Entwurfsmuster in der iOS Entwicklung

Kyrill Schmid, Markus Friedrich WiSe 17/18

Einführung

Was sind Entwurfsmuster?

- Lösungsschablonen für wiederkehrende Entwurfsprobleme (Wikipedia)

- Wird meistens mit Codestrukturschablonen gleichgesetzt.
  - Aber: Muster existieren auch für andere Bereiche außerhalb der Softwareentwicklung.
  - Beispiel: HCI, aber auch fachfremd: Architektur, Maschinenbau

- Fokus: Entwurfsmuster in der Software Entwicklung
  - Wichtig: Die Beschreibung (der meisten) Muster ist **unabhängig** von einer bestimmten Programmiersprache.

- Wir unterscheiden für unsere Zwecke:
  - **Architekturmuster**: Beschreiben die Grobarchitektur der gesamten Anwendung
  - (klassische) **Entwurfsmuster**: Beschreiben einzelne Teilelemente der Architektur in größerem Detail

Entwurfsmuster - Kategorien

Man unterscheidet verschiedene Kategorien von (OOP) Entwurfsmustern:

- **Erzeugungsmuster:** Trennung der (komplexen) Erzeugung von Objekten von Ihrer Repräsentation
- **Strukturmuster:** Schablonen zur Modellierung von Beziehungen zwischen Objekten
- **Verhaltensmuster:** Muster zur Modellierung des Verhaltens von Objekten

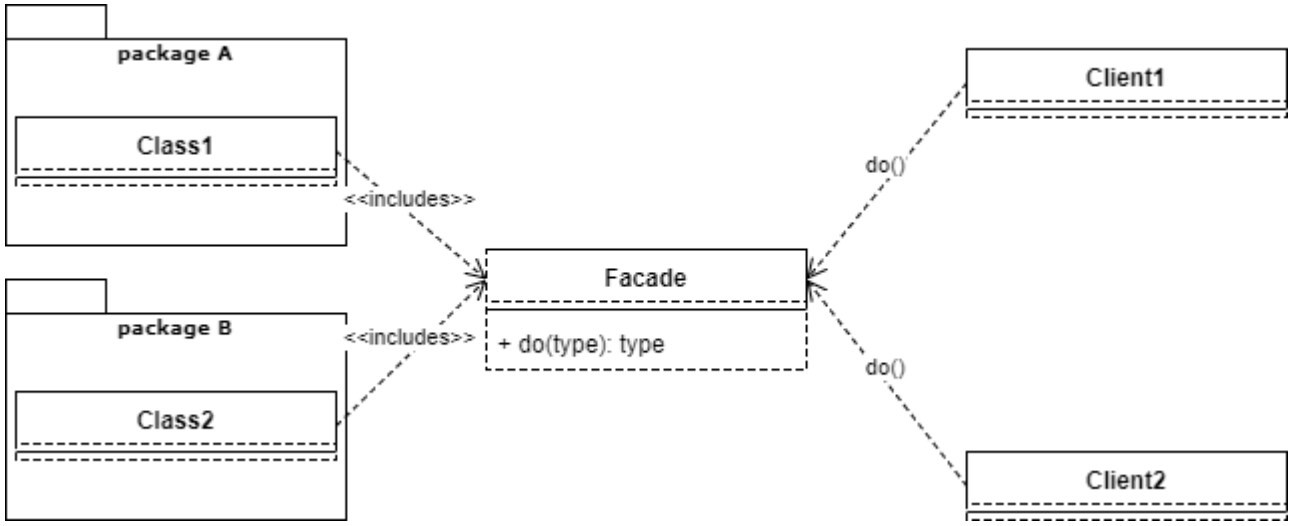
Diese Unterscheidung hat ihren Ursprung im Buch „Design Patterns – Elements of Reusable Object-Oriented Software“ der „Gang if Four“ (GOF), 1995

Neue Kategorien wurden im Laufe der Zeit hinzugefügt.

Beispiele

Strukturmuster „Fassade“

Ziel: Vereinfachte Schnittstelle zu einem größeren (und komplizierteren) System.



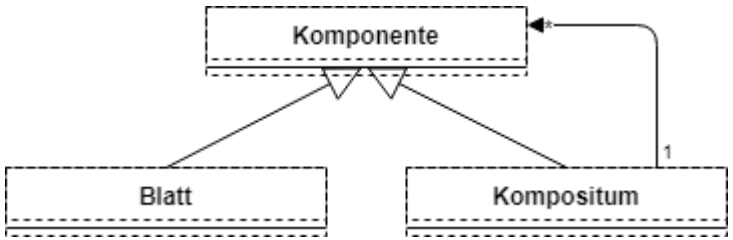
In iOS / Swift:

- NSImage
- …

Beispiele

Strukturmuster „Kompositum“

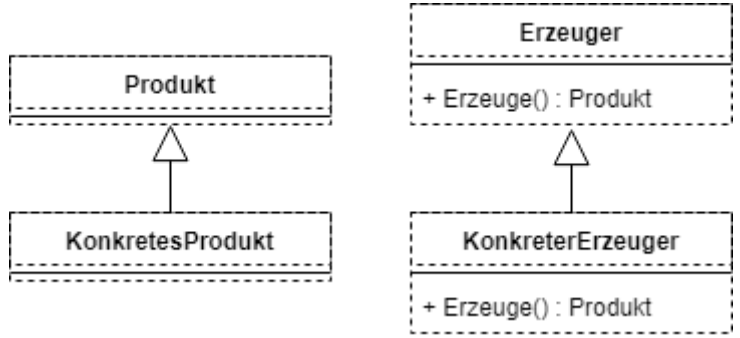
Ziel: Einheitliche Behandlung von Objekten und Kompositionen von Objekten.



In iOS / Swift:

- Views
- …

Beispiele



Erzeugungsmuster „Fabrikmethode“ mit Registry => Ziel: Entkopplung der Erzeugung konkreter Klassen

```
protocol Fruit {
  func grow()
}
class Cherry : Fruit {
  func grow() {
    print("Cherry grows...")
  }
}
class Banana : Fruit {
  func grow() {
    print("Banana grows...")
  }
}
```

```
class FruitFactory {
  func register(type : String, creator : @escaping () -> Fruit){
    creators[type] = creator
  }
  func create(type : String) -> Fruit {
    return creators[type]!()
  }
  var creators : [String :  () -> Fruit] = [:]
}
```

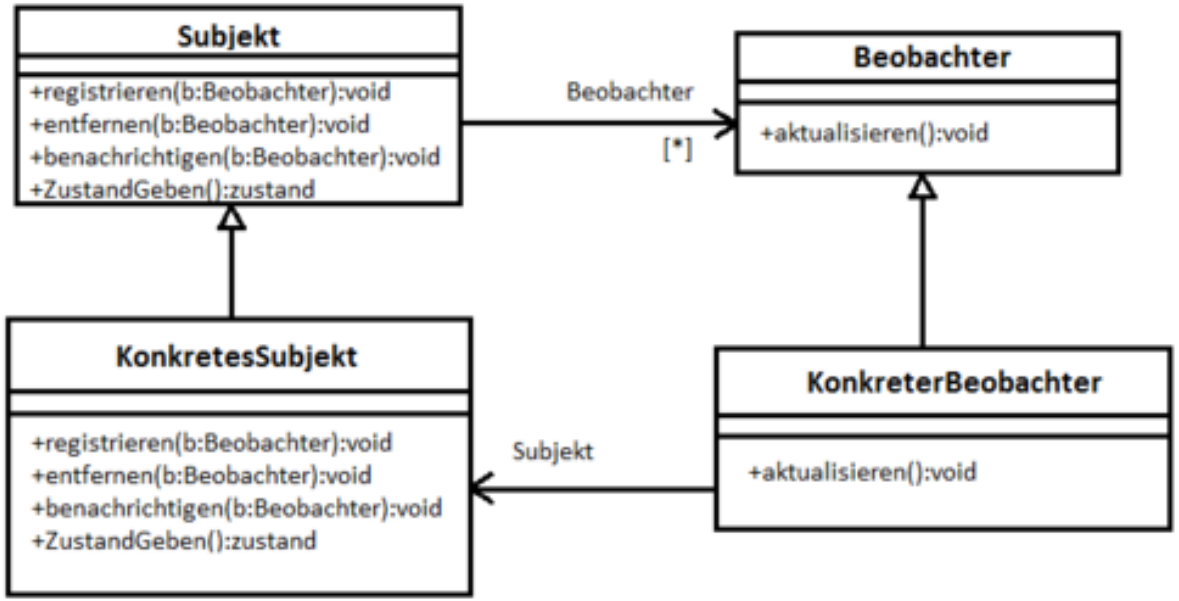
```
var factory = FruitFactory()
factory.register(type:"Cherry", creator:{ return Cherry() })
factory.register(type:"Banana", creator:{ return Banana() })
factory.create(type:"Cherry").grow()
factory.create(type:"Banana").grow()
```

Eine Variante mit `NSClassFromString` ist komfortabler aber nicht empfohlen in Verbindung mit Swift

Beispiele

Verhaltensmuster „Beobachter“ (Observer Pattern)

