WOTA

51CTO

World Of Tech 2017

全球架构与运维技术峰会

2017年4月14日-15日　北京富力万丽酒店

ARCHITECTURE

WOTA

51CTO

出品人及主持人：

王朝成

饿了么 首席移动架构师

移动端架构演进

WOTA

51CTO

第十年的选择

Mobile Application Architecture

WOTA

51CTO

唐平麟

咕咚技术总监

SwiftyJSon作者

分享主题：

第十年的选择

Mobile App Architecture

我们如何选择移动架构

什么才是好的架构

高内聚

模块内的处理元素都密切相关，共同完成功能

低耦合

按照业务和层级分模块，降低模块之间的依赖

易测试

可以针对类或者方法单元测试和性能测试

易扩展

架构能够考虑未来业务的发展

高

高内聚

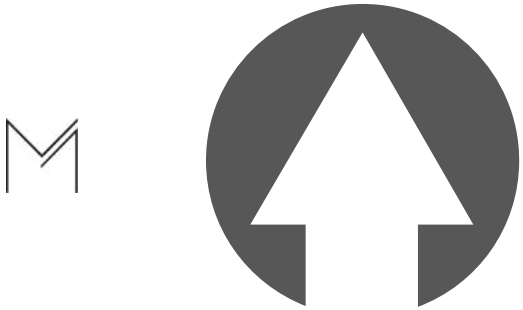
低

低耦合

今天分享的内容

内容局限在从移动开发中接触到的移动架构介绍和设计思想，以 iOS 为主。

- 什么才是好的架构
- MVC
- MVVM
- VIPER
- Riblets



无直接耦合

数据耦合

指两个模块之间有调用关系，传递的是简单的数据值，相当于高级语言的值传递

标记耦合

指两个模块之间传递的是数据结构，如高级语言中的数组名、记录名、文件名等这些名字即标记，其实传递的是这个数据结构的地址

控制耦合

指一个模块调用另一个模块时，传递的是控制变量（如开关、标志等），被调模块通过该控制变量的值有选择地执行块内某一功能

公共耦合

指通过一个公共数据环境相互作用的那些模块间的耦合。公共耦合的复杂程序随耦合模块的个数增加而增加。

内容耦合

这是最高程度的耦合，也是最差的耦合。当一个模块直接使用另一个模块的内部数据，或通过非正常入口而转入另一个模块内部

偶然内聚

指一个模块内的各处理元素之间没有任何联系

逻辑内聚

指模块内执行几个逻辑上相似的功能，通过参数确定该模块完成哪一个功能

时间内聚

把需要同时执行的动作组合在一起形成的模块为时间内聚模块

通信内聚

指模块内所有处理元素都在同一个数据结构上操作（有时称之为信息内聚），或者指各处理使用相同的输入数据或者产生相同的输出数据

顺序内聚

指一个模块中各个处理元素都密切相关于同一功能且必须顺序执行，前一功能元素输出就是下一功能元素的输入

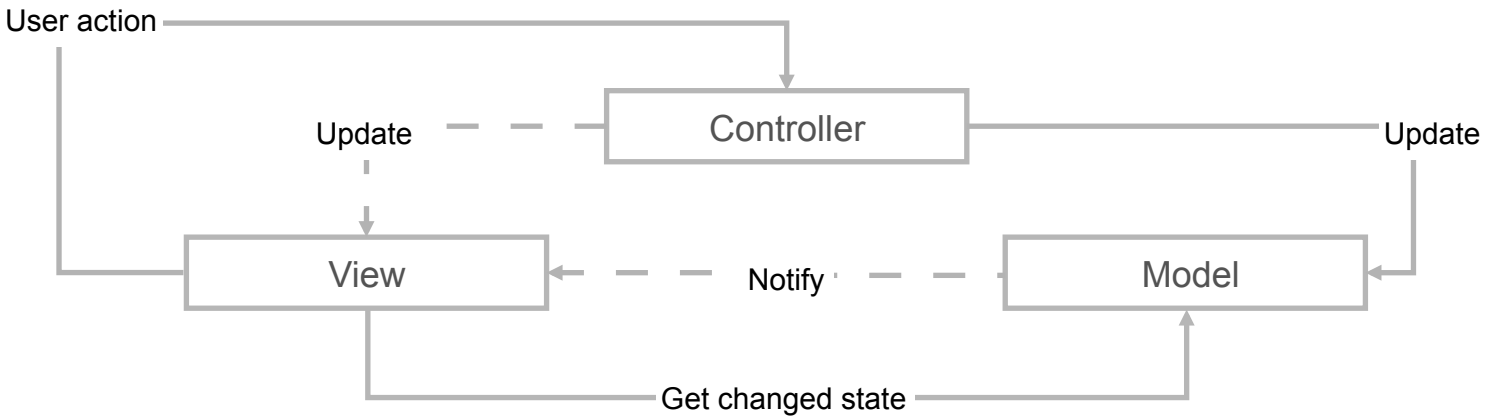
功能内聚

这是最强的内聚，指模块内所有元素共同完成一个功能，缺一不可。与其他模块的耦合是最弱的

M V C

MVC

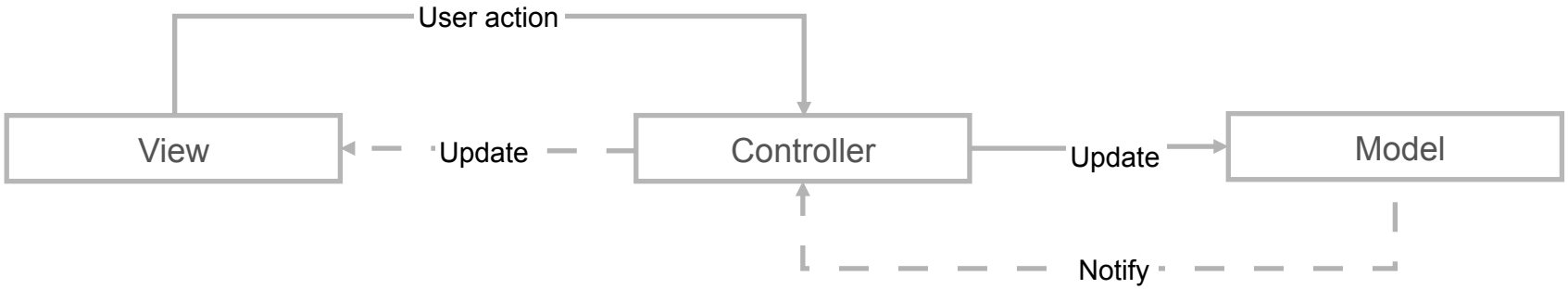
经典的MVC



Model就是作为数据管理者，View作为数据展示者，Controller作为数据加工者

MVC

