MVVM + Conceptos de Swift

PMDM - iOS Programación multimedia y dispositivos móviles

Ariel Hernández, Febrero 2021

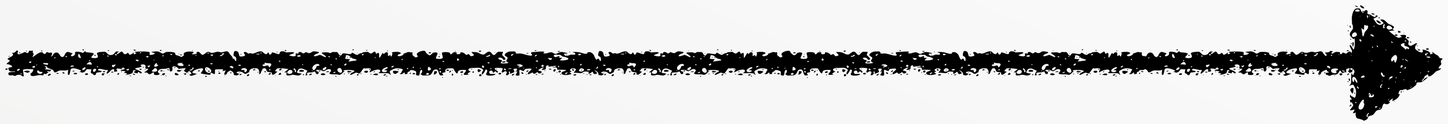
Qué es MVVM

- Un **Paradigma de Diseño** de **Arquitectura de Código**
- Ideal para UX "reactivas"
- Ideal para SwiftUI
- Distinta de MVC.

MVVM

Model

Información sólo del *Model* a la *View*

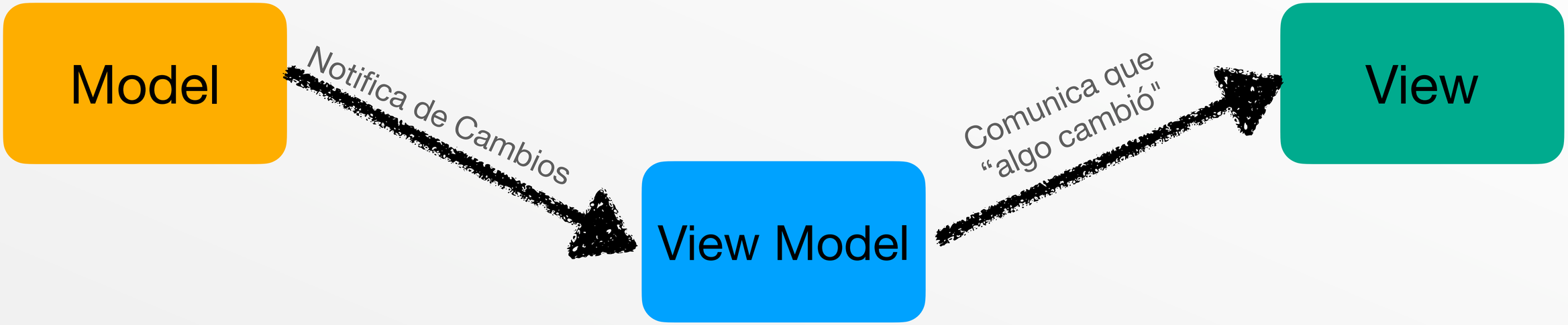


View

- Independiente de la UX
- Encapsula los Datos y la Lógica (el estado) de nuestra app.
- En la demo:
  - Cards
  - Logic

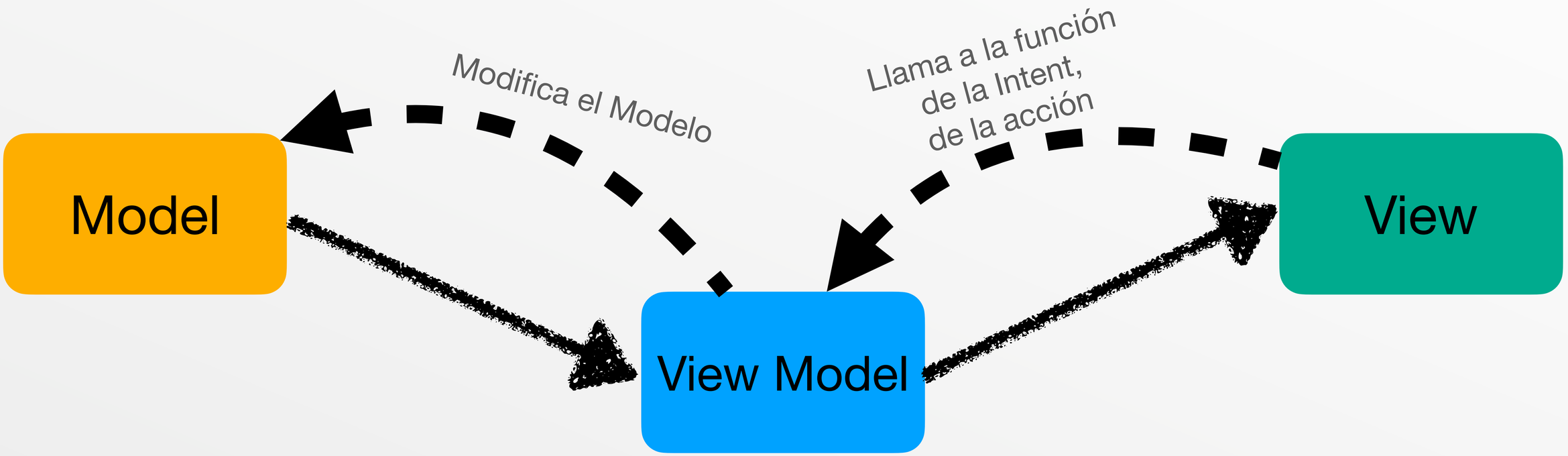
- Muestra la información contenida en el Modelo
- No tiene estados y siempre puede ser re-creada desde un Modelo
- Declarativa != Imperativa
- Permite aislar todo el código de UX en un mismo sitio.
- Reactivo

