Theory – Design Pattern

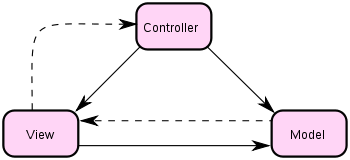
# Model View Controller

## Introduction

To ensure that code is correctly decoupled from one another strict design patterns are necessary, these allow for developing complex projects without losing the sight of the entire project. If no pattern is followed then code can easily become so entangled that later development might prove impossible. This section will cover all the rules and ideas behind the Model-View-Controller design pattern shortened MVC.

The MVC pattern principle is that programs that can be interacted with a user can be split into three different components. The three components are as follows:

* Model – Core of the program
* View – Visualization of the program
* Controller – Manager of state changes to the core of the program.



This image shows how the three components are connected to each other; the full arrows indicate that a component has complete knowledge of the component it is pointing to. A stripped arrow indicates that the component the stripped arrow is pointing to is listening to the component the arrow is originating from.

## Model

The model is the core of the program it is why the program functions as it does, it contains all information of the data and it is here all business logic is located. The model should have no knowledge of either the view or the controller, thus by not knowing either the program is ensured to not be tainted by their influence.

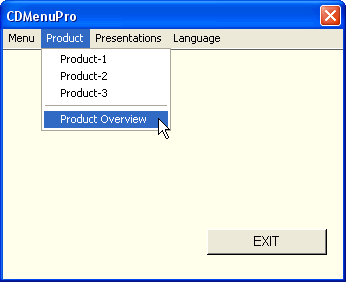
While may not know of the view or controller it is paramount that the model is built to optimally transfer information concerning its current state. That means putting in a way in the model for other components to listen to it is very welcome. Commonly this is referred to as events in most programming languages, what this does for the model is that in the case that model state has been altered it will have a way to provide the information of the state. If such features are not built into the model then it would require the component changing the state to inform of the state changes, in case of a MVC design. The component changing state is the controller and such the controller would both have the duty of changing the state and maintaining the view. This is generally the case of a badly designed model and can be completely avoided if the model simply has the ability to inform its listeners of any changes.

## View

The view is a way to visualize what is currently occurring inside model by visualizing it to the user, a view may take many shapes depending on the model. For instance if the model is a program calculating data on a server, then the view could take the form of a logging console. Or if the model was a computer game then the view would be the graphic representation of the game. Generally a view should only have knowledge of the model and not the controller, the idea is that if the model data is complete then interaction with the controller should not be necessary.

When designing a view there are some common pitfalls that happen easily if one is not careful of the design. First off the view is what it is named; a view this means that it should never do any state changes to the model. If getting hold of data means that the model must change its state to accommodate this then the model is poorly made and should be changed. However a view is allowed to change its own state without involving either the controller or model. To fully understand what is meant by this is easily understood with an example:

Assume you have a menu bar



To open a menu the user need to drag the mouse and click on one menu he or she wishes to open. However in this case many would confuse this to be the task of the controller. It is not since the changes done are only performed on the views own state and not the model of the program, thus it allowed to be its own controller.

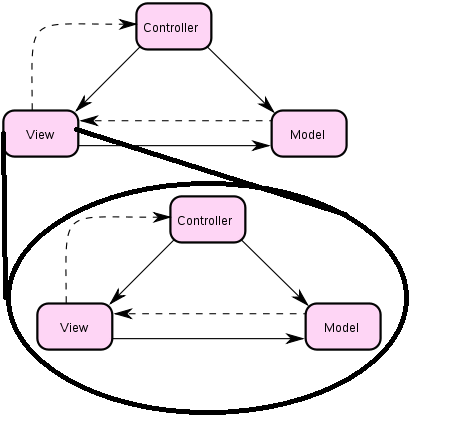
## Controller

The controlleris link between the model and the user, by convention all changes the user wish to do on the model should be done through controller. Like the view it can take many shapes, to name a few be that the controller directs the input from a mouse in a way that affect the model, however it could also be a controller that controls how a network data stream has effect on the model.

In a well-designed program the controller should never have to interact with the view, however this can be practically impossible on larger projects unless carefully planned and as such the controller by convention is allowed to know of both the view and the model.

A common mistake when designing the controller is to mistake the unit which the controller gets input from as the actual controller. In many cases the keyboard is the device from which input is transformed into state changes to the model, however that does not mean that the controller should be the only unit interacting with the keyboard. Going back to the example used to understand the view, the reason why controller should not deal with opening a menu bar on the view is because the controller is not responsible on the state of the view. The controller is only responsible for the state of the model; the only case in which it is allowed for the controller to interfere with the view is in the case that the model was unsuccessful in properly informing about its state change caused by the controller. In this case it is okay for the controller to call the view and ask it to adjust itself.

The reasoning for why the controller is normally mistaken to be responsible for handling changes to the state of the view is because it is mistakenly thought of as a controller for the entire program and not the model, a view may contain its own controller which should not be mistaken from the other controller, to fully understand this imagine that the view in itself also contains a MVC inside itself.



The model of a view such as a menu bar would contain data about the names of the menus and it would be responsible for ordering and accessing information as to what each menu contains. Its view would be that of a drawing board responsible for properly drawing the menu bars. Luckily most views are simple so one does not need to make an entire MVC design, but for graphical user interface used in most operating system it is very important to understand that a view can be an entire MVC setup in itself, this is why most operating system comes with libraries to easily design GUI.