Introduction - Model, View, ViewModel (MVVM pattern) AKA Model, View, Binder

This document contains quotes from multiple sources on the definition of the MVVM architectural pattern. It became quickly apparent that some internet sources disagree with others or are misleading.

I've highlighted keywords from the following 6 sources so you can form your own opinion (13/08/2021):

- https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
- https://www.tutorialspoint.com/mvvm/mvvm_introduction.htm
- https://www.c-sharpcorner.com/article/mvvm-architecture/
- https://intellitect.com/getting-started-model-view-viewmodel-mvvm-pattern-using-windows-presentation-framework-wpf/
- https://www.learnmvvm.com/
- https://riptutorial.com/mvvm

After producing this, I would say the misleading part is where the business logic stays. TutorialsPoint commands the model shouldn't contain business logic while other sources differ. My favourite definitions are from Wikipeida, IntelliTect and RIPTutorial.

Contents

Model

Wikipedia

"Model refers either to a **domain model**, which represents real state content (an object-oriented approach), or to the **data access layer**, which represents content (a data-centric approach)."

"A **domain model** is a **conceptual model** of the domain that incorporates both behaviour and data."

"A **conceptual model**, or domain model, represents concepts (entities) and relationships between them."

"A **data access layer** (DAL) in computer software is a layer of a computer program which provides simplified access to data stored in persistent storage of some kind, such as an entity-relational database."

TutorialsPoint

"It simply **holds the data** and has **nothing** to do with any of the **business logic**."

After researching this a little more, they could be talking about putting business logic in a separate service which is perfectly reasonable. "The service will implement all business operations you will want to perform by working with model instances as appropriate."

https://stackoverflow.com/questions/16338536/mvvm-viewmodel-and-business-logic-connection

C#Corner

"The model represents the **domain model** also considered as **Business Logic** / **Data Access Logic** or we can abstractly define it as the **backend** of the application. They hold application data."

IntelliTect

"Broadly speaking the model provides **access** to the **data and services** that your application needs. Depending on your application, this is where the **real work gets done**. While the view model is concerned with pulling together the model's data, the model classes perform **the actual work of the application.** If you are using dependency injection, the model classes will typically be passed as interface constructor parameters in your view model."

LearnMVVM

Responsibility: “To model a business object containing **the required data**.”

Design Tips: “As the model is only a **representation of the data of a business entity** (object) it should contain only properties containing object’s data and constructors. Implementing INotifyPropertyChanged is right as we consider this part of the object’s responsibility.”

RIPTutorial

“Model is the layer that **drives the business logic**. It retrieves and stores information from any **data source** for consumption by the ViewModel.”

View

Wikipedia

“As in the model–view–controller (MVC) and model–view–presenter (MVP) patterns, the view is the **structure, layout, and appearance** of what a user sees on the screen. It displays a **representation of the model** and **receives the user's interaction** with the view (mouse clicks, keyboard input, screen tap gestures, etc.), and it **forwards** the handling of these **to the view model** via the data binding (properties, event callbacks, etc.) that is defined to link the view and view model.”

TutorialsPoint

“It simply **holds the formatted data** and essentially **delegates** everything to the **Model**.”

C#Corner

“View Represents the **UI** of the application. This is what the user **interacts** with. It is the presentation part.”

IntelliTect

“These are all of the **UI elements**, the pretty face of your application. For WPF, these are all of your XAML files. They may be Windows, User Controls, or Resource Dictionaries. Though it is certainly possible to construct the view entirely from code, the vast majority of the UI will (and should be) built with XAML. The view may be quite dynamic, and even **handle some user interaction** (see the Commands section below).”

LearnMVVM

Responsibility: “To **present the data** available at the DataContext to the end user to allow him to **interact** with it.”

Design Tips: “**Avoid** putting **logic** on the View even when XAML allows to do it. Prefer moving that logic to the ViewModel over adding it to the View because:

- XAML logic cannot be debugged.
- XAML logic cannot be tested.”

“View is the layer that represents the **interface** of the software (i.e. the GUI). Its role is to display the **information from** the **ViewModel** to the user, and to **communicate the changes** of the information back to the ViewModel.”

View model

Wikipedia

"The view model is an **abstraction of the view** exposing public properties and commands. Instead of the controller of the MVC pattern, or the presenter of the MVP pattern, MVVM has a binder, which **automates communication** between the view and its bound properties in the view model. The view model has been described as a **state of the data** in the model. The main difference between the view model and the Presenter in the MVP pattern is that the presenter has a reference to a view, whereas the view model does not. Instead, a view directly binds to properties on the view model to send and receive updates. To function efficiently, this **requires** a **binding technology** or generating **boilerplate code** to do the binding."

TutorialsPoint

"It acts as the **link/connection** between the Model and View and makes stuff look pretty."

C#Corner

"It is the **logic of View**. The ViewModel is also called as presentation logic. The View and ViewModel **communicate** with each other. The Request from ViewModel is forwarded to Model / Business Logic Layer / Data Access Layer. It allows sharing of computed/resultant data to the view."

IntelliTect

"These are the objects that **provides the data and functionality** for each of your **views**. In general there is typically a **one-to-one mapping** between views and view model classes. The view model class, exposes the data to the view, and **provides commands** to **handle user interaction**. Unlike other design patterns, the view model should not know about its view. This separation of concerns, is one the key tenets of MVVM. The view model is the connection between the view and model."

LearnMVVM

Responsibility: "To **contain the logic** that acts as a **bridge** between the View and the Model."

Design Tips: "Avoid putting too much logic inside the ViewModel and consider creating more classes (services, engines, etc.) if the ViewModel logic is too big to fit in one class. Include always a reference to the model on the ViewModel."

“ViewModel is the layer that acts as a **bridge** between the View and the Model. It may or may not transform the **raw data** from the Model into a **presentable form** for the View. An example transformation would be: a boolean flag from the model to string of 'True' or 'False' for the view.”

Binder (Extra)

Wikipedia

Declarative data and command-binding are implicit in the MVVM pattern. In the Microsoft solution stack, the **binder is** a markup language called **XAML**. The binder **frees** the developer from being obliged to write **boiler-plate logic** to synchronize the view model and view. When implemented outside of the Microsoft stack, the presence of a declarative data binding technology is what makes this pattern possible, and **without** a binder, one would typically use **MVP** or **MVC** instead and have to write more boilerplate (or generate it with some other tool).