. Nigay, "From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW," pp. 242–249.
- [7] S. Burbeck, "Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC)."
- [8] M. Hameed, M. Abrar, A. Siddiq, and T. Javeed, "MVC Software Design Pattern in Web Application Development," *Int. J. Sci. Eng. Res.*, vol. 5, no. 5, 2014.
- [9] S. Meenakshi, "Ruby on Rails – An Agile Developer's Framework," *Int. J. Comput. Appl.*, vol. 112, no. 1, pp. 975–8887, 2015.
- [10] A. Hunt and D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, 1st ed. Addison Wesley, 1999.
- [11] J. Coutaz, "PAC, an Object Oriented Model for Dialog Design," *Human-Computer Interact. - INTERACT*, 1987.
- [12] J. Coutaz, "PAC-ing the Architecture of Your User Interface."
- [13] M. Potel, "MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java."
- [14] A. Hussey and D. Carrington, "Comparing two user-interface architectures: MVC and PAC," 1995.

Jennifer Ada Assignment 1 2017-18 Advanced Web Development Report

GRADEMARK REPORT

FINAL GRADE

32/100

GENERAL COMMENTS

Instructor

Investigation of design patterns (30) : 20

Investigation of MVC (20): 12

Quality of website and techniques used (30): 0

Login system (Observer pattern) (15): 0

Software testing (5): 0

Good discussson of design patterns, I particularly liked the discussion of the applications.

MVC briefly introduced and benefits/drawbacks analysis provided although a bit briefly.

No code submission.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13
