App Architecture by Manual DI

@yoshikuni_kato

Tokyo iOS meetup
2018/07/21

Who am I ?

- Yoshikuni Kato（加藤由訓）
- iOS Engineer（3.5 years）
- Yahoo! Japan -> OHAKO -> Pangea
- Twitter: @yoshikuni_kato
- GitHub: @yoching
- Interests: Software Design, FRP (ReactiveSwift), UI Implementation

Agenda

1. Coordinator Pattern
2. Goals
3. Architecture Sample

Coordinator Pattern

Connecting View Controllers [1]

```
let nc = window?.rootViewController as! UINavigationController
let episodesVC = nc.viewControllers[0] as! EpisodesViewController

let storyboard = UIStoryboard(name: "Main", bundle: nil)

episodesVC.didSelect = { episode in
    let detailVC = storyboard.instantiateViewControllerWithIdentifier("Detail")
                            as! DetailViewController
    detailVC.episode = episode
    nc.pushViewController(detailVC, animated: true)
}
```

- transition logics are **outside** of view controller

[1] https://talk.objc.io/episodes/S01E05-connecting-view-controllers

Coordinator Pattern [2] [3]

- Objects to handle view controller transition = Coordinator
- View Controllers can be isolated each other -> DI friendly
- Other names: Router (in VIPER), Wireframe, Navigation, ...

---

[2] https://speakerdeck.com/yoching/hua-mian-qian-yi-falseguan-li-tomvvm

[3] https://speakerdeck.com/yoching/coordinatorpatanfalseshi-jian

More commonized way

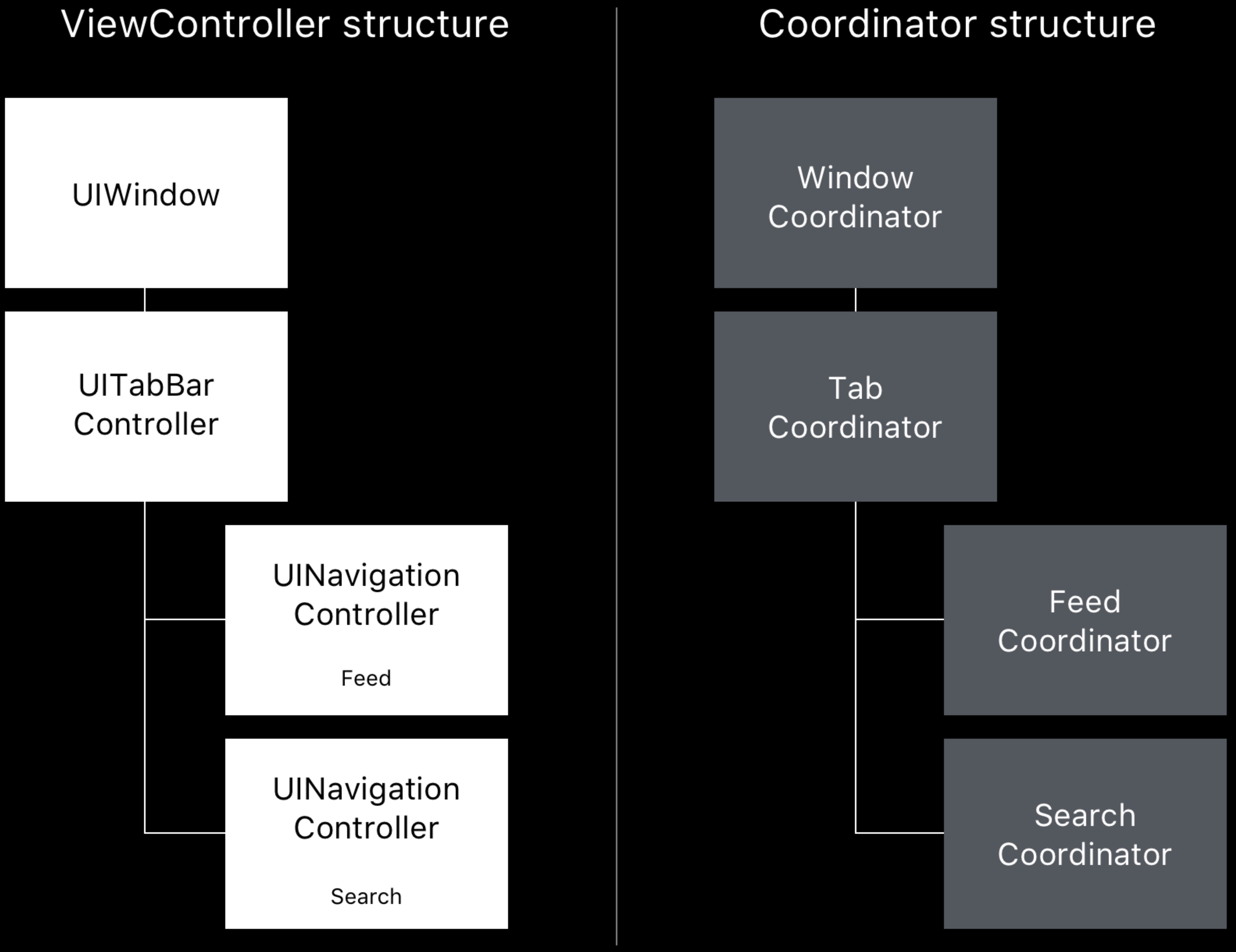
```swift
// in ViewController
enum EpisodesRoute {
    case detail(Episode)
}
protocol EpisodesRouting: class {
    var routeSelected: ((EpisodesRoute) -> Void)? { get set }
}
class EpisodesViewController: UIViewController, EpisodesRouting {
    var routeSelected: ((EpisodesRoute) -> Void)?
}

// in Coordinator
episodesVC.routeSelected = { route in
    switch route {
    case .detail(let episode):
        // present detail
    }
}
```

