Model-View-ViewModel

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

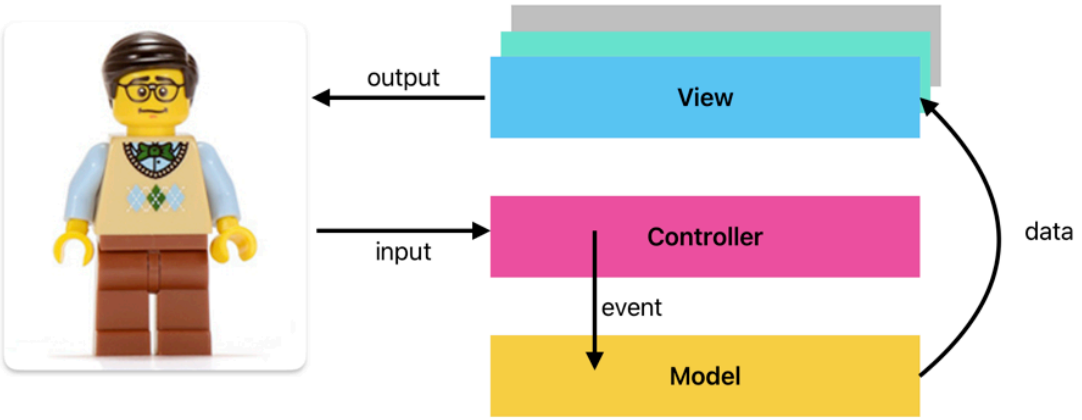
Contents

MVC Patterns

Model-View-Controller

It's impossible to talk about building interactive systems without acknowledging the Model-View-Controller (MVC) pattern. `MVC` originated with Smalltalk (1988), as a pattern for building interactive applications. It divides your program logic across three classes.

- The `controller` accepts and handles input, sending updates to the model.
- The `model` handles all application state (single "source of truth").
- The model published changes to the `view`.



.

Model-View-Controller

Observer Design Pattern

The Observer pattern is a lightweight notification mechanism between a Publisher & Subscriber. It's commonly used when implementing MVC [1].

- Subscriber/Publisher are interfaces.
- The `View` (Subscriber) registers with the `Model` (Publisher) when the application launches.
- The `Controller` accepts user input and updates the `Model`.
- The `Model` (Publisher) notifies all Subscribers of the state change.
- Each Subscriber decides how to act on the information, often by fetching updated data from the Publisher.

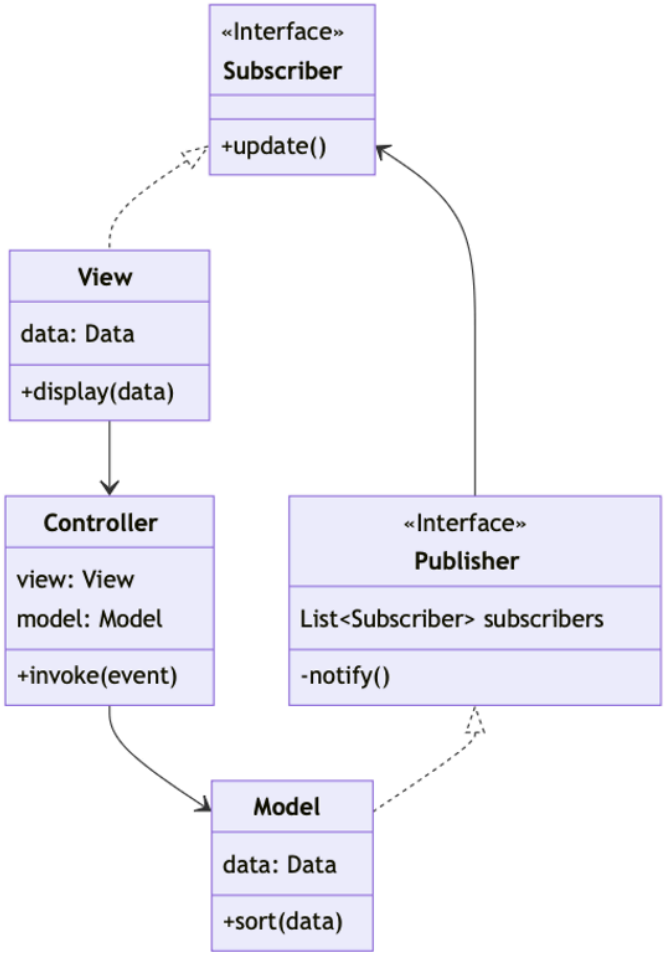
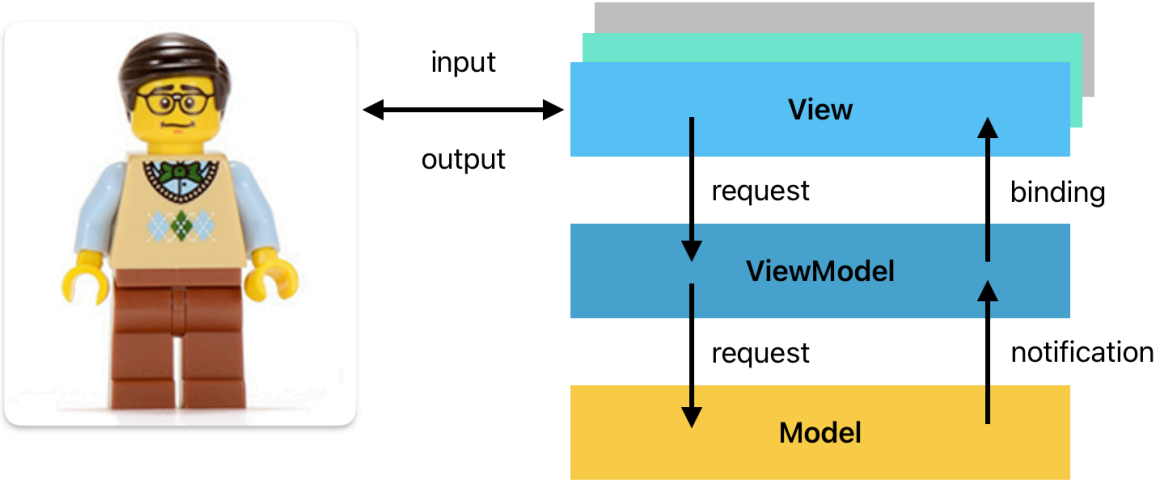