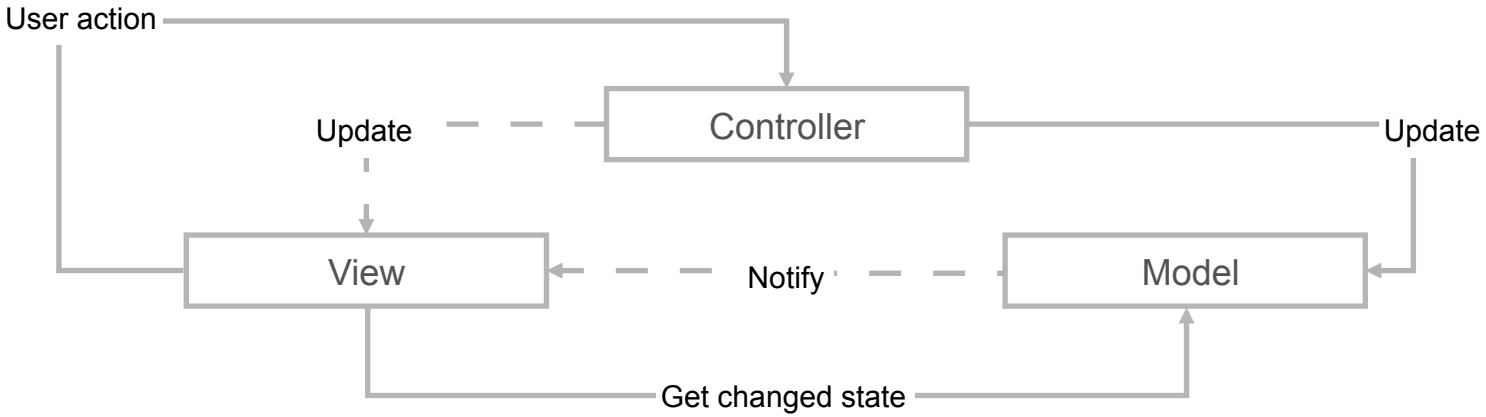
Cocoa MVC

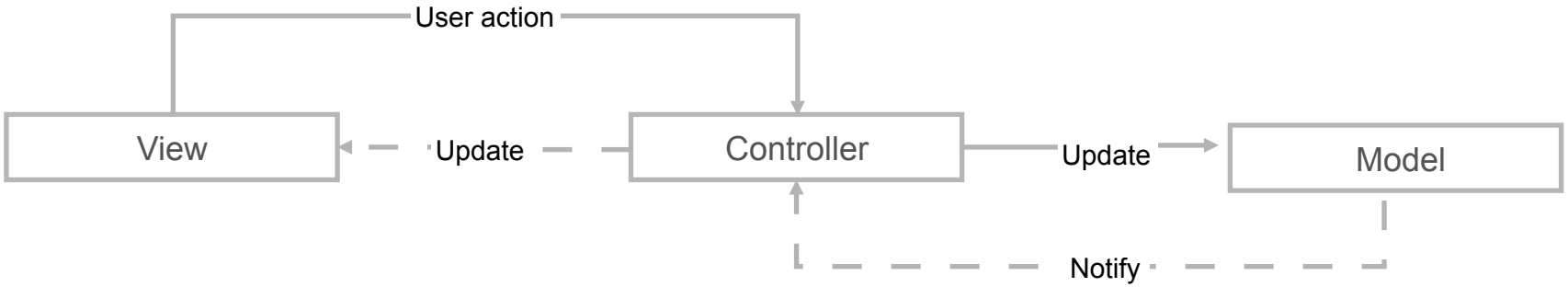


Controller是一个介于View 和 Model之间的协调器

MVC

传统的MVC和Cocoa MVC的比较

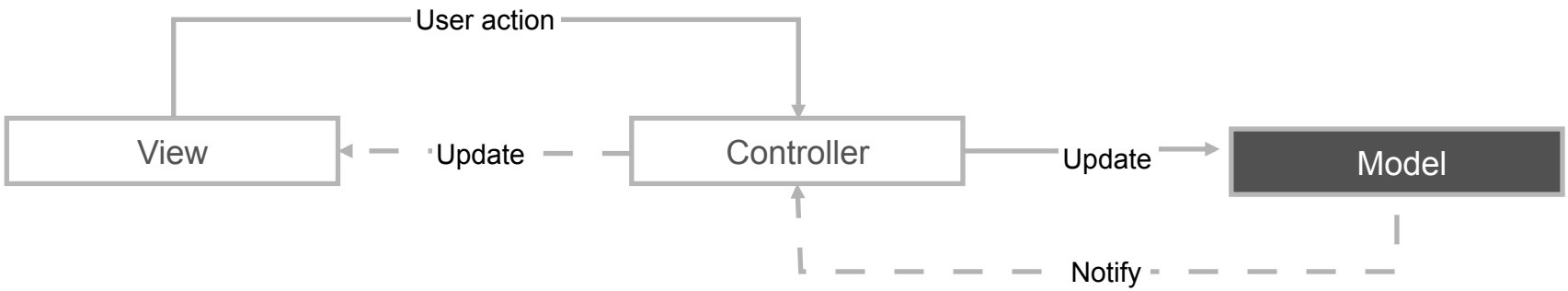




Cocoa MVC = MVC + Data Service

MVC

Cocoa MVC



Model

程序中要操纵的实际对象的抽象，包含了代表着实际对象属性(例如汽车的颜色)的属性(property)和操作这些属性的方法(method)，在MVC的相互协作中，View会通过Controller来向Model索要数据，经由Controller转换之后展示到View上，同时会将用户操作通过Controller反馈到Model中，更新Model的内容。而至于如何获取需要的数据，以及如何处理用户发出的数据请求，这些通常都定义在Model中。

FatModel

胖

胖Model包含了部分弱业务逻辑。胖Model要达到的目的是，Controller从胖Model这里拿到数据之后，不用额外做操作或者只要做非常少的操作，就能够将数据直接应用在View上。

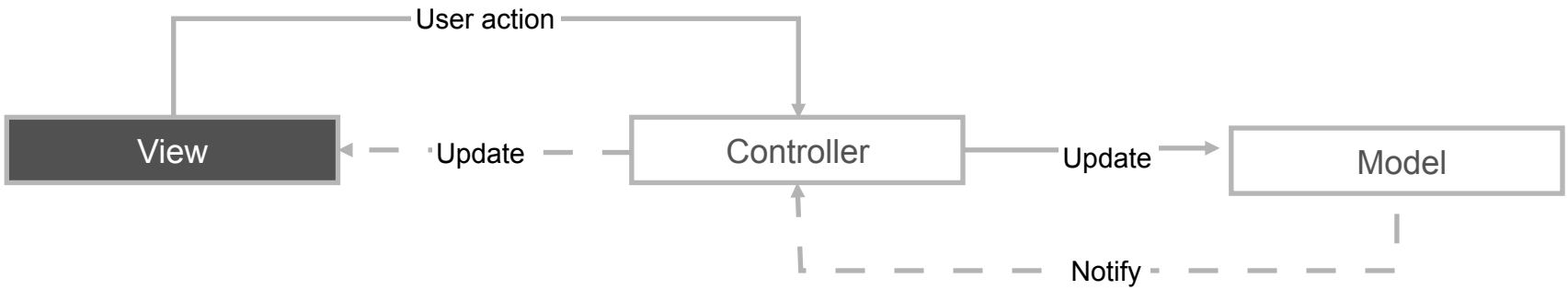
SlimModel

瘦

瘦Model只负责业务数据的表达，所有业务无论强弱一律扔到Controller。瘦Model要达到的目的是，尽一切可能去编写细粒度Model，然后配套各种helper类或方法来对弱业务做抽象，强业务依旧交给Controller。

MVC

Cocoa MVC



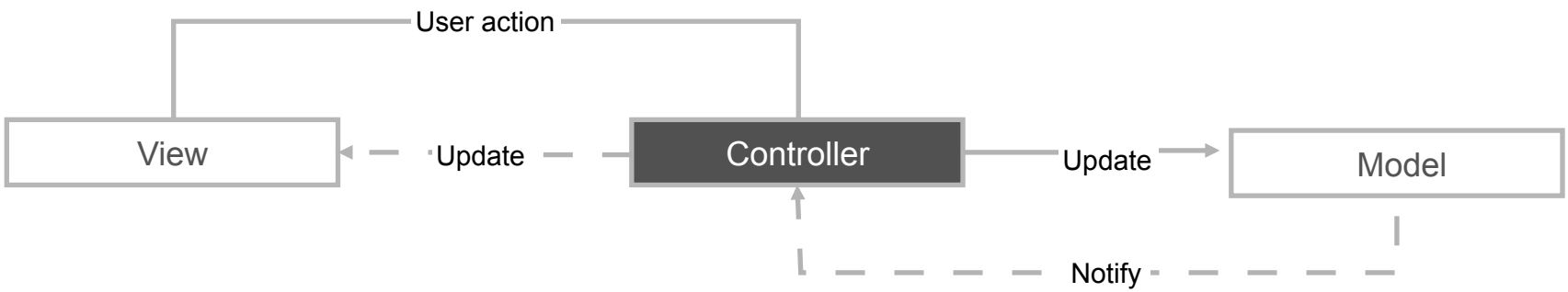
View

View只负责机械地展示来自Model的数据，并且将用户的操作反馈到Controller中，自己并不参与到整个数据的处理过程当中。在iOS开发中，标签、文本框、表单、图片、滑动页等等这些都属于View这一类。

iOS开发领域，虽然也有让View去监听事件的做法，但这种做法非常少，都是把事件回传给Controller，然后Controller再另行调度。所以这时候，View的容器放在Controller就非常合适。Controller可以因为不同事件的产生去很方便地更改容器内容，比如加载失败时，把容器内容换成失败页面的View，无网络时，把容器页面换成无网络的View等等。

MVC

Cocoa MVC

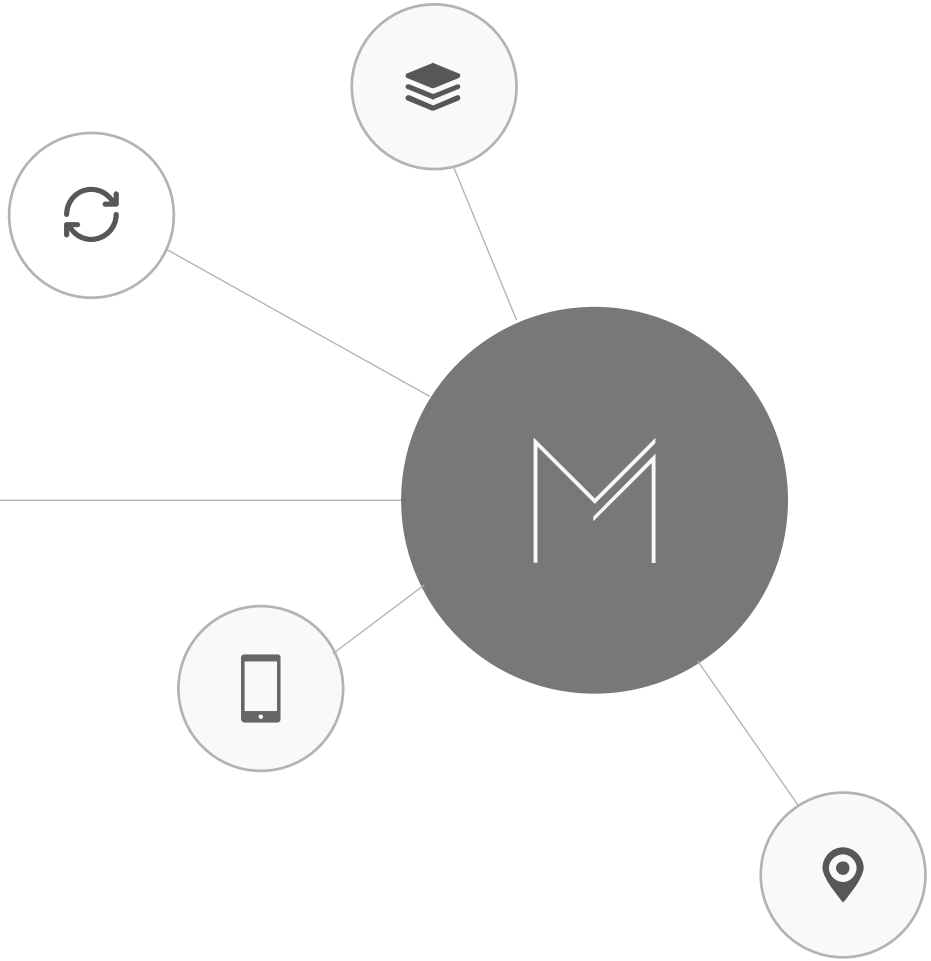


Controller

可以说Controller是MVC中最重要的一部分，Controller负责从Model获取数据，经过处理后交给View(Controller直接持有View)展示；同时，Controller负责对用户在View上的操作进行响应，并相应地更新Model。

