Over All Idea about MVC: How to use Model-View-Controller (MVC)

Parth Jivani
B. H. Gardividyapith Engg. &Tech.

Chhaya Chopara
B. H. Gardividyapith Engg. & Tech.

Mehta Prashant
B. H. Gardividyapith Engg. & Tech.

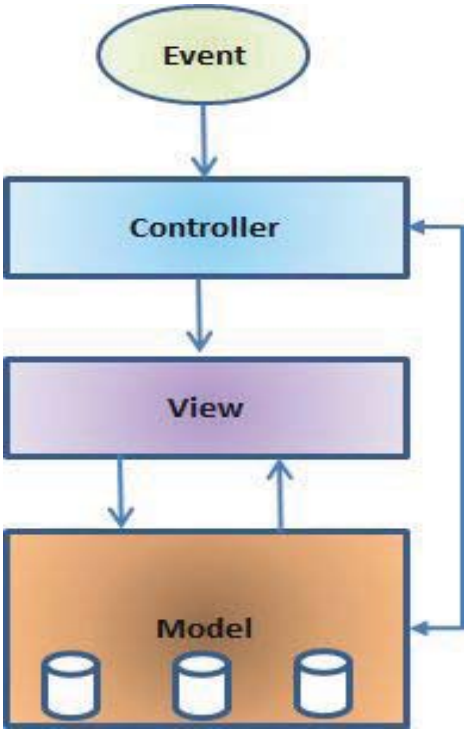
Abstract— Model View Controller (MVC) is a standard Design pattern in web domain and is known for its extensibility, maintainability, re-usability and testability capabilities. Yet it is new in the embedded domain. But in the time of fast emerging and user driven technologies there is a requirement of this approach in embedded soft wares. In this paper, I am presenting an approach to implement MVC Architecture in embedded domain. In this approach Embedded Linux kernel and its IPC infrastructure are used as a framework and python is used for quick modeling and implementing Object threads.

Keywords— MVC Design pattern, Embedded systems, Embedded Linux, Python, Enlightenment, IPC, XML.

I. INTRODUCTION

In today's era Embedded Systems Firmware are increasingly becoming cumbersome and complex having lot of functionalities (Functionalities like GUI, network monitoring, Remote Configuration, local Database etc.).For every, functionality lot of libraries are available off the shelf. For example, for GUI applications GTK, Enlightenment, QT etc. As an embedded developer, One have to select the most optimum solution for its application depending on various parameters (e.g.: memory footprint vs. User Experience) and have to be ready for the new emerging libraries delivering smiler functionality but having greater optimization. One also requires extensibility, maintainability and testability in the code. These difficulties give scope to implement a better model in embedded systems. MVC model had presented a successful architecture in the web and enterprise domains. It proposed architecture comprised of different process/modules taking care of business logic, GUI and Interrupts. This architecture provides a modular approach of development. Modules can be develop parallel hence faster development time. It also GENERATES reusable code (same thread can serve more than two processes.) and the design is lot more expressive. It also gives freedom in switching between different technologies giving similar functionality. For example for GUI one can select between GTK, Enlightenment or QT at any time during the development (as one have to change only one python script managing that technology). This methodology shows lot of scope for implementing it into Embedded Systems. The paper is a discussion of implementation of MVC Architecture in an Embedded System project.