Ziel: Benachrichtigungen zwischen Objekten mit loser Kopplung



Quelle: https://de.wikipedia.org/wiki/Datei:Beobachterentwurfsmuster.png

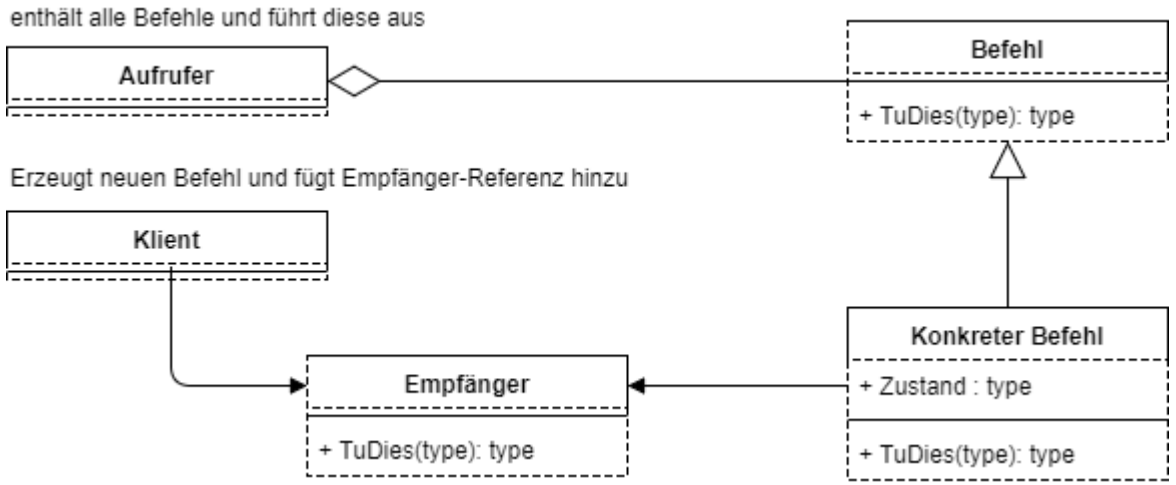
In iOS / Swift:

- Notification Center
- Key-Value-Observing (KVO)
- [Delegate]

Beispiele

Verhaltensmuster „Kommando“

Ziel: Kapselung von Methodenaufrufen in Objekte => Warteschlangen



In iOS / Swift:

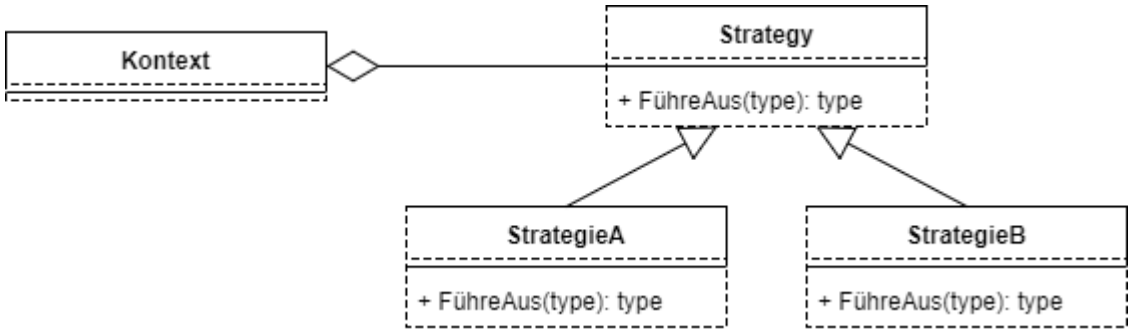
- Views: Target-Action Mechanismus

Beispiele

Verhaltensmuster „Strategie“

Ziel: Beschreibung einer Familie von Algorithmen

Interessant: Verbindung mit Fabrikmuster



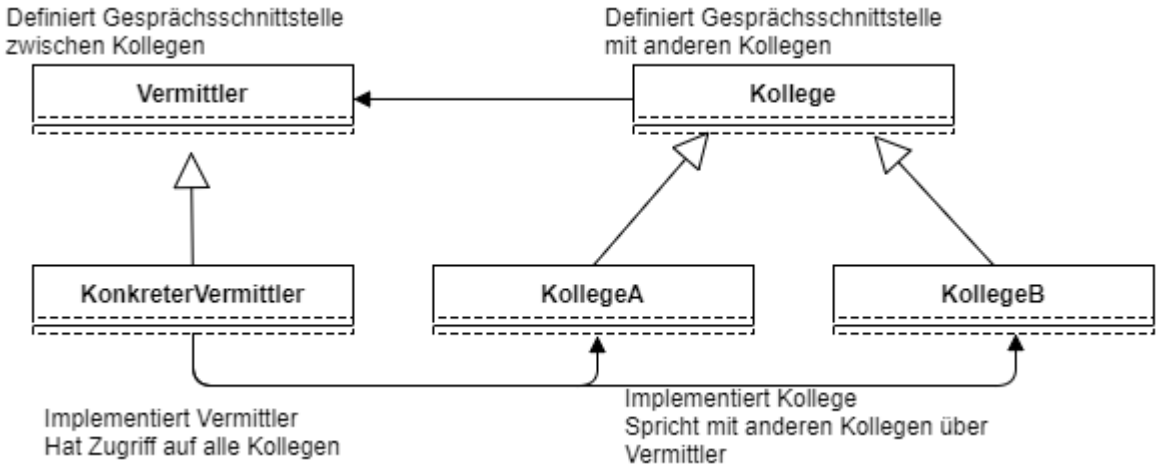
In iOS / Swift:

- View Controller

Beispiele

Verhaltensmuster „Vermittler“

Ziel: Steuerung der Kooperation verschiedener Objekte



In iOS / Swift:

- View Controller

Beobachter – Notification Center

- Mechanismus zur Verteilung von Ereignissen an registrierte Beobachter.
- Wie funktioniert das?

```
// Get application-wide NotificationCenter Singleton.
let nc = NotificationCenter.default

// Register Observer
nc.addObserver(forName:Notification.Name(rawValue:"MyNotification"),
  object:nil, queue:nil){
  notification in
// Handle notification
}

// Notify
nc.post(name:Notification.Name(rawValue:"MyNotification"),
 object: nil,
 userInfo: ["message":"Hello there!", "date":Date()])
```

Quelle: http://dev.iachieved.it/iachievedit/notifications-and-userinfo-with-swift-3-0/

Beobachter – Key-Value-Observing (KVO)

- Beobachtung einer Eigenschaft (Property) eines Objekts durch ein anderes Objekt
- Gute Option, um zwei beliebige Objekte synchronisiert zu halten
- Wichtig: Klasse muss von NSObject erben

```
class MyObjectToObserve: NSObject {
    @objc dynamic var myDate = NSDate()
    func updateDate() {
        myDate = NSDate()
    }
}
```

```
class MyObserver: NSObject {
    @objc var objectToObserve: MyObjectToObserve
    var observation: NSKeyValueObservation?

    init(object: MyObjectToObserve) {
        objectToObserve = object
        super.init()

        observation = observe(\.objectToObserve.myDate) {
                  object, change in
                  print("Observed a change to
         \(object.objectToObserve).myDate, updated to:
         \(object.objectToObserve.myDate)")
        }
    }
}

let observed = MyObjectToObserve()
let observer = MyObserver(object: observed)
observed.updateDate()
```

Keys & Key Paths
Identifikation von Attributen und Relationen

Quelle: https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/AdoptingCocoaDesignPatterns.html

Vergleich

Delegate	Notification Center	KVO
+ Strikte Definition, Compiler hilft	+ Einfachere Nutzung	+ Zeigt neuen und alten Eigenschaftswert
+ Kontrollfluss gut nachvollziehbar	+ 1:n Observer einfach möglich	+ Verschachtelte Properties einfach beobachtbar (KeyPaths)
+ Kein Drittobjekt als Monitor nötig	+ Notifikation kann flexiblen Kontext enthalten (userInfo	+ Kein extra Code notwendig
- Delegate muss auf nil gesetzt werden, wenn Objekt gelöscht wird	- Abmeldung nötig, wenn Objekt gelöscht wurde	- Beobachter muss entfernt werden, wenn Objekt gelöscht wurde
- 1:n Observer schwer zu implementieren	- Keine Checks zur Kompilierzeit	- Keine Checks zur Kompilierzeit
- Viel „Boilerplate Code“	- Kontrollfluss schwer nachvollziehbar	- Komplexe IF-Unterscheidung wenn mehrere Eigenschaften beobachtet werden sollen

Quelle: https://shinesolutions.com/2011/06/14/delegation-notification-and-observation/

Architekturmuster - Kategorien

Man unterscheidet verschiedene Kategorien von Architekturmustern:

- **Adaptive Systeme:** Muster für Systeme mit Fokus auf Erweiterbarkeit
- **Datengetriebene Systeme:** Muster für datengetriebene Applikationen
- **Verteilte Systeme:** Muster zur Orchestrierung und Kommunikation in verteilten Anwendungen
- **Interaktive Systeme:** Muster für die Strukturierung von HCI Anwendungen
- **…**

Beispiele

Muster adaptiver Systeme: **„Reflektion“**

- Eigenschaften und Methoden einer Klasse / eines Objekts zur Laufzeit inspizieren und ändern.
- In Swift: Nur lesender Zugriff
- Beispiel: Zugriff über die Mirror-Klasse:

```
struct Book {
  let title: String
  let author: String?
  let published: Date
  let numberOfPages: Int
  let chapterCount: Int?
  let genres: [String]
}
```

```
let bookMirror = Mirror(reflecting: book)

for (name, value) in bookMirror.children {
  guard let name = name else { continue }
  print("\(name): \(type(of: value)) = '\(value)'")
}
```