Model-View-Controller

Cocoa MVC



在iOS中，View和Controller分的并不是特别清楚，有人认为iOS开发中并没有什么View和Controller，只有Model+ViewController

应用广泛

几乎所有的前台程序都有类似 MVC 的设计痕迹

便于理解和使用

设计思想非常的简洁，学习成本很低，新人上手非常的容易。资料和文档都非常齐全

高耦合

Controller View Model之间难以分离。

复杂的代码

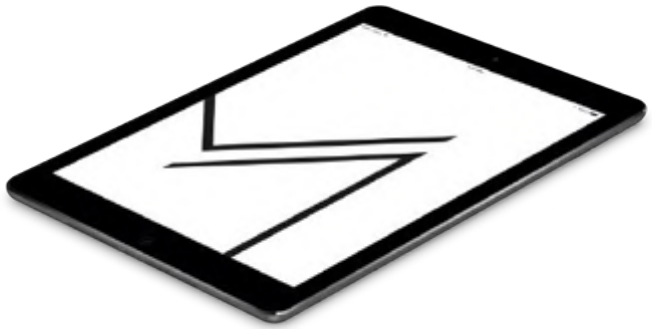
代码划分不明确的情况下，代码会越来越难维护

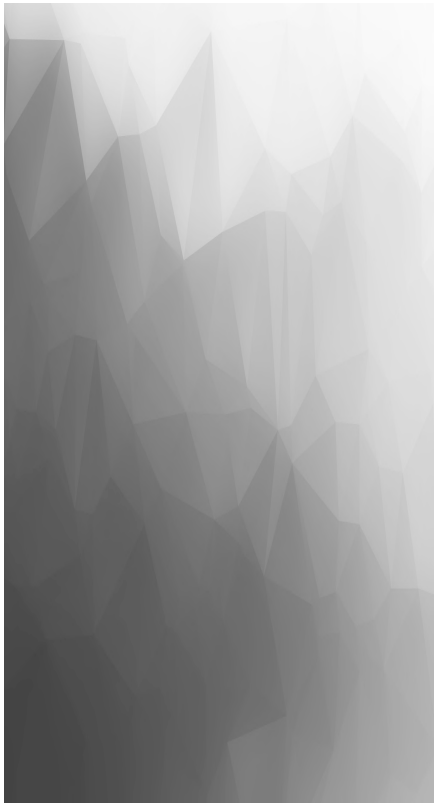
Model-View-Controller

MVC 的问题

MVC 中并没有定义得很清楚究竟应该放在什么地方，导致他们很容易就会堆积在 Controller 里

由于Model、View、Controller三个部分之间的耦合非常紧密，导致对显示逻辑的单元测试几乎是不可行的，你只要实例化Controller就必须要同时实例化一堆与之相关联的View





Model-View-Controller

MVC中如何划分代码?

Model View Controller分别应该如何划分才更加合理?

Model

给ViewController提供数据
给ViewController存储数据提供接口
提供经过抽象的业务基本组件，供Controller调度

Controller

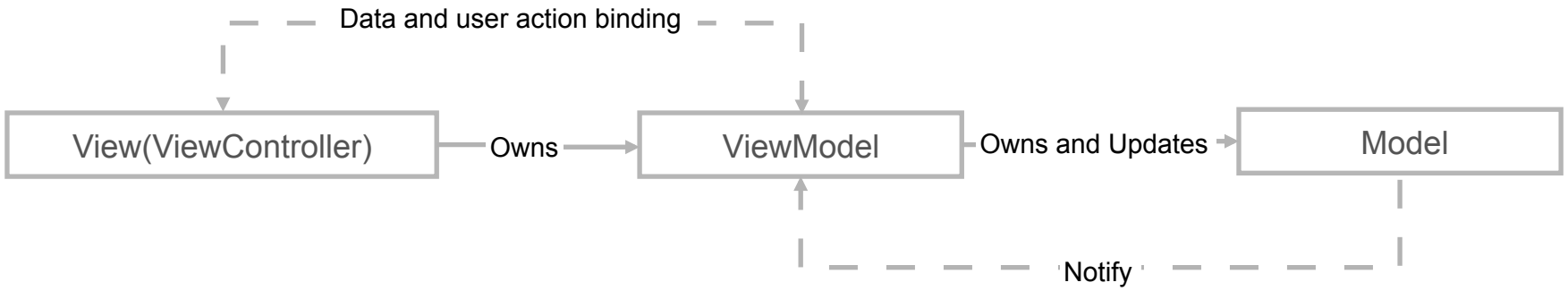
管理View的生命周期
负责生成所有的View实例，并放入View容器中
监听来自View与业务有关的事件，完成对应事件的业务

View

响应与业务无关的事件，并因此引发动画效果
处理点击反馈
界面元素表达

M V V M

MVVM

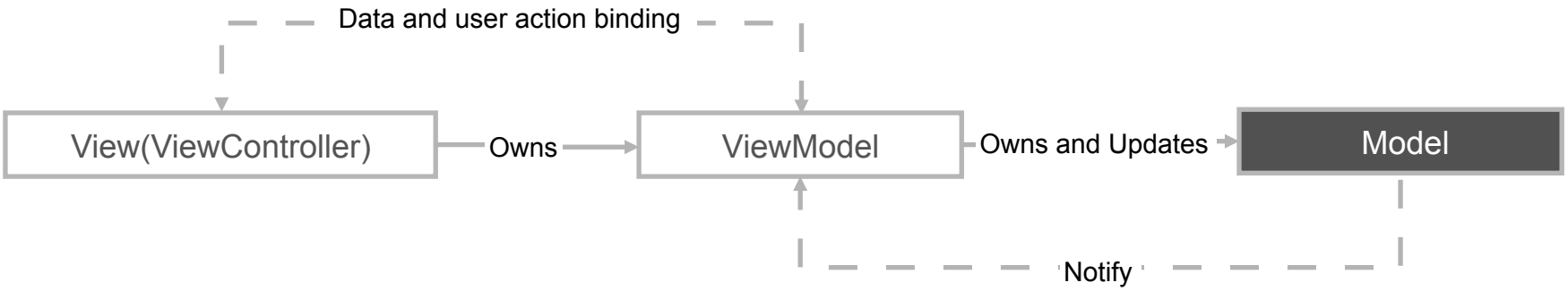


MVVM本质上也是从MVC中派生出来的思想，MVVM着重想要解决的问题是尽可能地减少Controller的任务。

Model的抽象是基于对业务（现实）层级，而ViewModel的抽象是基于视图（表现）层级，这样就就解决了MVC模式下Controller对于业务数据和视图数据之间的翻译、控制。比如：Model会抽象用户数据（昵称、年龄、性别）；ViewModel拥有页面上某个开关状态，6位Pin码中每个数字的值。

View(ViewController)和ViewModel之间进行双向绑定后，对于ViewModel的赋值操作非常的降低和清晰。并且控件之间的联动也变的非常容易。

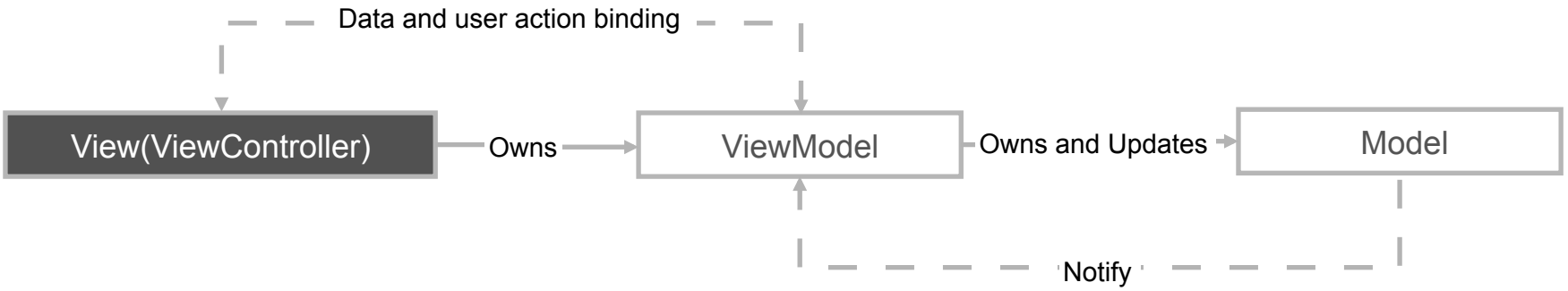
MVVM



Model

程序中要操纵的实际对象的抽象。胖瘦Model的设计都是可行的。

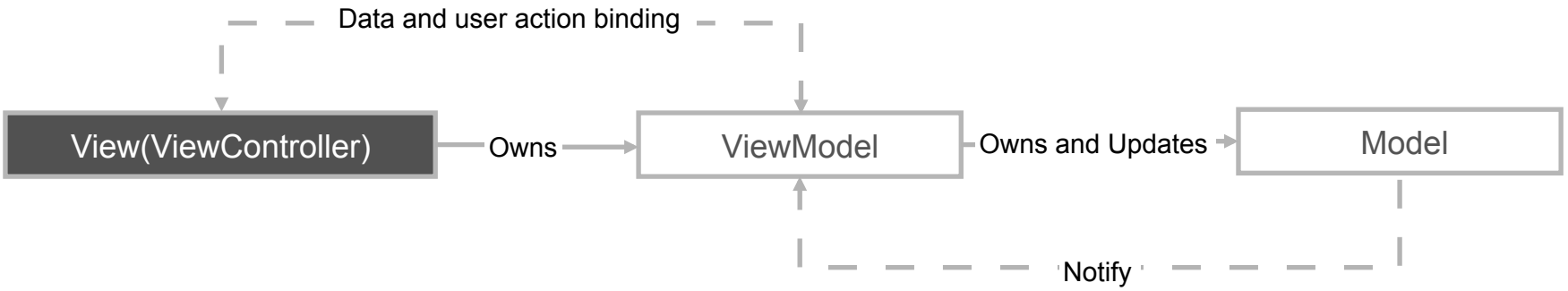
MVVM



View(ViewController)