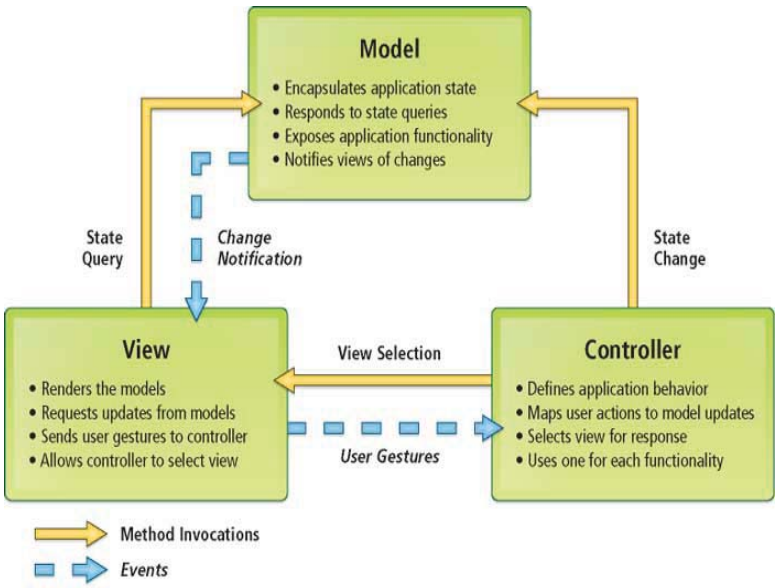
II. MVC ARCHITECTURE



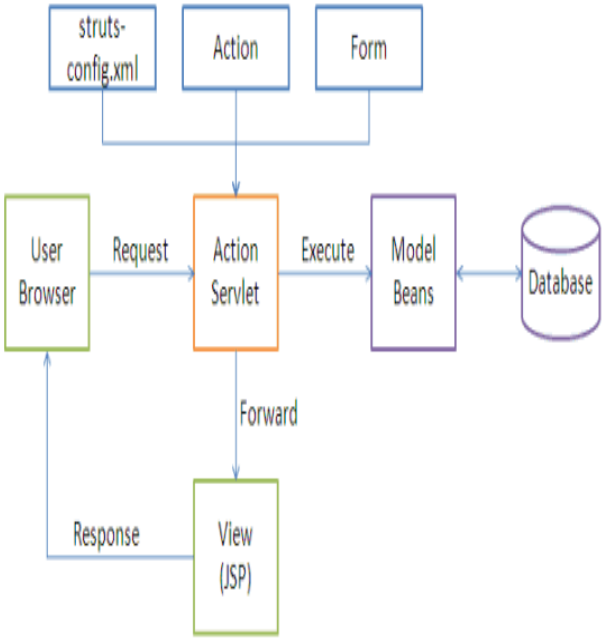
The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user.

Here are the reasons why we should use the MVC design pattern.

1. They are **reusable**: When the problems reoccurs, there is no need to invent a new solution, we just have to follow the pattern and adapt it as necessary.
2. They are **expressive**: By using the MVC design pattern our application becomes more expressive.



1) **Model:** The model object knows about all the data that need to be displayed. It is model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data.
2) **View:** The view represents the presentation of presentation data and how that data will be displayed to the browser the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the of the view's to maintain the consistency in its presentation when the model changes.
3) **Controller:** Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUIs, the views and the controllers often work very closely together.



2.1 Difference between Model 1 and Model 2 architecture:

2.1.1 Features of MVC1:

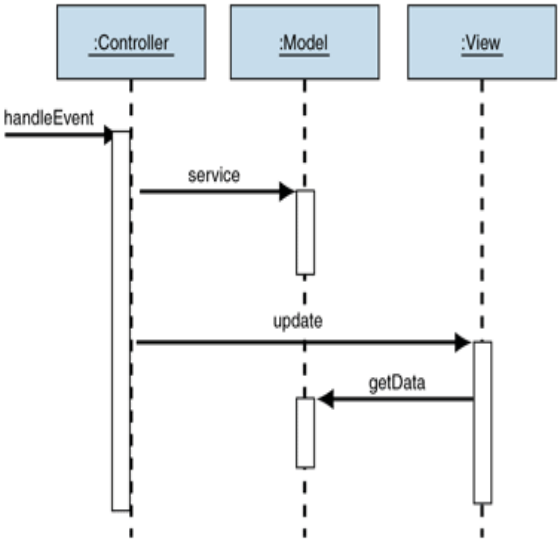
1. Html or jsp files are used to code the presentation. To retrieve the data JavaBean can be used.
2. In mvc1 architecture all the view, control elements are implemented using Servlets or Jsp.
3. In MVC1 there is tight coupling between page and model as data access is usually done using Custom tag or through java bean call.

2.1.2 Features of MVC2:

1. The MVC2 architecture removes the page centric property of MVC1 architecture by separating Presentation, control logic and the application state.
2. In MVC2 architecture there is only one controller which receives all the request for the application and is responsible for taking appropriate action in response to each request.