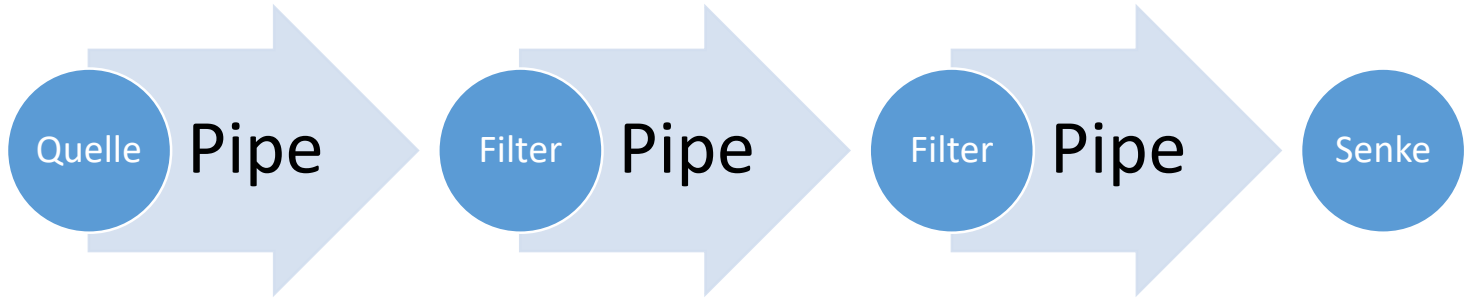
Quelle: https://makeitnew.io/reflection-in-swift-68a06ba0cf0e

title: String = 'Harry Potter'
author: Optional<String> = 'Optional("J.K. Rowling")'
published: Date = '2016-10-02 19:17:43 +0000'
numberOfPages: Int = '450'
chapterCount: Optional<Int> = 'Optional(19)'
genres: Array<String> = '["Fantasy", "Best books ever"]'

- Wann benutzen?
  - Generischer JSON/XML Serializer

Beispiele

Muster datengetriebener Systeme: „Pipes and Filter“

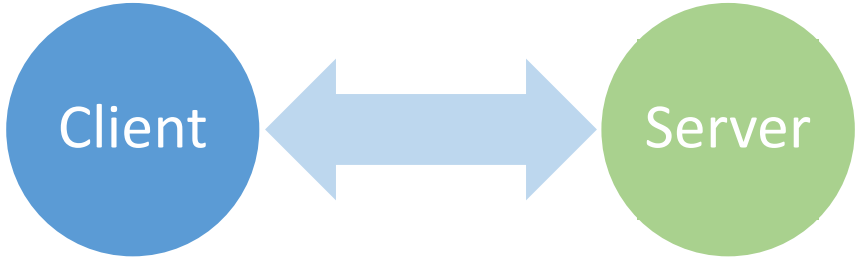


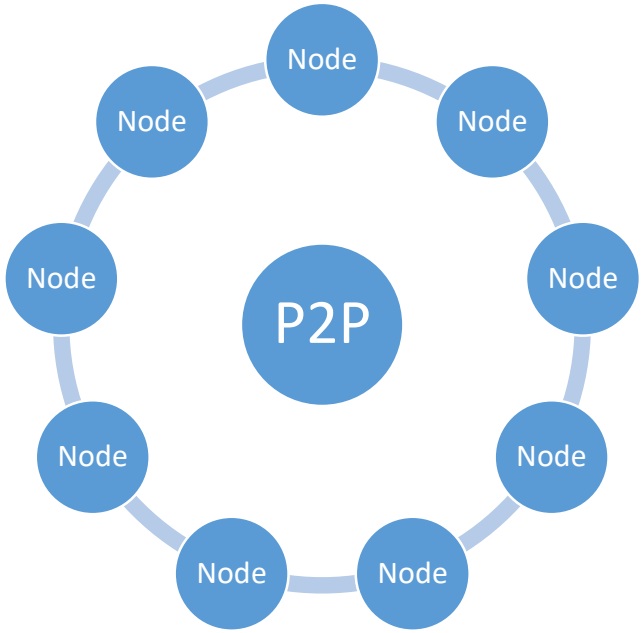
- **Quelle:** Ursprung des Datenflusses (z.B. Datei, Sensor, Socket)
- **Pipe:** Transport zwischen zwei Filtern oder Quelle-Filter, Filter-Senke (z.B. Queue, Socket)
- **Senke:** Ziel des Datenflusses (z.B. Datei)

=> Interessante Präsentation zum Thema Data Pipelines mit Swift:
https://de.slideshare.net/jarsen7/pipes-48485417

Beispiele

Muster verteilter Systeme: „Client-server“ vs „Peer-to-peer“

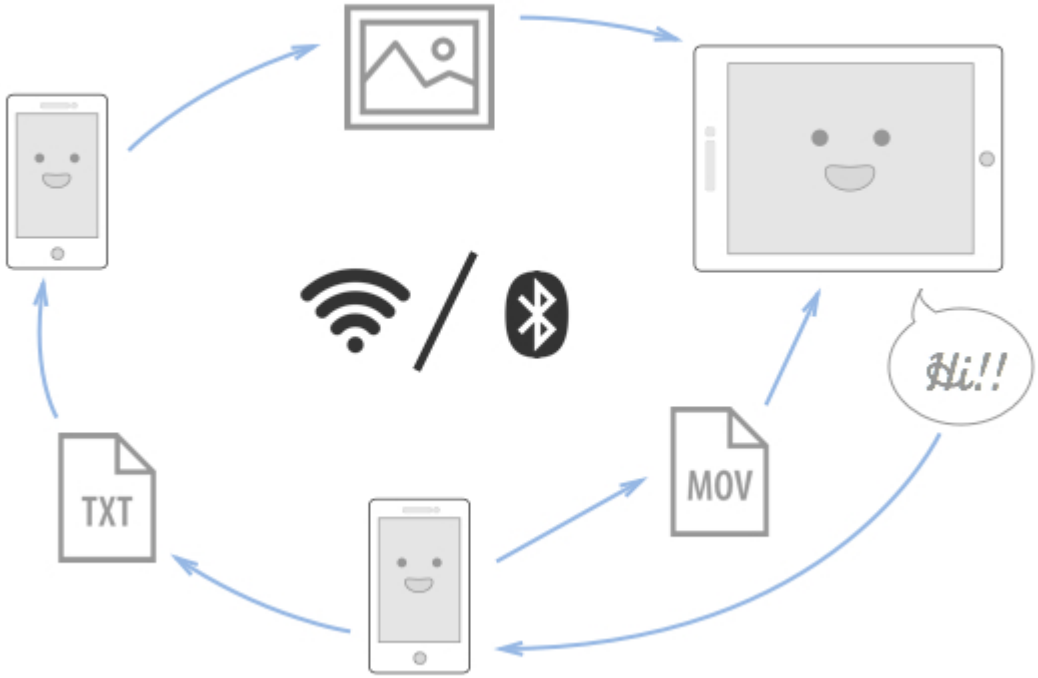




P2P in iOS

Multipeer Connectivity Framework (Seit iOS 7)

- Kommunikation mit Apps auf Geräten in der Nähe
- Schicht über dem Bonjour-Protokoll
- Wählt automatisch: Wi-Fi, P2P Wi-Fi, Bluetooth
- Details: https://developer.apple.com/documentation/multipeerconnectivity

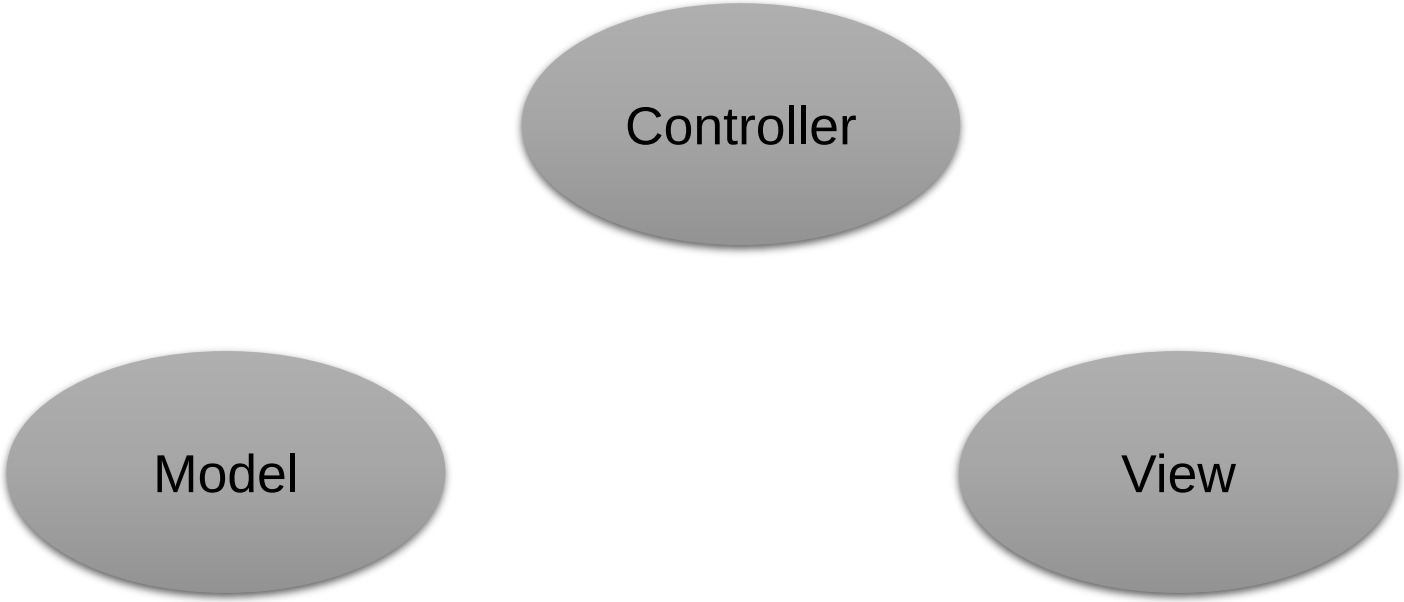


Quelle: https://leftshift.io/multipeer-connectivity-ibeacons-other-interesting-features-ios7