在MVVM中，View不再是UIView的子类，而变成了UIViewController的子类。因此我们可以在上图中看到，View实际是和ViewController绑定在一起的。这种View实际上就是MVC中剥离了处理呈现View的逻辑部分的Controller，因此它仍然有各种UIView的属性，仍然有ViewController生命周期的各种方法(因此View部分负责将视图展示出来，也负责响应用户的操作)，但是它不知道该展示些什么数据(实际上MVC中的Controller也是不知道的)，它也不知道该用什么方法处理并展示来自Model的数据(这个是MVC中的Controller知道的)。少做了一大部分工作之后，View(Controller)终于不再臃肿了。

MVVM



ViewModel

在MVVM中，扮演协调者(Interactor)角色的不再是Controller，而变成了ViewModel。ViewModel被View持有，同时也持有着Model。ViewModel中定义了如何从Model获取数据、如何更新Model、如何处理用户的数据请求以及何时以何种方式更新View等等的众多方法，可以说是MVC中Controller的一个精简版。
ViewModel做什么事情？就是把RawData变成直接能被View使用的对象的一种Model。

Binding & React Programming

基于KVO

RZDataBinding

SwiftBond

React Programming

ReactiveCocoa

RxSwift

PromiseKit

Model-View-ViewModel

React Programming

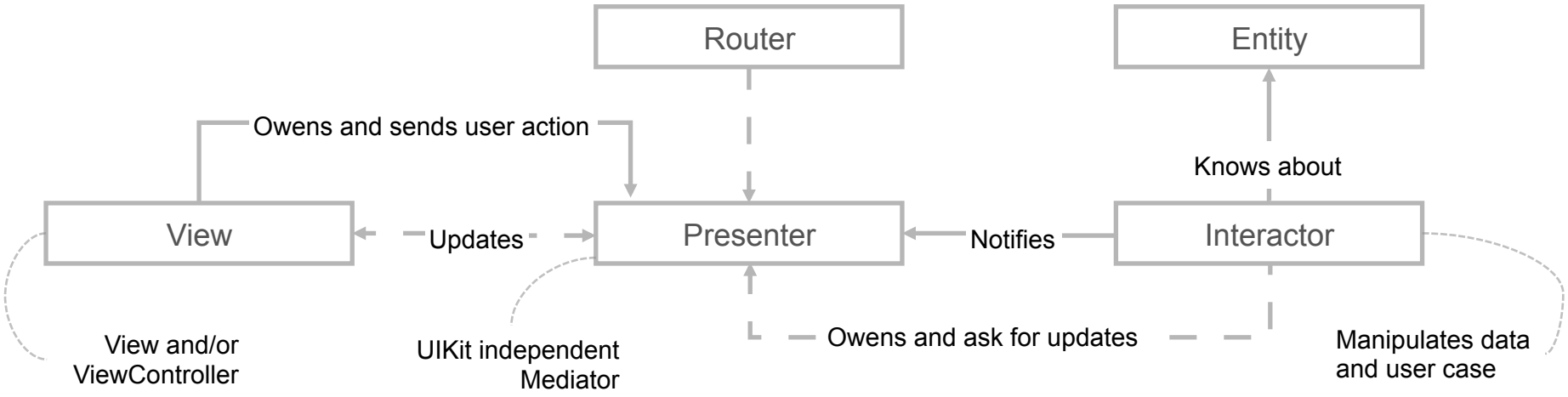
数据绑定使得 Bug 很难被调试。你看到界面异常了，有可能是你 View 的代码有 Bug，也可能是 Model 的代码有问题。数据绑定使得一个位置的 Bug 被快速传递到别的位置，要定位原始出问题的地方就变得不那么容易了。

并且对于过大的项目，数据绑定需要花费更多的内存

VIPER

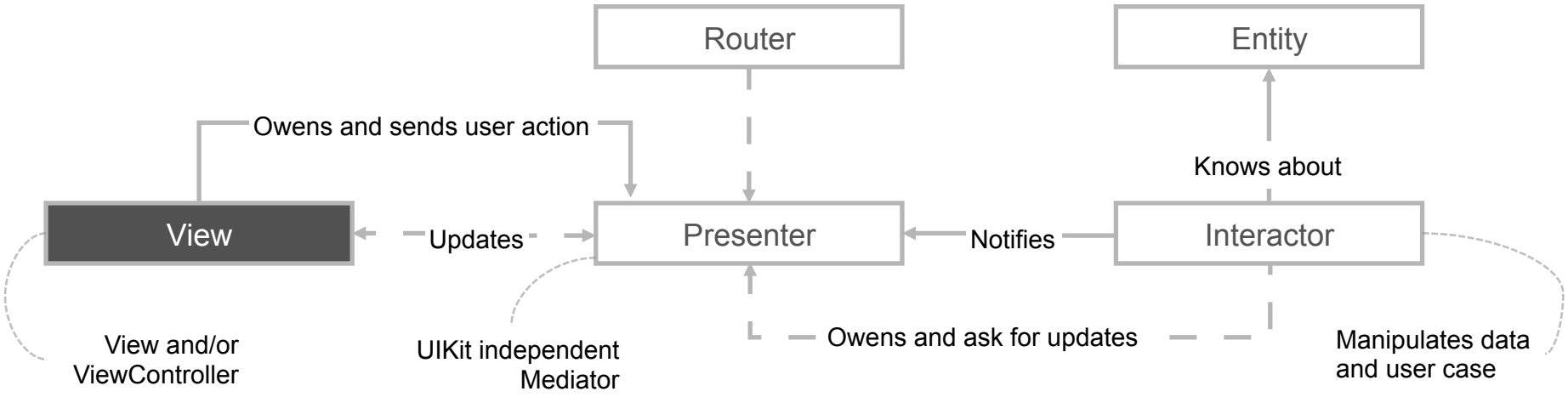
用例(Use Case)：用例就是指用户会用你的程序做的事情，一个事情成为一个用例。比如我点击了这个按钮，程序会有什么样的反应，这就是一个用例。

VIPER



VIPER是一种很有意思的架构，因为它把要处理的职责划分成了五层（View-Interactor-Presenter-Entity-Router）。实际上VIPER是MVVM的一个更细致的划分，让各个模块有了更清晰单一的分工。

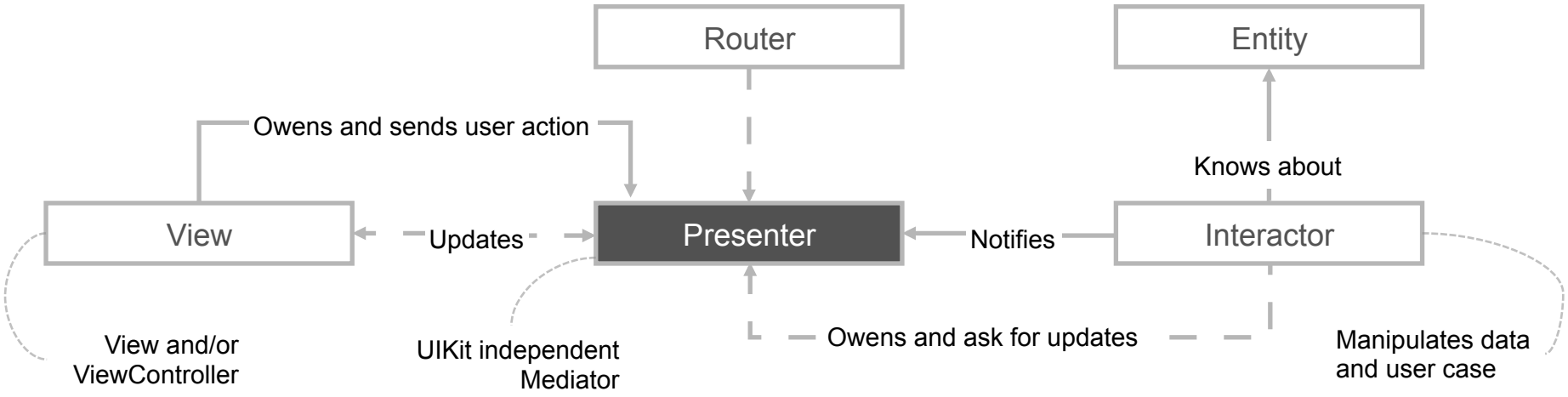
VIPER



View

和MVVM中的View一样，它也是UIViewController的子类，它仍然负责将各种UI组件展示到屏幕上。但是与MVVM不同的是，它现在不负责响应用户对UI的操作了，这个响应部分现在被移到了展示器(Presenter)里。也就是说View根据展示器(Presenter)的要求显示界面，并将用户输入反馈给展示器(Presenter)。