(inspired by "Deep Linking at Kickstarter" @ SwiftTalk [4])

[4] https://www.slideshare.net/DerekLee/minimizing-decision-fatigue-to-improve-team-productivity

Coordinator structure sample



Coordinator Pattern problems

- 2 tasks in Coordinator
  - view transition
  - view controller creation
- lots of dependencies

Goals

Goals

- All dependencies are injected from outside
- Coordinator doesn't do view controller creation
- Project is well organized

Goals

- All dependencies are injected from outside
  -> Manual DI [5]
- Coordinator doesn't do view creation
  -> using `ViewFactory`, `CoordinatorFactory`
- Project is well organized
  -> Application / UI / Component [6]

[5] https://ja.wikipedia.org/wiki/%E4%BE%9D%E5%AD%98%E6%80%A7%E3%81%AE%E6%B3%A8%E5%85%A5

[6] https://talk.objc.io/episodes/S01E49-deep-linking-at-kickstarter

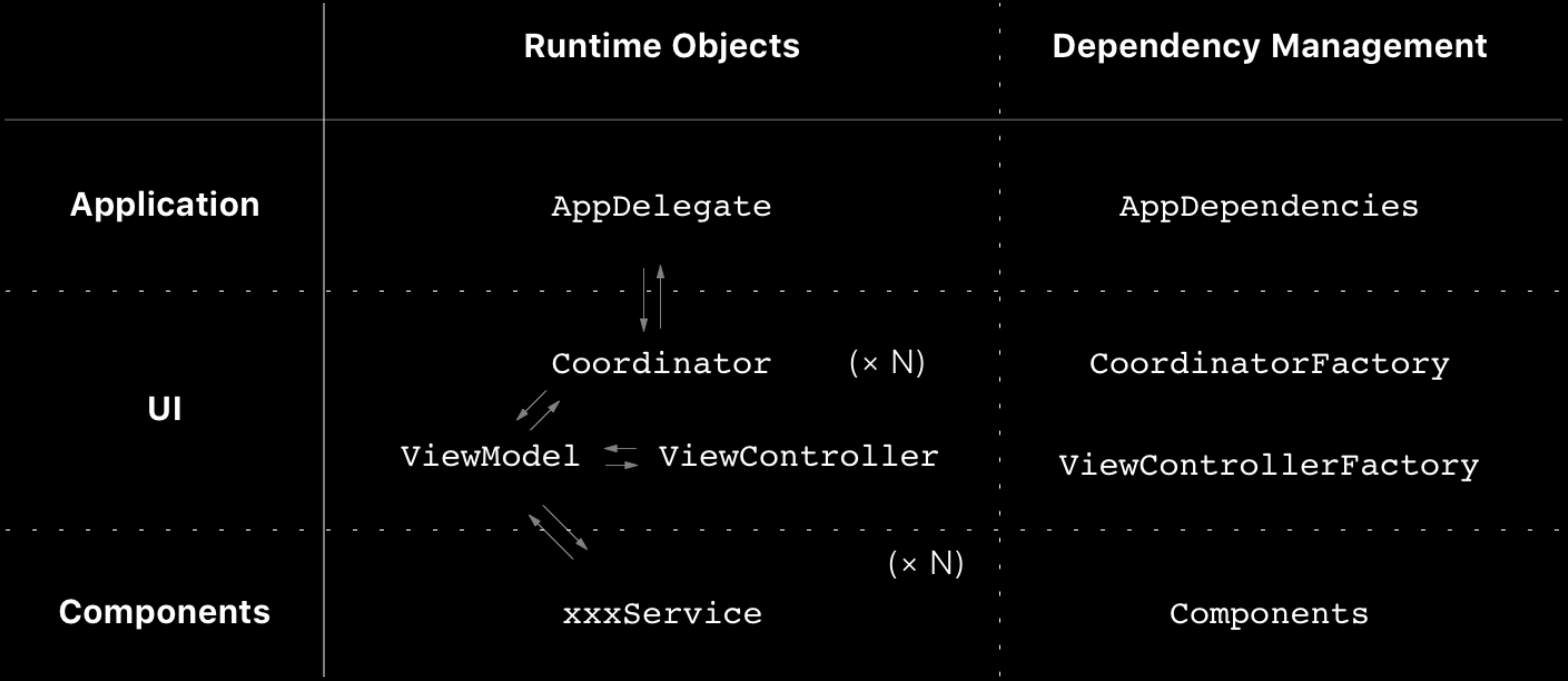
Architecture Sample

Sample Code

- yoching/iOSAppArchitectureSample [7]

[7] https://github.com/yoching/iOSAppArchitectureSample

Figure


Runtime Objects
Dependency Management
Application
AppDelegate
AppDependencies
UI
Coordinator
(× N)
CoordinatorFactory
ViewModel
ViewController
ViewControllerFactory
(× N)
Components
xxxService
Components

Development Workflow

situation	workflow
make service	make service -> update `Components`
make view	make VC & VM -> make function at `ViewFactory`
make transition	update `Coordinator`

More Practical Sample

- yoching/JSONPlaceholderViewer [8]
  - persistance using CoreData
  - networking
  - ReactiveSwift

[8] https://github.com/yoching/JSONPlaceholderViewer

Discussions

- Over engineered?
- Dependency management objects = DI container?

Thank you!

@yoshikuni_kato