Beispiele

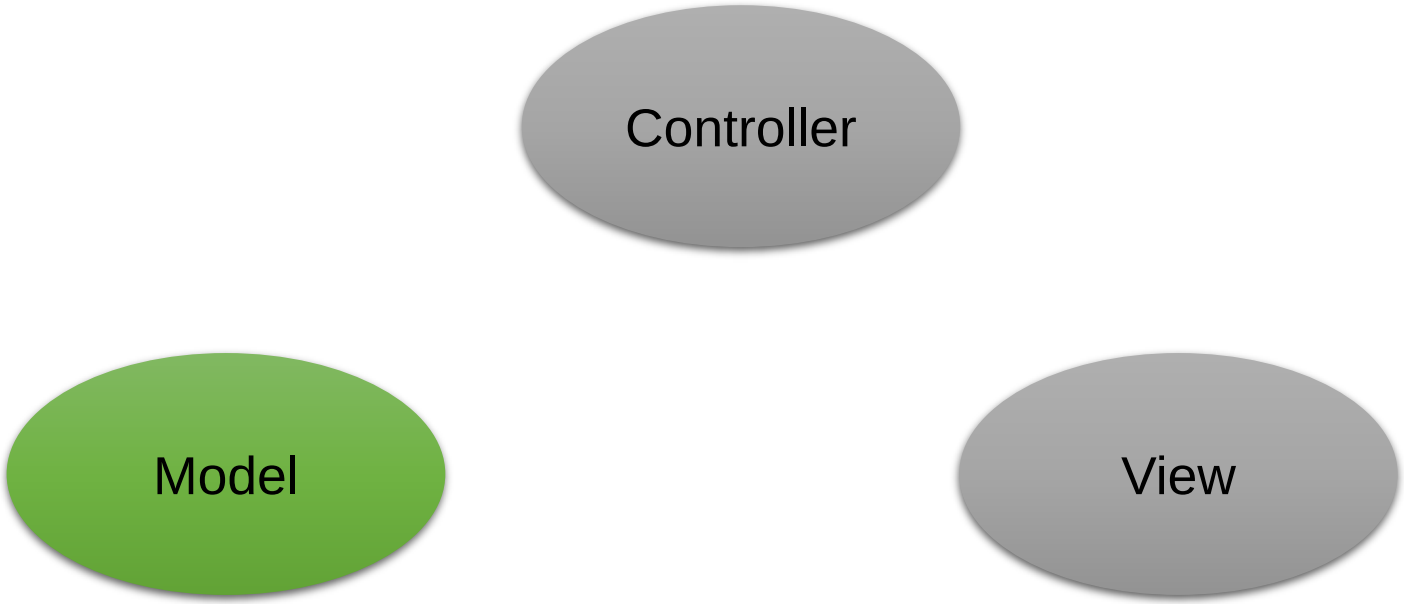
Muster interaktiver Systeme: „Model-View-Controller“ (MVC)

- Aufteilung von Objekten in drei Gruppen:



MVC - **Model**

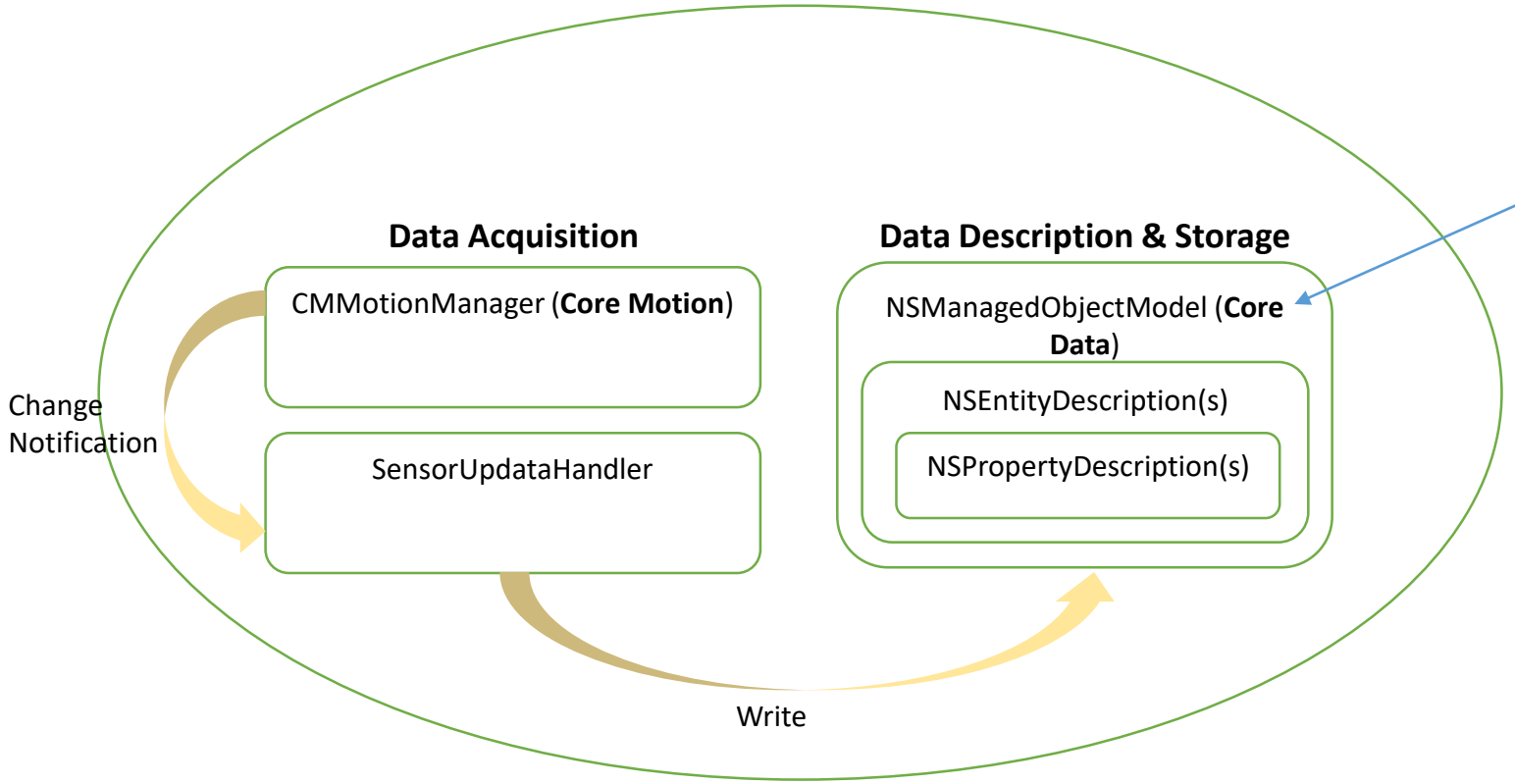
- Enthält **domänenspezifisches** Datenmodell.
- Wichtig: Datenmodell ist **unabhängig** von späterer UI-Darstellung!



MVC - **Model**

Models in iOS

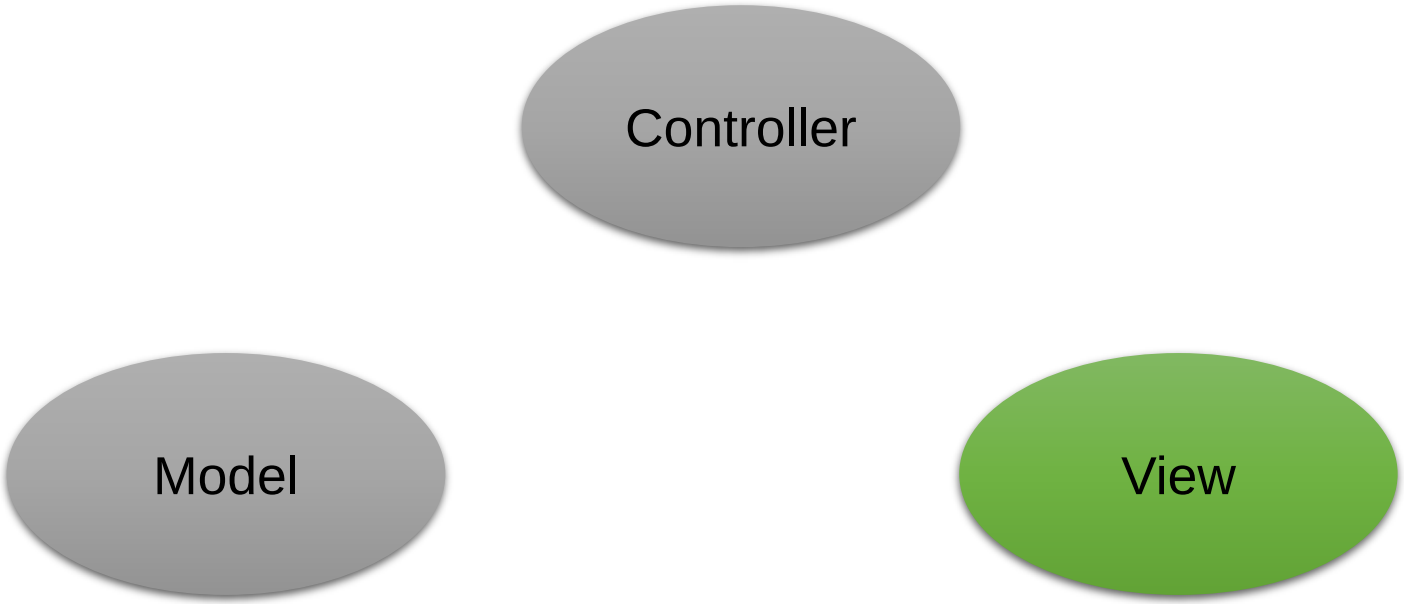
Persistenzalternativen:
- Firebase Cloud Storage
- SQLite eingebettete, lokale Datenbank