Figure 1: Subscribers register themselves with a Publisher, which agrees to send them updates when data changes.

Model-View-ViewModel

Model-View-ViewModel (MVVM) was invented by Ken Cooper and Ted Peters in 2005. It was intended to simplify event-driven programming and user interfaces in C-sharp and .NET.

`MVVM` adds a `ViewModel (VM)` that sits between the `View` and `Model`. The VM manages all data requests to-and-from the views.



.

Model-View-ViewModel

Why is a ViewModel useful?

We often want to pull “raw” data from the `Model` and modify it before displaying it in a View

- e.g., currency stored in USD but displayed in a different format.

We sometimes want to make local changes to data but not push them automatically to the global `Model`

- e.g., undo-redo where you don’t persist the changes until the user clicks a Save button.

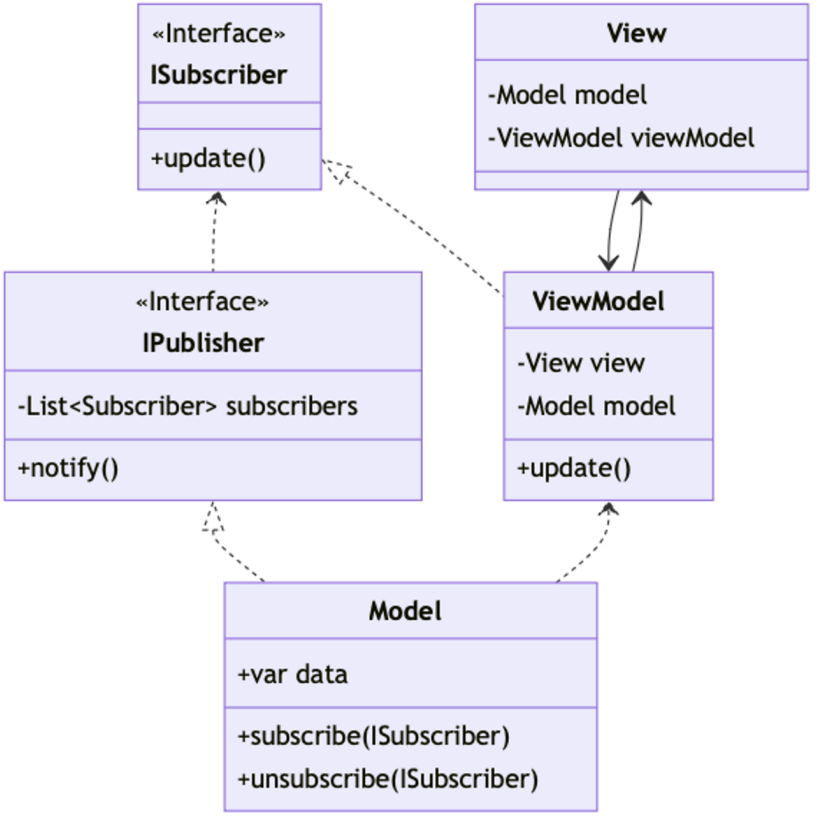


Figure 2: With MVVM, the ViewModel is the Subscriber, and coordinates updating the Views.

Other Variants

There are other variants of MVC.

Model-View-Presenter is similar to MVVM, with View and Model classes, but usually no Controller. MVP adds a mediating class (`Presenter`) sitting between the Model and the View, whose job is to present data to the View. It's structurally very similar to MVVM, including the use of Publisher-Subscriber.

Model-View-Adapter also has a View and a Model, but no controller. The `Adapter` is a mediating class between these two. The main difference between MVA and MVP is that MVA does not allow for any direct Model-View communication (i.e., no `Publisher-Subscriber`).

> **Tip**
>
> The differences are minor and mostly historical. The key principle for each of these is to maintain a clear separation between Presentation and Data.

Bibliography

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison-Wesley Professional, 1994.