2.1.3 Passive Mode:

The passive model is employed when one controller manipulates the model exclusively. The controller modifies the model and then informs the view that the model has changed and should be refreshed (see Figure 2). The model in this scenario is completely independent of the view and the controller, which means that there is no means for the model to report changes in its state. The HTTP protocol is an example of this. There is no simple way in the browser to get asynchronous updates from the server. The browser displays the view and responds to user input, but it does not detect changes in the data on the server. Only when the user explicitly requests a refresh is the server interrogated for changes.



2.1.4 Active Model:

The active model is used when the model changes state without the controller's involvement. This can happen when other sources are changing the data and the changes must be reflected in the views. Consider a stock-ticker display. You receive stock data from an external source and want to update the views (for example, a ticker band and an alert window) when the stock data changes. Because only the model detects changes to its internal state when they occur, the model must notify the views to refresh the display.

However, one of the motivations of using the *MVC* pattern is to make the model independent from of the views. If the model had to notify the views of changes, you would reintroduce the dependency you were looking to avoid. Fortunately, the *Observer* pattern [Gamma95] provides a mechanism to alert other objects of state changes without introducing dependencies on them. The individual views implement the *Observer* interface and register with the model. The model tracks the list of all observers that subscribe to changes. When a model changes, the model iterates through all registered observers and notifies them of the change. This approach is often called "publish-subscribe." The model never requires specific information about any views. In fact, in scenarios where the controller needs to be informed of model changes (for example, to enable or disable menu options), all the controller has to do is implement the *Observer* interface and subscribe to the model changes. In situations where there are many views, it makes sense to define multiple subjects, each of which describes a specific type of model change. Each view can then subscribe only to types of changes that are relevant to the view.

Figure shows the structure of the active MVC using *Observer* and how the observer isolates the model from referencing views directly.

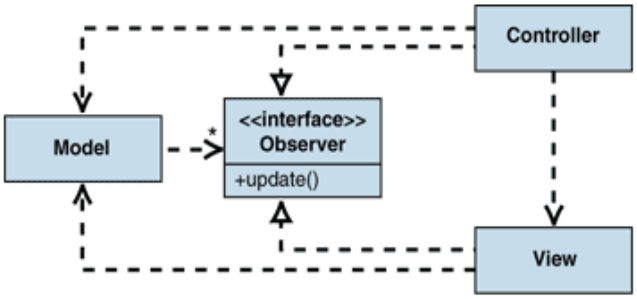
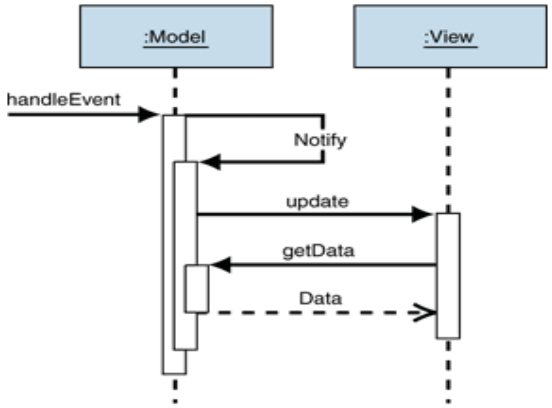


Figure 3: Using Observer to decouple the model from the view in the active model

Figure illustrates how the *Observer* notifies the views when the model changes. Unfortunately, there is no good way to demonstrate the separation of model and view in a Unified Modeling Language (UML) sequence diagram, because the diagram represents instances of objects rather than classes and interfaces.



III. CONCLUSIONS

By using MVC we can develop a many type of application.
It is very use full to create a any type of application. We understand how it work at the platform of the develop the application.

ACKNOWLEDGMENT

In the MVC introduction we can understand how the it's work like,
Model: The model object knows about all the data that need to be displayed.
View: The view represents the presentation of presentation data and how that data will be displayed to the browser the application.
Controller: The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action.

REFERENCES

[1] *en.wikipedia.org/wiki/Model–view–controller*
[2] *samet.kilictas.com/what-is-**mvc-architecture**-model-view-controller/*
[3] *msdn.microsoft.com/en-us/library/ff649643.aspx*
[4] *en.wikipedia.org/wiki/**MVC***
[5] mvc architecture ppt
.