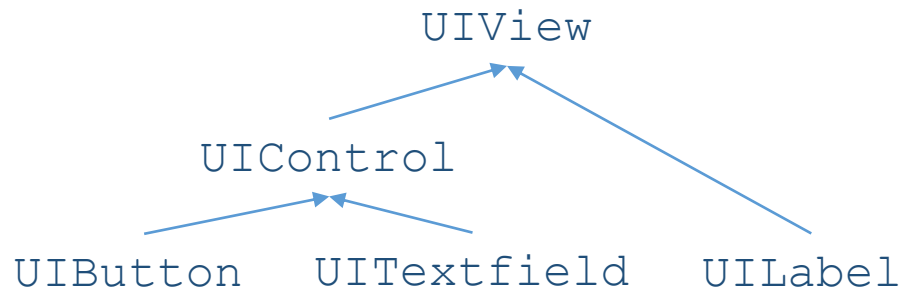
Verwaltet kompletten Objektlebenszyklus:
- Persistenz
- Undo/Redo
- Grouping/Filtering
- Komplexe Abfragen
- Versionierung
- View Controller Integration
- …

MVC - **View**

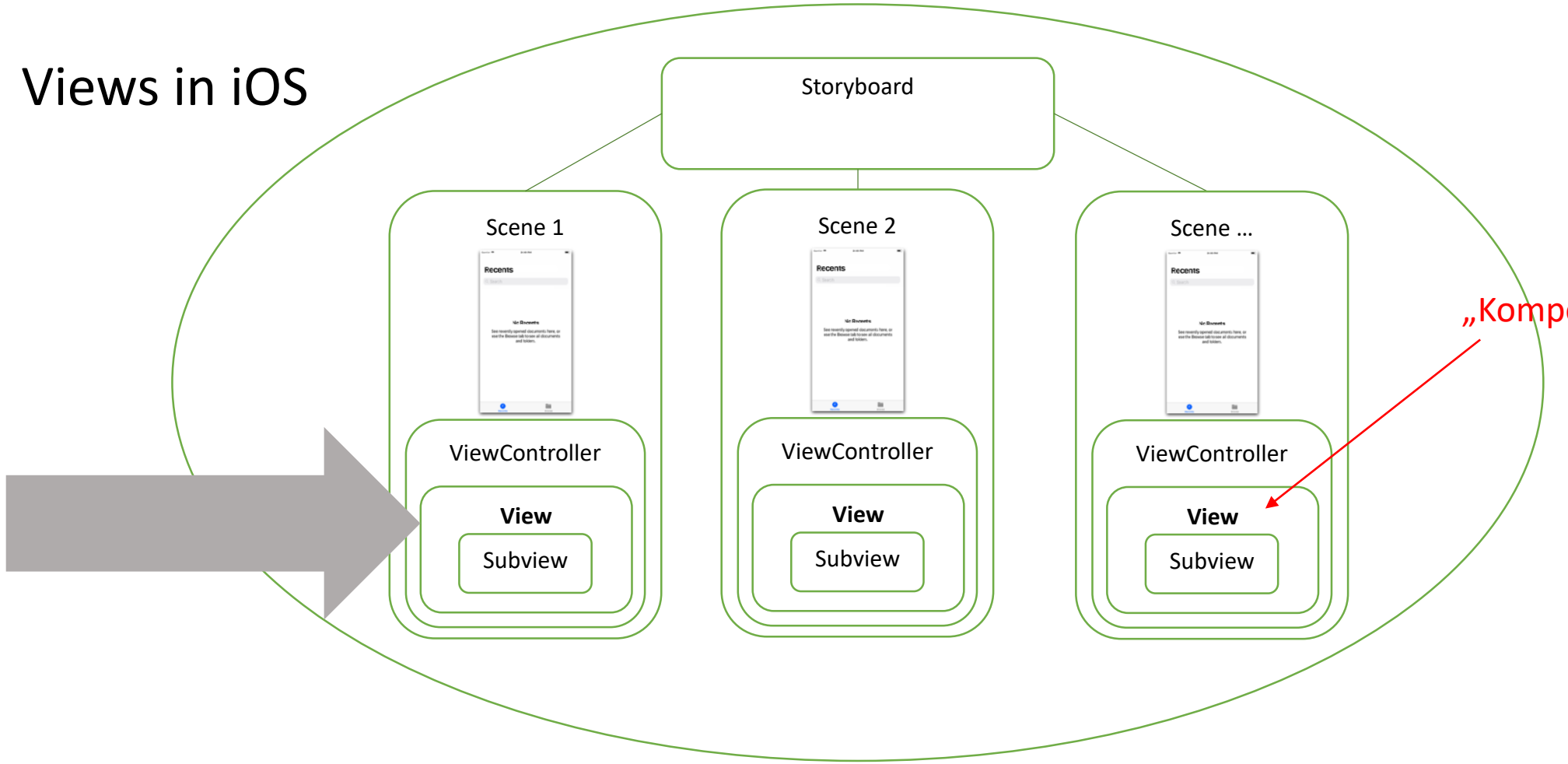
- UI-Darstellung der im Modell definierten Daten
- Benutzer interagiert mit den Daten, es findet aber **keine Verarbeitung** statt!



MVC - **View**

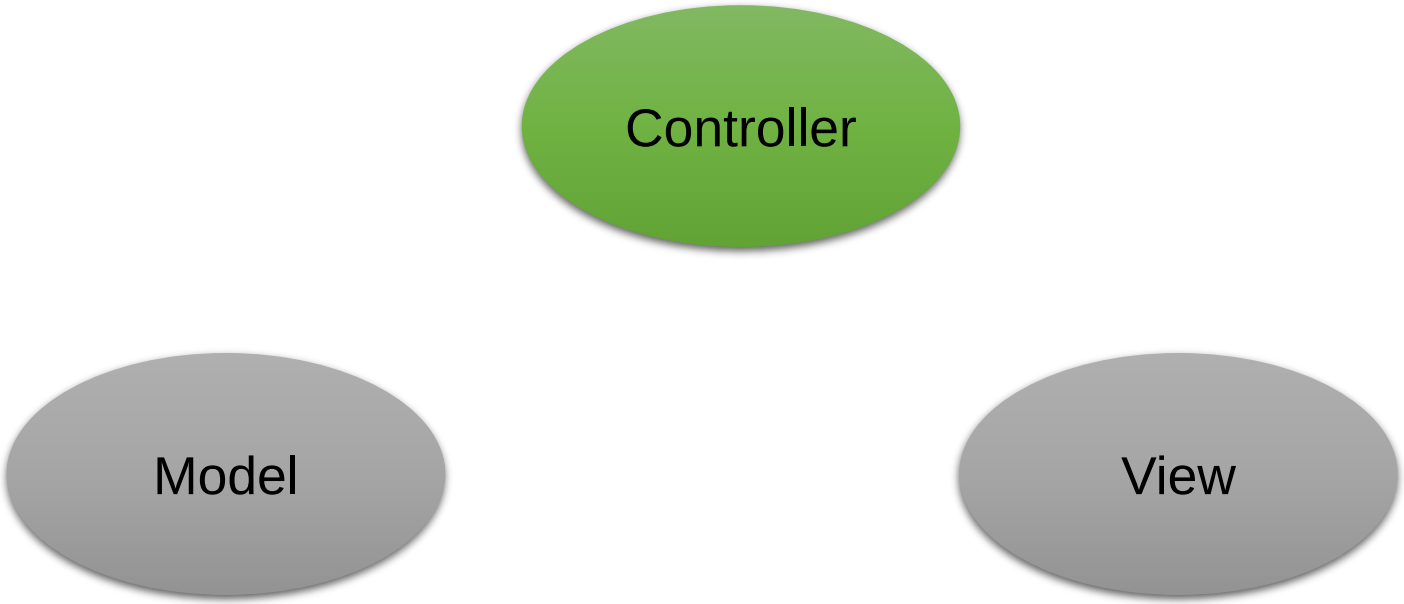


Views in iOS



MVC - **Controller**

- Vermittlung zwischen Datenmodell und Darstellung (Logik der Darstellung)
  - Auswertung von Benutzerinteraktionen (View)
  - Manipulation von Daten (Model)
  - Zu jeder View existiert meistens genau ein Controller