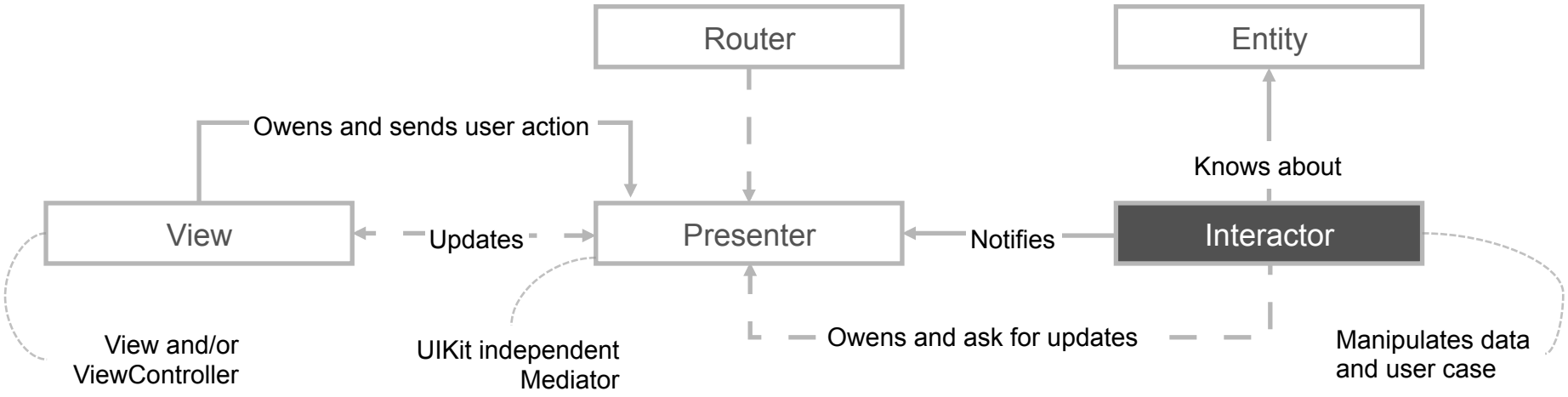
VIPER



Persenter

展示器(Persenter)负责将业务逻辑(Business logic)和视图逻辑(View logic)联结在一起的展示逻辑(Presentation logic)，还负责响应各种用户事件(比如按钮的点击)。包含为显示（从交互器(Interactor)接受的内容）做的准备工作的相关视图逻辑，并对用户输入进行反馈（从交互器获取新数据）。展示器(Persenter)对View的通信仍然可以使用React Programming来进行，因为这个方向上的通信和MVVM实际上是一致的，都是在刷新UI

VIPER

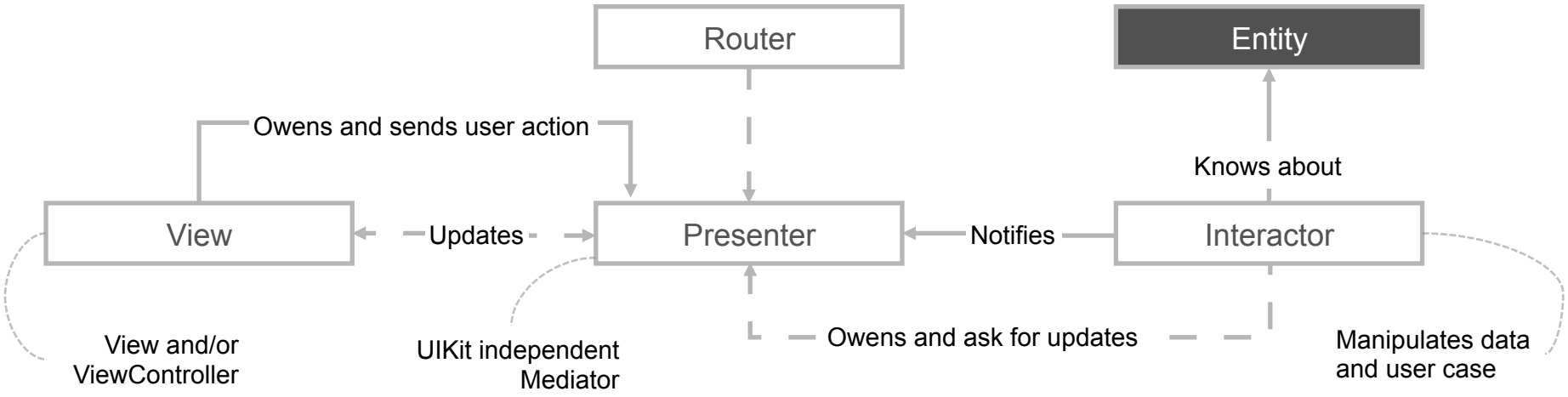


Interactor

交互器(Interactor)包含由用例指定的业务逻辑。交互器(Interactor)负责处理用例中规定好的逻辑，交互器(Interactor)负责获取并处理数据，并将数据送给展示器(Presenter)。交互器Interactor处理纯粹的数据操作和验证，包括对后端服务的调用来控制状态变化等。交互器(Interactor)主要包含了逻辑，因此很容易使用 TDD 进行开发。

交互器(Interactor)在应用中代表着一个独立的用例。它具有业务逻辑以操纵模型对象（实体）执行特定的任务。交互器中的工作应当独立与任何用户界面，同样的交互器可以同时运用于 iOS 应用或者 OS X 应用中。

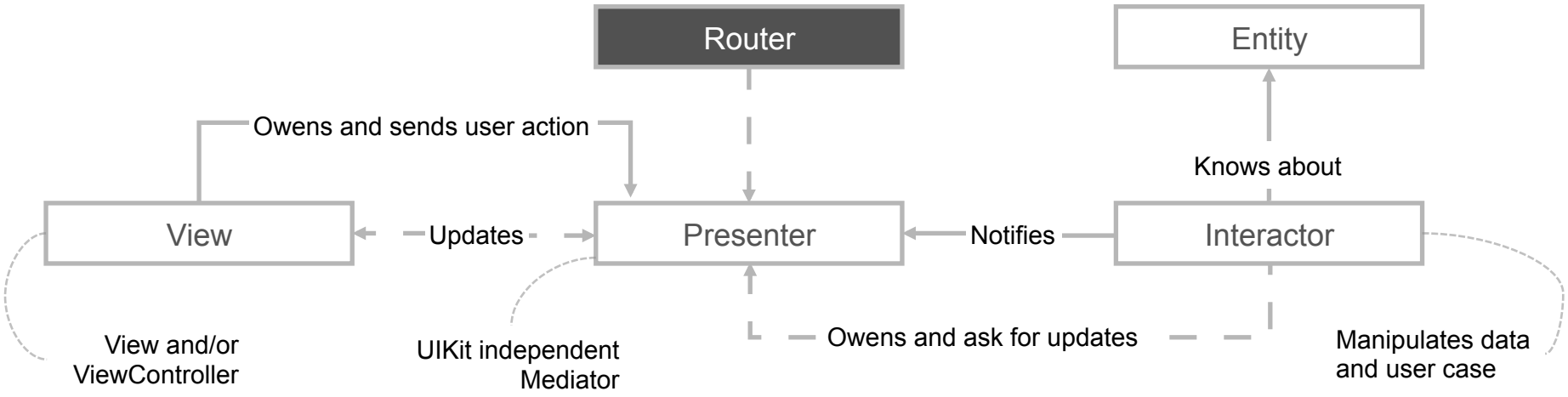
VIPER



Entity

实体(Entity)仅仅是一个数据结构的定义，它定义了程序要处理的对象应该具有哪些属性，而没有定义处理这些数据的方法(这一点和MVC、MVVM中的Model不同)。实体是被交互器操作的模型对象，并且它们只被交互器(Interactor)所操作。交互器永远不会传输实体至表现层 (比如说展示器)。

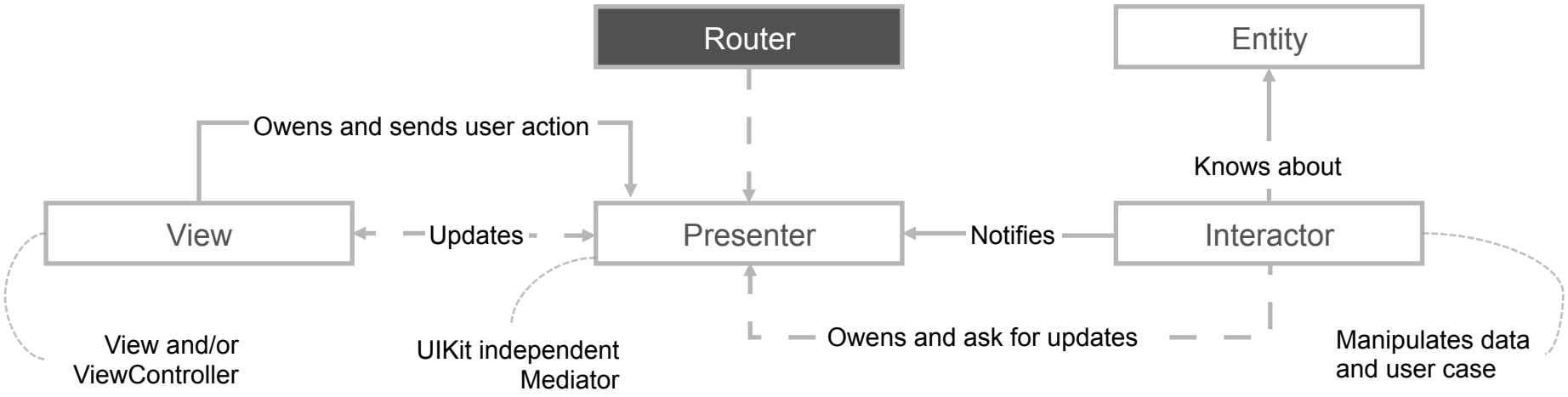
VIPER



Router

路由(Router)负责处理各种转场(Transition)逻辑，也就是用于实现导航功能。以前写在Controller里的presentViewController、performSegue等方法就可以放在这里来实现了。包含用来描述屏幕显示和显示顺序的导航逻辑

