Model-View-ViewModel Architecture (MVVM)

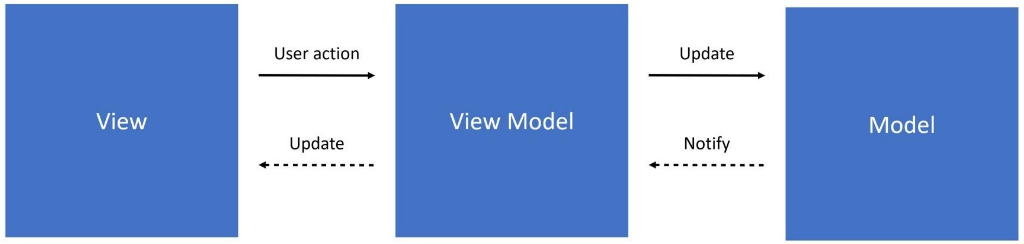
CMSC 10500

Overview

MVVM is a way of organizing applications with user interfaces. It is not specific to iOS applications.

It is used with SwiftUI and is new to Swift development (before iOS applications used the model-view-controller (MVC) architecture)

An intermediate global object called a *view model* separates the *model* (the underlying code) and the *view* (what is presented to the user on the screen).



The Problem

The user interface and the code which is run by your application must always consistent with one another.

Changes in the state of the code should be reflected in the UI, and the user should be able to change the state of the code by interacting with the UI.

Trying to manage the interaction between these two parts of our application would be **complicated** if we try to do it by hand.

MVVM simplifies these interactions.

Terms

Model. UI independent code for the primary functionality of the application. For project 2, this will be an implementation of the game Tic Tac Toe.

View. Code for the UI which appears on a screen. In Swift, it is also the name of UI Elements (more specifically, it is the name of a *protocol* which is implemented by things that can appear on screen.

View Model. Code that tells the view when the model changes and tells the model with the view wants it to change.

The Model

Typically implemented as a structure.

For project 2, you will build out a `struct` called `TicTacToeGame` with methods for adding pieces to the grid during game play.

The model is **UI independent**. You should be able to use the same model whether you are building a command-line program or an iOS application.

Basically the kind of coding we have been doing in this course.

The View

For single-page applications, there is a file called `ContentView.swift` by default. It contains a structure `ContentView`, which conforms to the `View` protocol and describes what should appear on screen.

Building Views in Swift is a form of *declarative programming*. We do not write a procedure for constructing the view, we describe or declare the view.

In particular, **views have no sense of time**. We do not implement how the view changes over the course of running the application.

The view **immediately reacts to changes in the model** (as broadcasted by the view model).

The View Model

Typically implemented as a `class` **(basically a reference type version of a structure)**, it contains an *instance* of the model.

This is a very important use of reference types. We want all parts of the view to refer to the **same** view model, not copies.

It broadcasts changes of the model to the view, and provides limited functionality to the view to change the model.

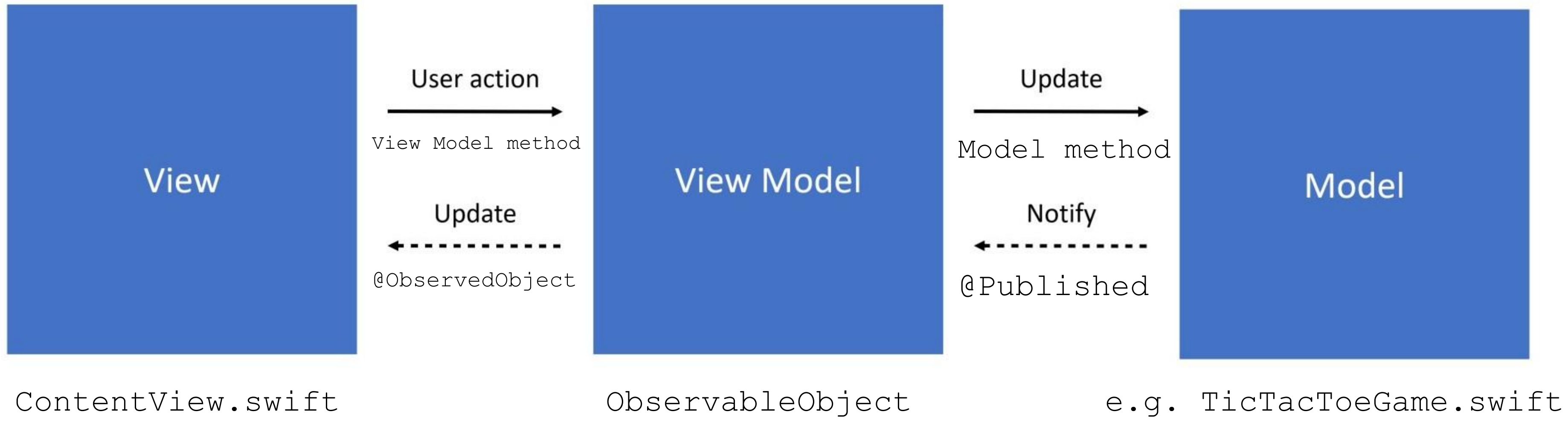
We will have to create our own file for the view model, it is not pre-built.

View Models in SwiftUI

Creating a view model with SwiftUI is very simple (at least the easy we are going to do it).

1. Create a file with a single class for the view model that implements the `ObservableObject` protocol. This make it observable to the view. We don’t have to do anything to conform to this protocol.
2. Create a property in the class for an *instance* of the model and label it with the decoration `@Published`. This tells the view what the model is, which it will react to.
3. Add a view model instance to the view with the decoration `@ObservedObject`. The view will then react to changes in its published properties (like the model) and use it to make changes to model.

MVVM with SwiftUI


View
User action
View Model method
Update
@ObservedObject
View Model
Update
Model method
Notify
@Published
Model
ContentView.swift
ObservableObject
e.g. TicTacToeGame.swift