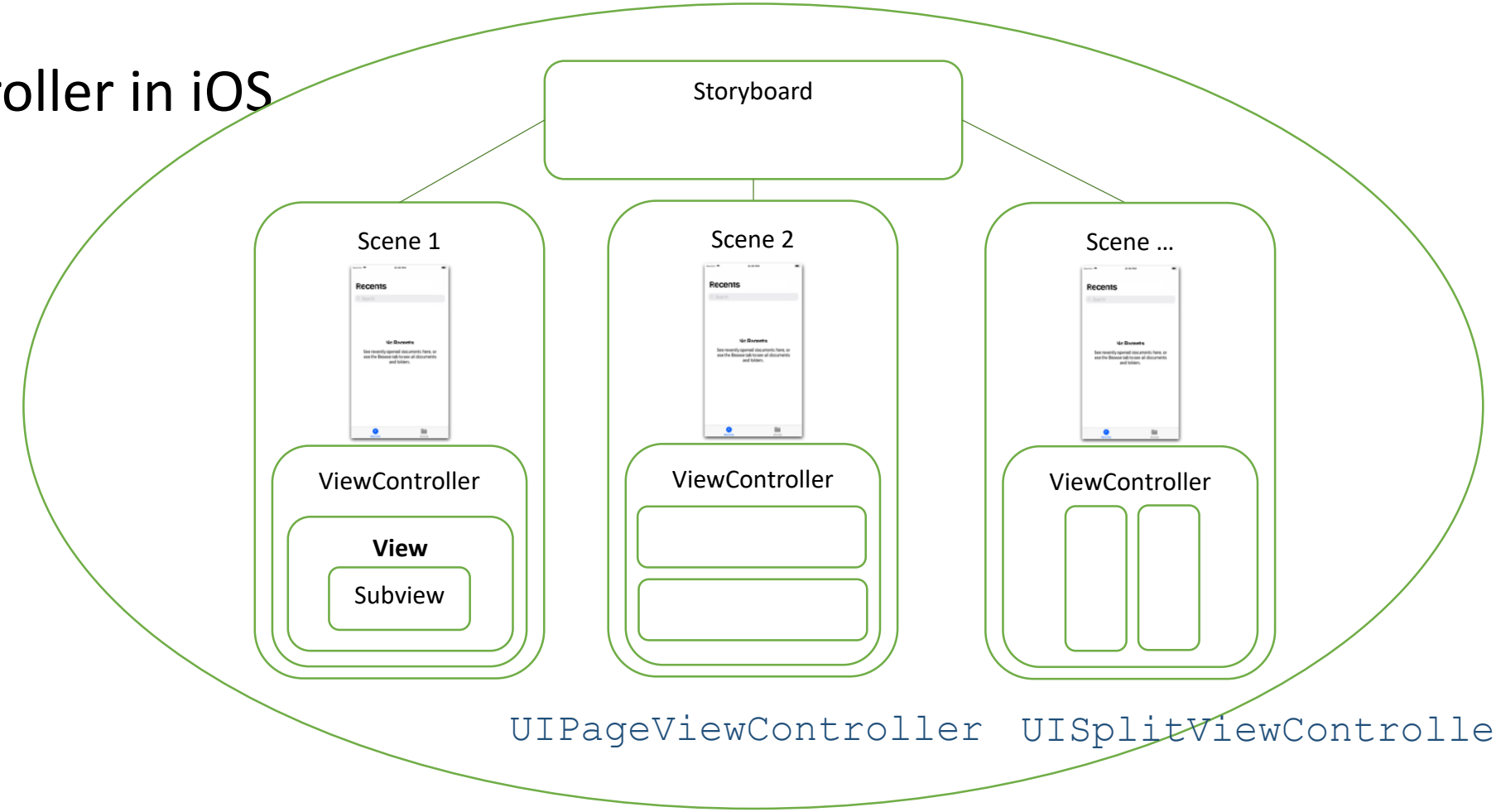


MVC - **Controller**

UIViewController

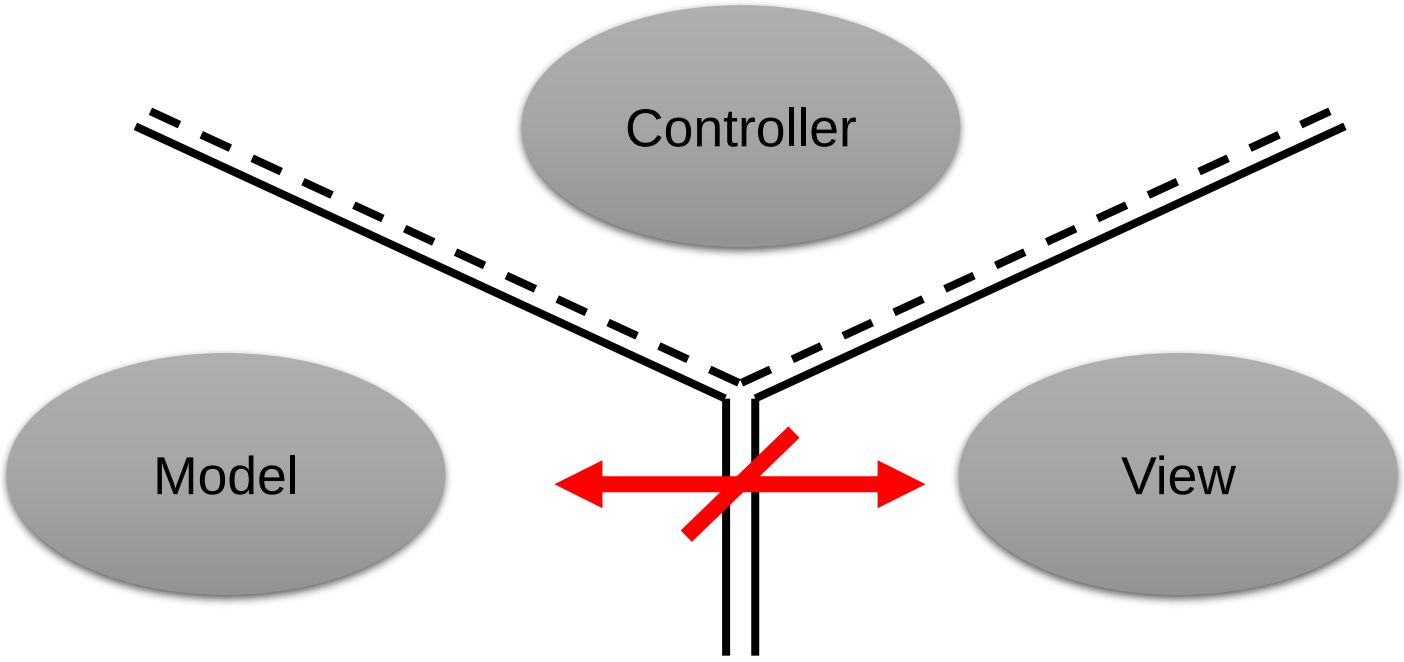
UITableViewController IOCollectionViewController

Controller in iOS



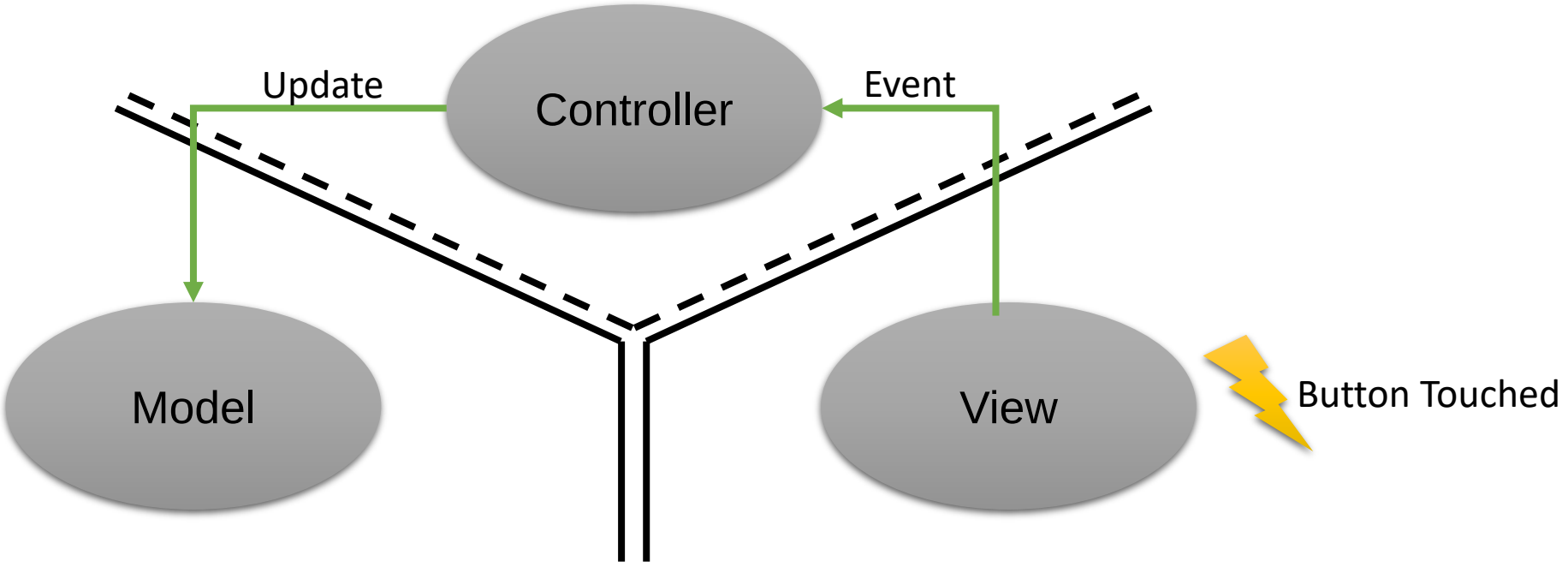
MVC - **Kommunikationswege**

- Wer kommuniziert mit wem?
- Wichtig: **Kein** Kommunikationspfad zwischen **Model** und **View**!