VIPER



通过对VIPER中各个模块分工的介绍我们可以发现，VIPER实际上是对MVVM的各个部分进行了进一步的细化，但又不仅仅是对VVM是更为细致的划分，同时两者的各个模块又有着功能上的交叉。View和Presenter是对MVVM中的View的进一步细化，同时Presenter又包揽了一部分ViewModel的工作(例如Presentation logic的处理)；

1. Interactor和Entity是对Model的细化，但Interactor又包含了一部分ViewModel的工作(例如根据用户的操作来更新Model/Entity)；
2. Router则又分担了View的一部分工作

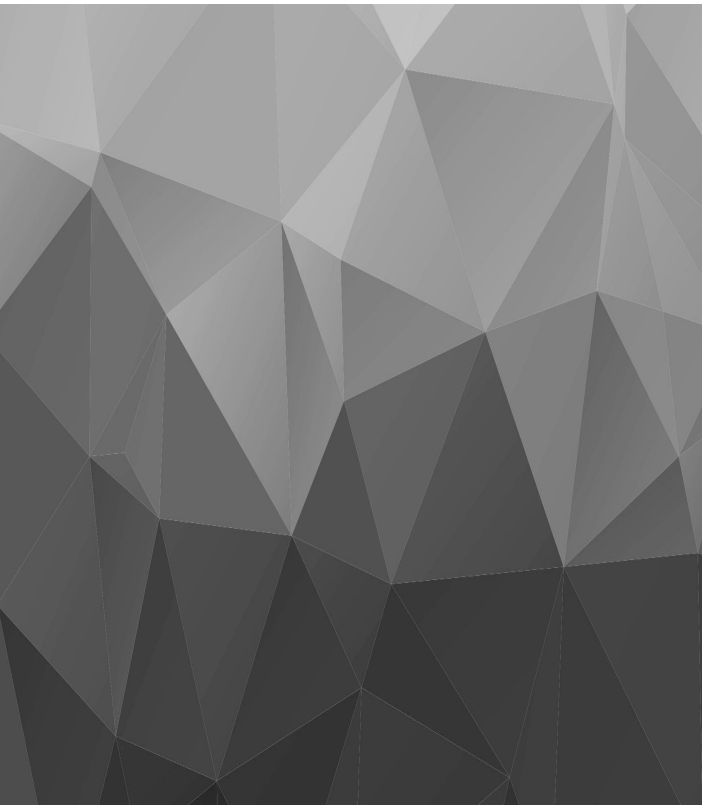
VIPER的优势

相对于MVC，VIPER具备几大优势。首先，它提供了更多的抽象层。Presenter包括了将业务逻辑(business logic)和视图逻辑(view logic)联结在一起的Presentation Logic。 Interactor处理纯粹的数据操作和验证，包括对后端服务的调用来控制状态变化等，如登录和请求出行。最后，Router发起状态切换，例如将用户从主界面(Home Screen)带到确认界面。其次，Presenter和Interactor都是传统的object，可以用简单的unit test测试。

VIPER的问题

依赖于视图逻辑驱动，换句话说它依赖于Interactor控制状态变化来实施的业务逻辑实际上始终要通过Presenter层，从而仍然将业务逻辑暴露在别的层面。最后，因为它的视图树和业务逻辑树紧密耦合，要实现一个只有业务逻辑或者只有视图逻辑的模块会非常困难。

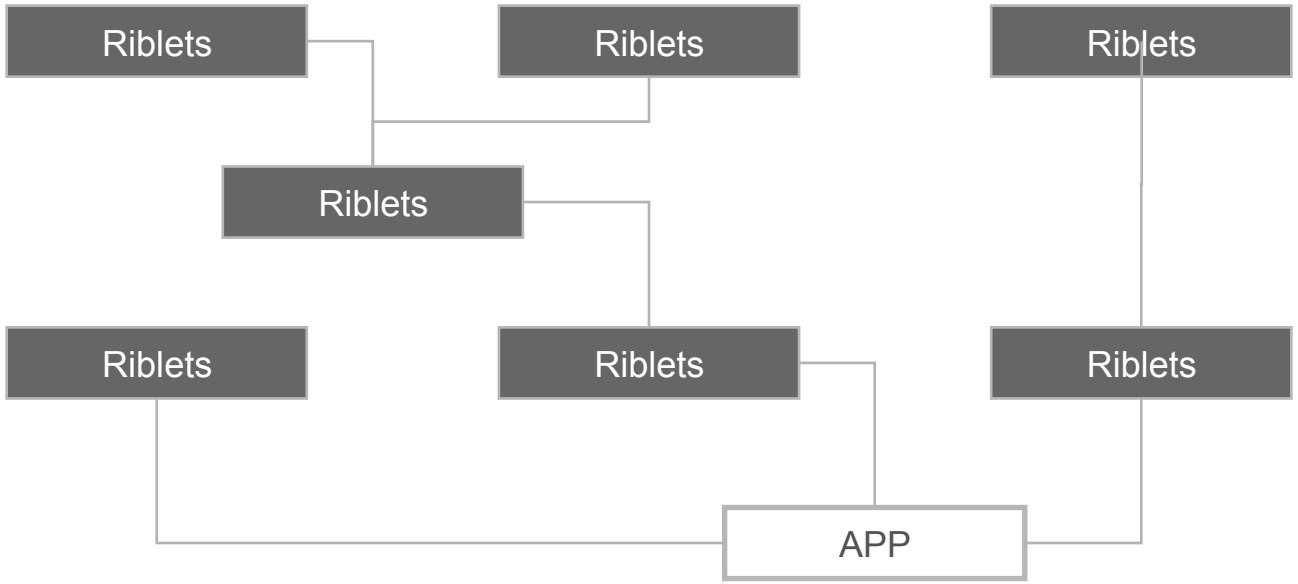
RIBLETS



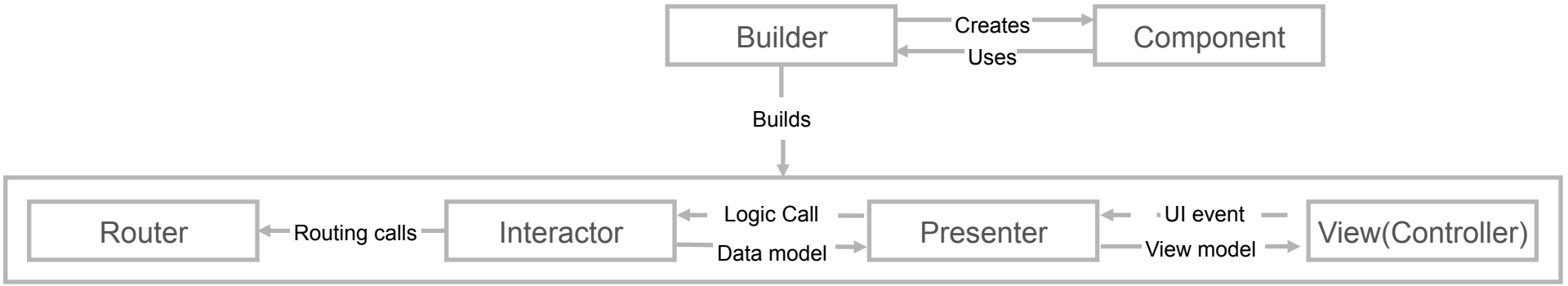
满足适应大规模开发并具备清晰的模块化平台的需要

Riblets

在使用Riblets架构的App中所有的逻辑都划分成小块且可独立测试的模块，每个模块只具备一个单一目的和责任。我们把这些模块成为Riblets，而整个应用则建立在一个由Riblets构成的树状结构上。



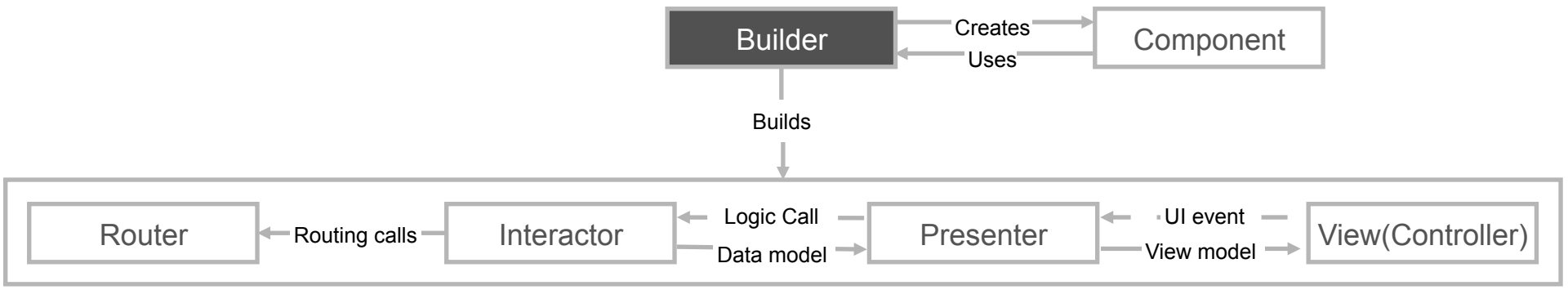
Riblets



一个Riblets由这几部分组成：Router, Interactor, Builder, 以及可选的Presenter和View。其中Router和Interactor处理业务逻辑，而Presenter和View负责视图和交互逻辑