MVVM: View Model



- Es la conexión entre el *Model* y la *View*
- Se encarga de que los cambios del *Model* sean comunicados a la *View*, para que “reaccione"
- Convierte posibles valores “en bruto” del *Model* a datos para ser mostrados en la *View* (ej: Convertir `Date` en `String`)

MVVM


Modifica el Modelo
Llama a la función
de la Intent,
de la acción
Model
View Model
View

MVVM: Reglas de Oro

- El *Model* no sabe **NADA** fuera de él mismo y otros modelos con los que interactúe.
- El *View Model* tiene conocimiento del *Model* pero **NUNCA** lo expone para que lo use la *View, s*ólo expone datos de su estado.
- El *View Model* no sabe **NADA** de la *View*.
- La *View* no sabe **NADA** del *Model.*

Swift: Conceptos

- `struct`
- `class`
- *Genericidad*
- *Funciones*

Swift: `Struct` vs `Class`

`struct == class ??`

(Semejanzas)

- Similar definición

```
struct CardViewS {}
class CardViewY {}
```

- Stored `var`s (propiedades)

```
var body = true
```

- Computed `var`s (propiedades)

```
var body: Bool { true }
```

- Constantes

```
let body = true
```

- Funciones

```
func sum(first: Int, second: Int) -> Int{
    return first + second
}
sum(first:3, second:4)

func sum(_ first: Int, plus second: Int) -> Int{
    return first + second
}
sum(4, plus: 5)
```

- Inicializadores

```
init(numberOfCards:Int){}
```

Swift: `Struct` vs `Class`

`struct ≠ class ??`
(Differences)

- Value Type
  - Copy at Write
- Functional Programming (define *Funcionalidad*)
- No Herencia
- El `init` “gratis" inicializa **todas** sus `var`
- Si se usa en un let, es inmutable
- Ideal para Views y Models
- SwiftUI es Struct Based

- Reference Type
  - Automatic Reference Counter
- Object Oriented Programming (define *Funcionalidad* y encapsula *Datos*)
- Herencia (Simple)
- El `init` “gratis" **no** inicializa sus `var`
- Siempre es Mutable
- Ideal ViewModels
- UIKit es Class Based

Swift: *Genericidad*

- A veces el *type* de una variable no nos interesa demasiado, nos basta con que funcione de una manera.
  - No nos importa si es Text o CardView
  - Swift es **MUY** fuertemente tipado, y **TODO** tiene que tener un tipo.
- El `Array` nos interesa que contenga elementos, pero puede ser `Int` o `String`
  - Pero dentro el Array necesita saber que contiene, cómo se define?

```
struct Array<Element> {
    func append(_ element:Element){}
}
var letters = Array<String>()
letters.append(“A")
letters.append(“B”)
```

Swift: Funciones

- Las Funciones son siempre un tipo.
  - Las puedes usar donde sea como si fuera un tipo.
  - Se definen así:

```
(Int, Int) → Int
(Double) → Void
() → String
() → Void
```

- Como son tipos, pueden declarase variables con ellas.

```
func sum(_ first: Int, plus second: Int) → Int{
    return first + second
}
var operation: (Int, Int) → Int
operation = sum
operation(4,5) // Pierde las etiquetas
```

Demo

- MVVM
- `init`
- funciones
- `class` & `struct`
- UX Reactiva