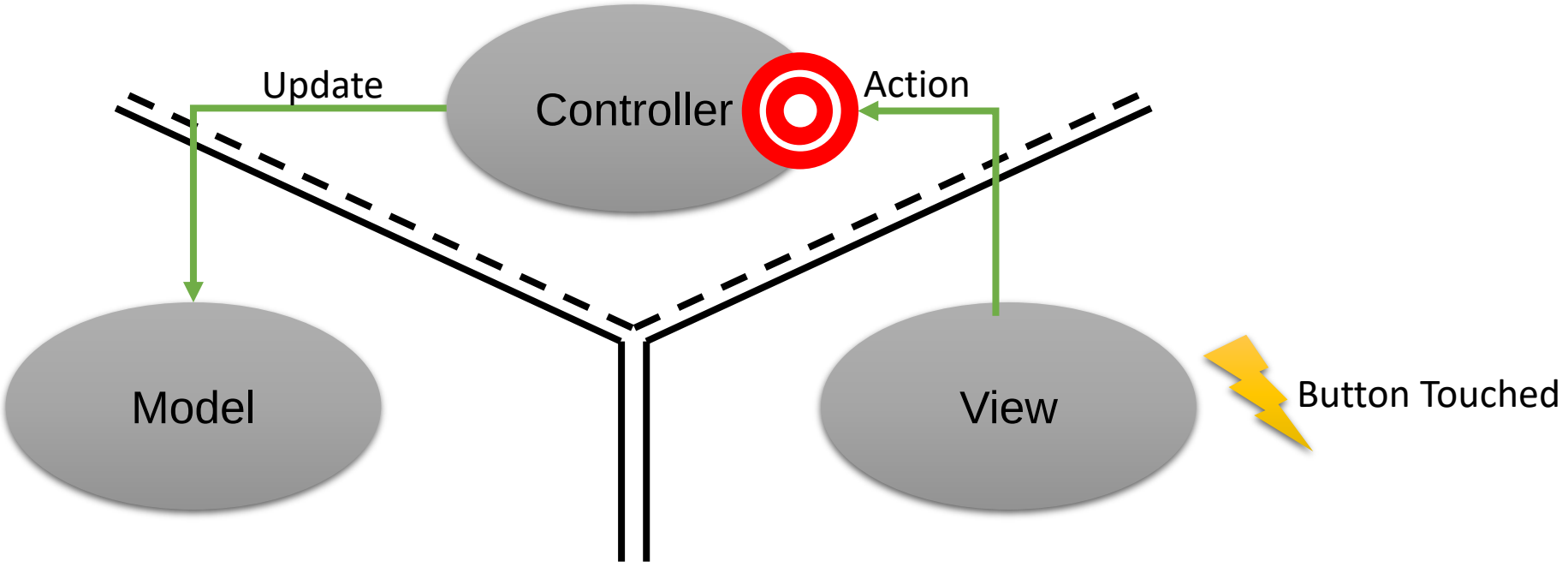
MVC – Kommunikationswege I

- **Benutzerinteraktion:**
  - Kommunikation von Ereignissen erfolgt als Aktion
  - Controller aktualisiert das Modell **nach Prüfung der Eingabe.**



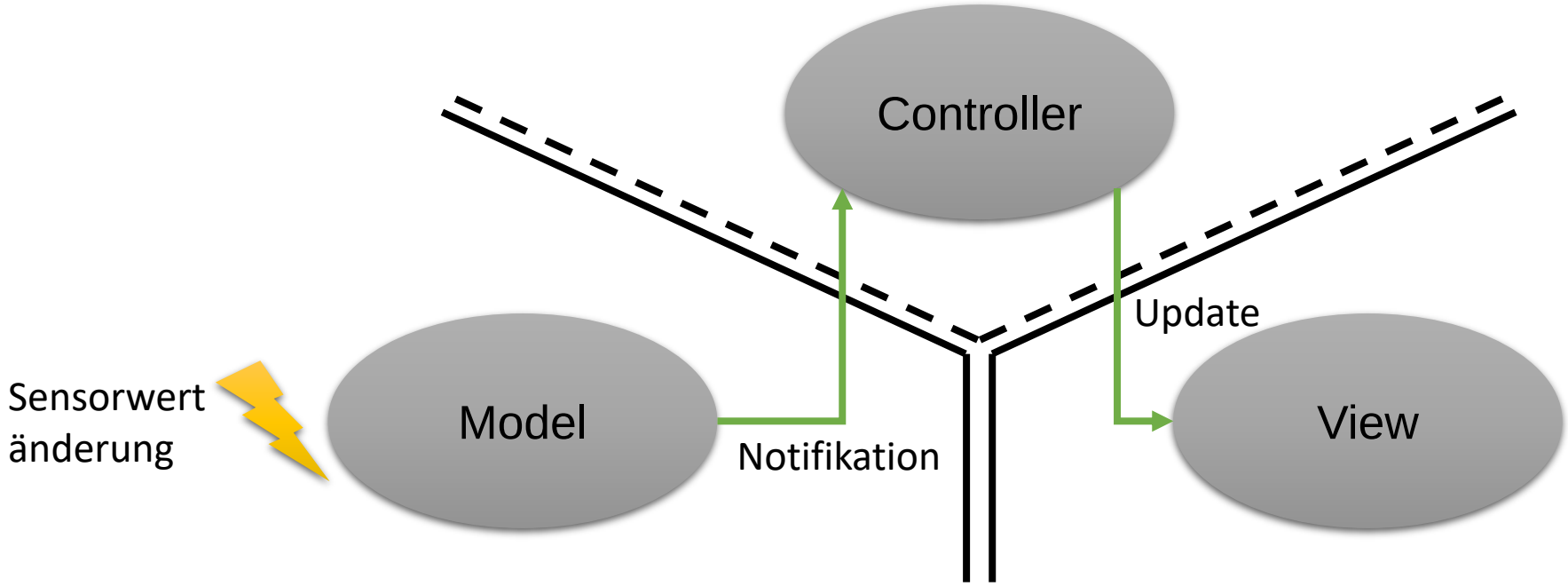
MVC – Kommunikationswege II

- **Benutzerinteraktion in iOS:**
  - Target-Action Mechanismus
  - UIControl.addTarget(targetObject, action, controlEvents)
  - targetObject ist für gewöhnlich der assoziierte ViewController



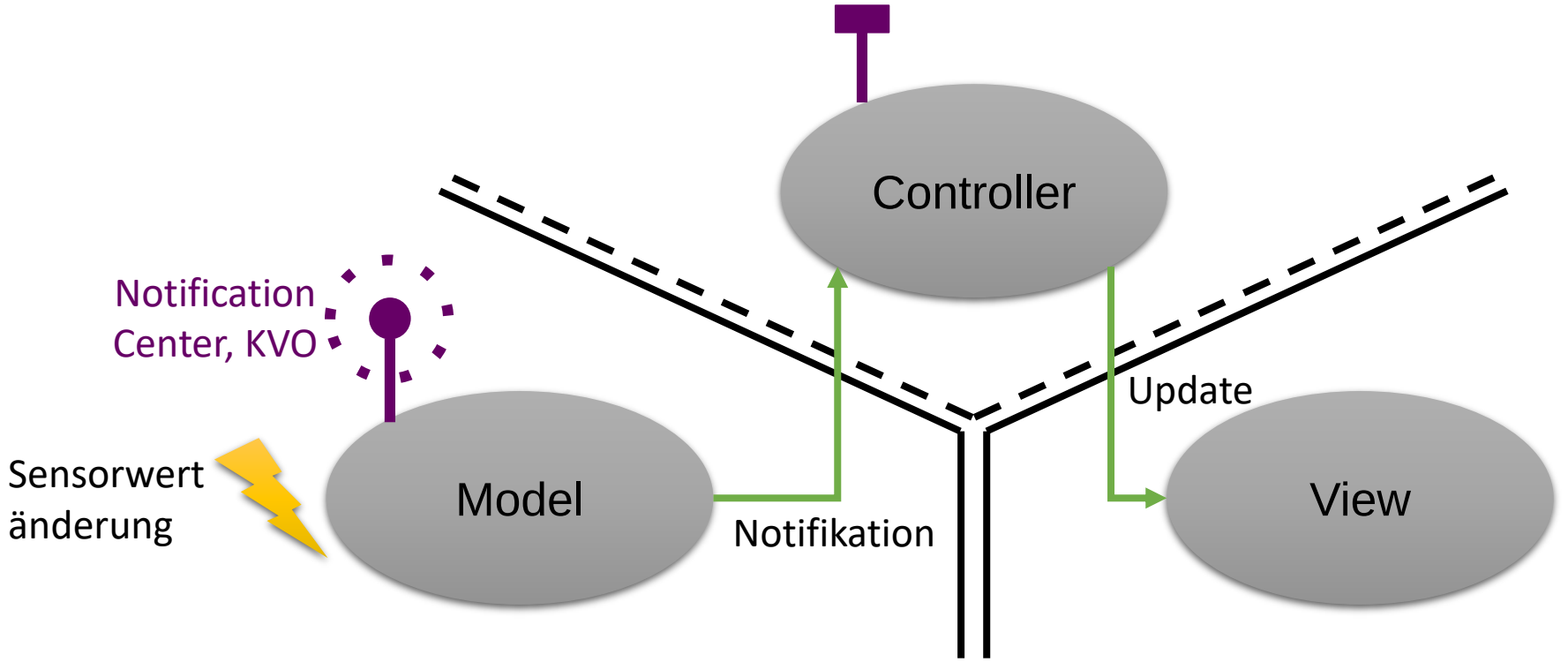
MVC – Kommunikationswege III

- **Datenaktualisierung:**
  - Model benachrichtigt nach Datenänderung den **Controller**
  - **Controller** aktualisiert **View**