路由的选择由业务逻辑决定而非视图逻辑。这也意味着程序是由信息和决策来驱动，而不是在表现层由界面来驱动（举例来说，就不是因为你点了某个按钮就从一个 ViewController跳到另一个ViewController，这是视图交互逻辑驱动；而是因为在业务逻辑上你从一个状态切换到另一个状态，例如从热身状态切换到了跑步进行中状态，因此才带来界面的切换）

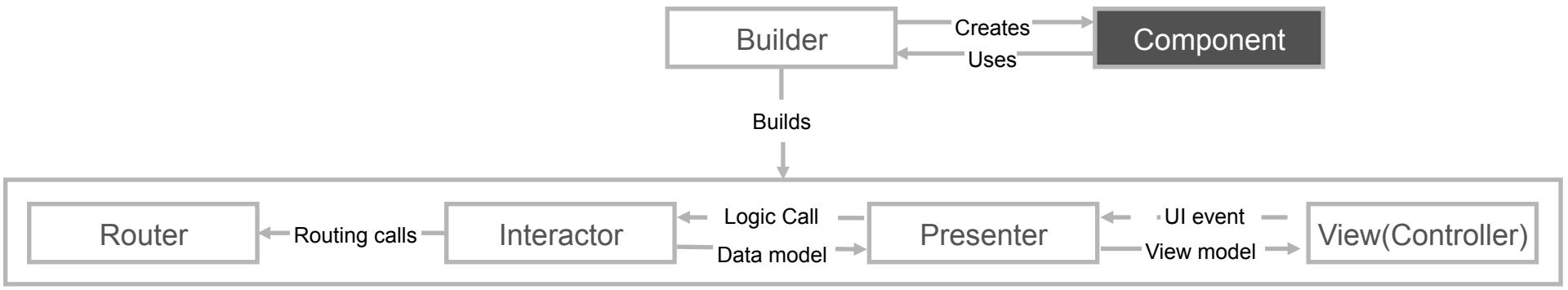
Riblets



Builder

Builder负责初始化所有的Ribletss组件，如Router/Interactor以及ViewController等都在Builder中初始化，并在其中定义依赖关系。

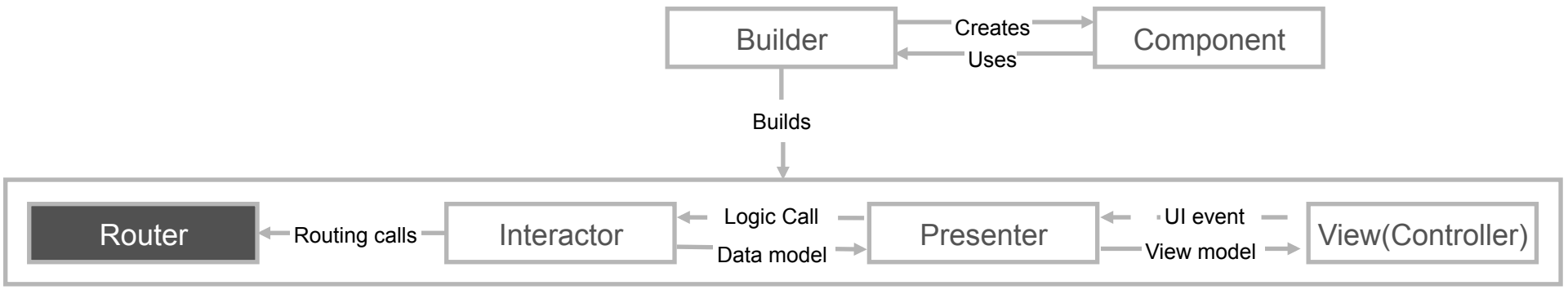
Riblets



Component

Component获取并初始化一个Riblets的dependency。这包括了各种服务（例如获得当前地址的服务，需要的网络服务或数据流，以及其它任何需要从其它部分传下来的非Riblets对象）。把它和合适的网络事件绑定，并inject到Interactor中

Riblets

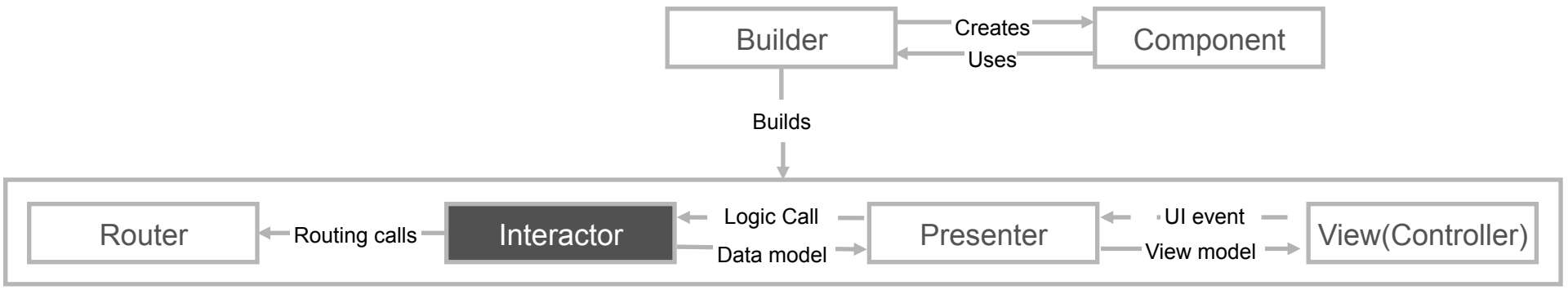


Router

Router的作用是在程序的Riblets Tree中通过不同逻辑来attach/detach相关的Ribletss. 所作出的决定又传送到Interactor. Router同时也会在状态切换的时候通过actiave/deactivate相关的Interactor来控制其lifecycle. 通常来说，Router包括两部分的商业逻辑:

1. Attach/Detach router的helper methods
2. 状态切换逻辑，用于决定多个子节点（child riblets）的状态

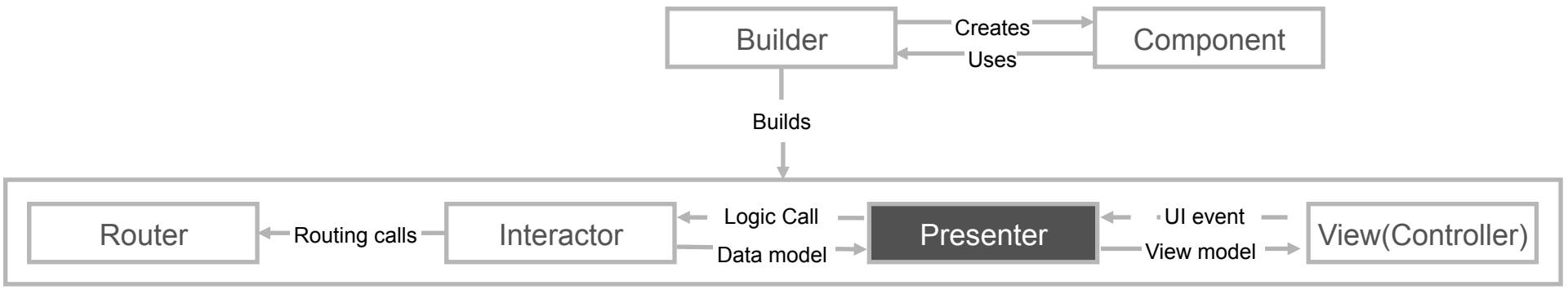
Riblets



Interactor

Interactors执行商业逻辑，包括：对对应服务发出请求以启动某种行为，例如发出叫车请求；抓取数据；决定要转换到哪种状态。比如根节点的Interactor (root interactor)注意到用户的认证token缺失，它会发出请求给router要求转换到欢迎界面

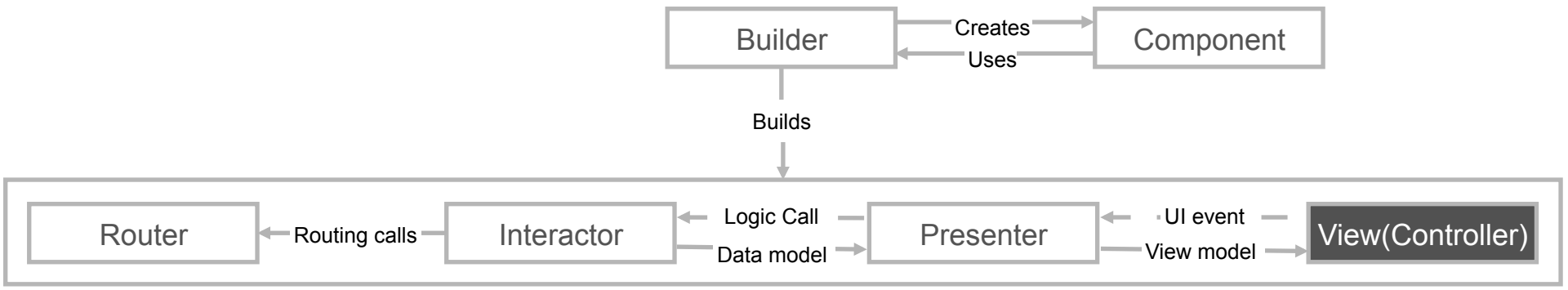
Riblets



Presenter

Presenter管理Interactors和Views之间的通信，从Interactors到Views，Presenter将逻辑数据转化为Views能够显示的对象。从Views到Interactors，Presenter将用户操作的事件触发Interactors中的事件，例如点击按钮选择产品等。