MVC – Kommunikationswege IV

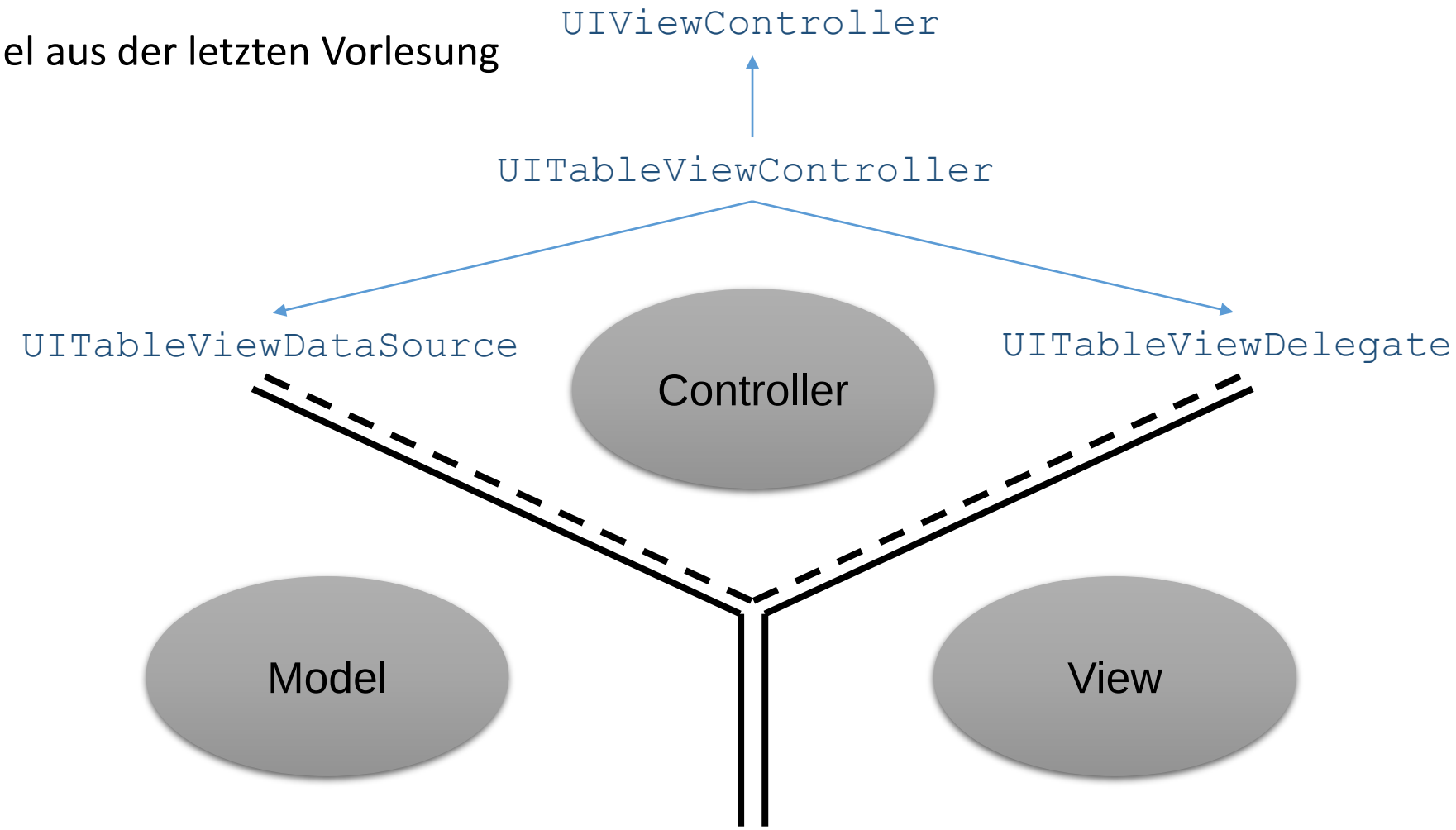
- **Datenaktualisierung in iOS:**
  - Notifikation kann durch Muster aus der Beobachterkategorie erfolgen(Delegation, Notification Center, KVO)
  - Verwendung von **Delegation** ist der am häufigsten verwendete Weg.



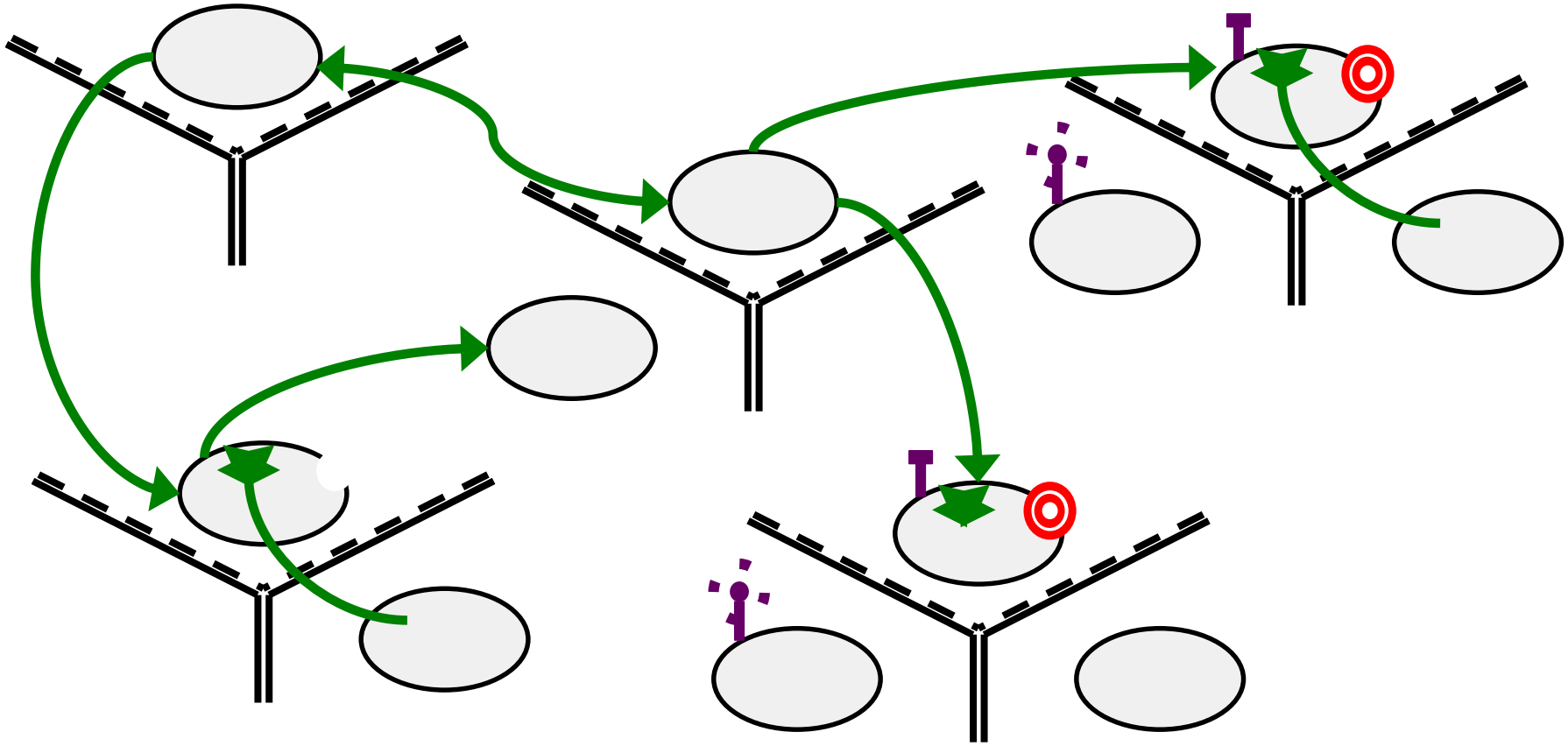
Delegation:
- Controller Implementiert Protokoll
- Controller übergibt sich selbst als Delegat an Model

MVC – **TableView Beispiel**

- Beispiel aus der letzten Vorlesung



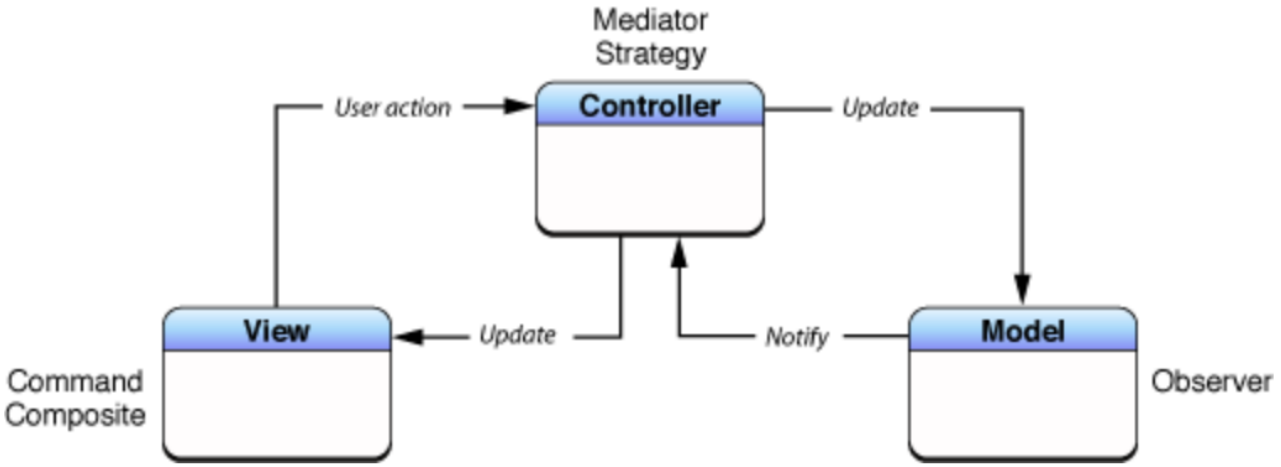
MVC



Komplexe Programme entstehen durch die Kombination mehrerer MVC-Gruppen.

MVC als Zusammensetzung von Entwurfsmustern

- Kommando:
  - Views‘ Target-Action Mechanismus
- Kompositum:
  - Views sind ineinander geschachtelt
- Vermittler:
  - View Controller vermittelt den Datenfluss zwischen Model und View
- Strategie:
  - UI Verhalten wird von View Controller implementiert (Delegate)
- Beobachter:
  - Model benachrichtigt View Controller bei Änderungen



Quelle: https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

Gefahren

- Entwurfsmuster sind kein Allheilmittel!
  - Codequalität != # Muster pro 1000 Zeilen Code
  - Hilft nicht gegen falsche Wahl von Algorithmen und/oder Datenstrukturen
  - Muster passen nicht auf jedes Problem

- Antimuster!

Interessante Links

- Entwurfsmuster in Swift
  - https://github.com/ochococo/Design-Patterns-In-Swift => Repo mit Implementierungen aller klassischen Muster
- MVC
  - https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52 => Von MVC über MVP zu MVVM
- Object Modeling mit dem Core Data framework
  - https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/ObjectModeling/ObjectModeling.html#//apple_ref/doc/uid/TP40010810-CH15-SW1 => Einführung

Themenausblick bis Weihnachten

- 22.11.: Gesten
- 29.11.: Core Data
- 06.12.: Core Motion
- 13.12.: ???
- 20.12.: Projektpräsentationen