Riblets



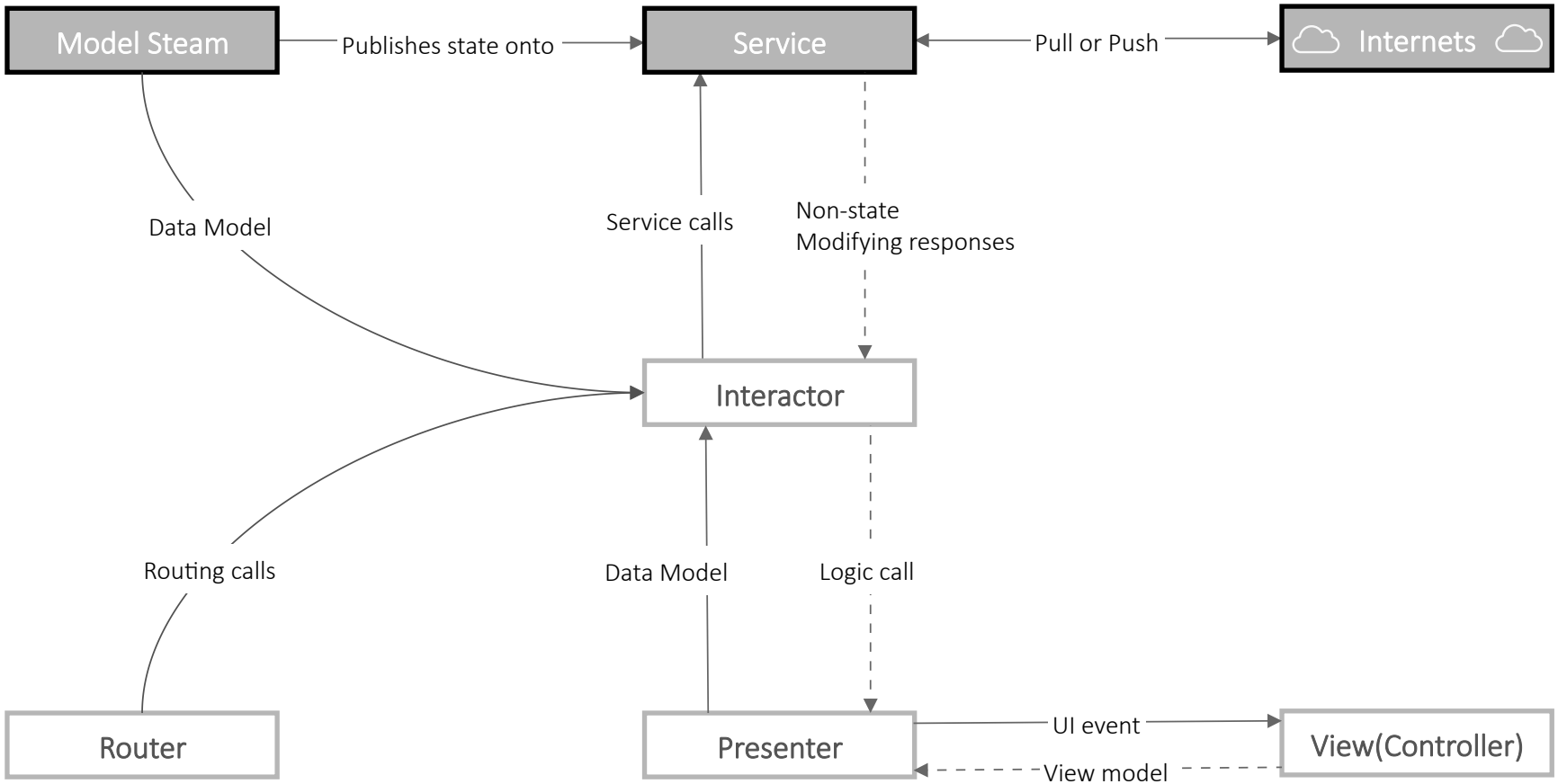
Views(Controller)

View创建并更新UI，包括初始化和布局各种UI组件，处理用户交互，填充数据和动画等（一般来说View在iOS中对应的就是一个ViewController）。对于Product Selection，这包括了价格数据和车辆类型。

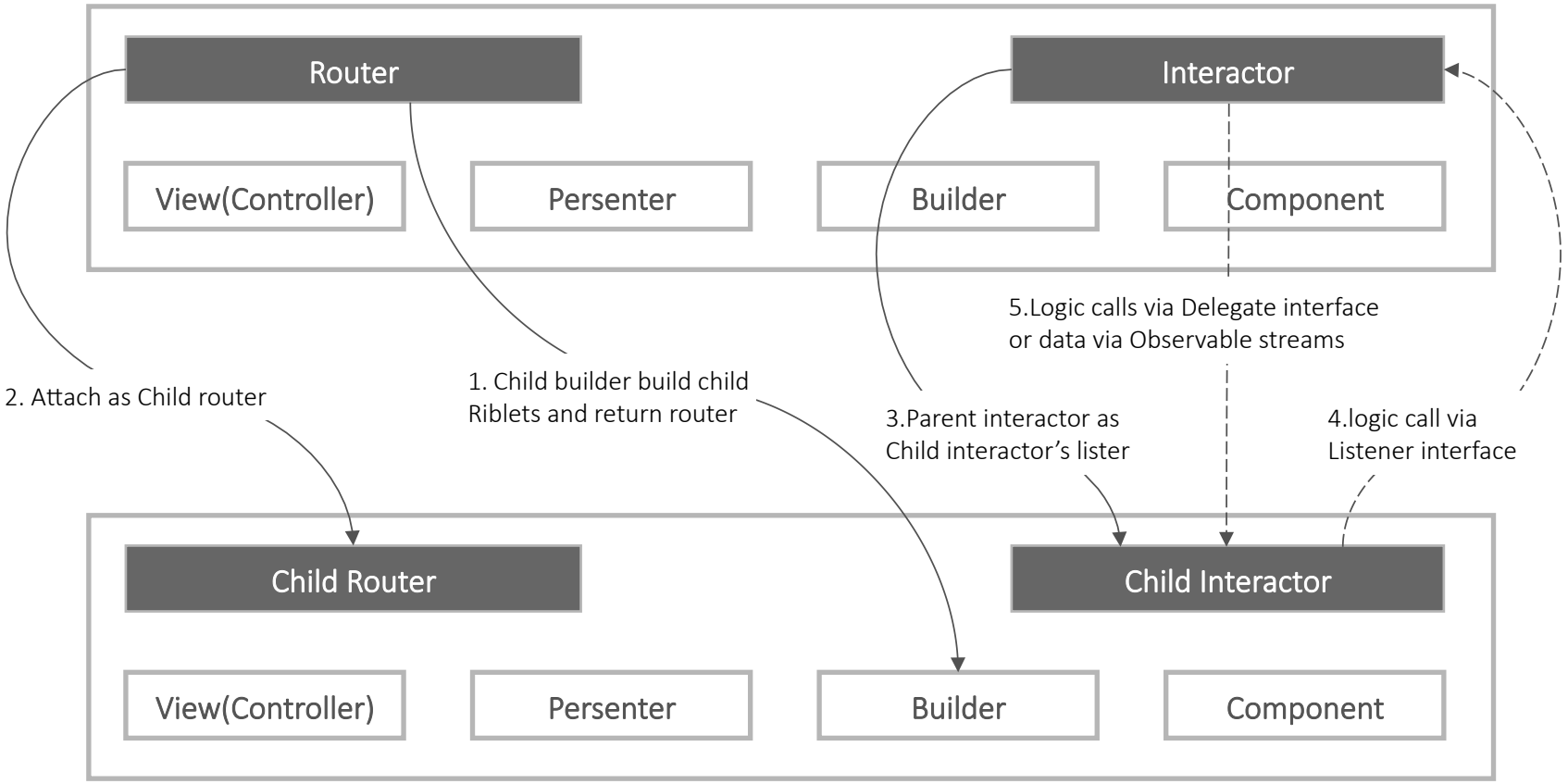
Riblets中Presenter和View的对应关系多对多，并且Presenter和View都可以为零

允许了业务逻辑树的结构和深度和视图树不必一致，从而简化屏幕的切换。

Data flow within a Riblet



Communication between Riblets



分离的功能模块

架构是无关具体平台，iOS和Android开发者能轻松了解对方的开发情况，从而互相学习，避免错误，

如何建立未来开发模式的正确方向

M

就算用MVC也能架构

Model

给ViewController提供数据
给ViewController存储数据提供接口
提供经过抽象的业务基本组件，供Controller调度

Controller

管理View的生命周期
负责生成所有的View实例，并放入View容器中
监听来自View与业务有关的事件，完成对应事件的业务

View

响应与业务无关的事件，并因此引发动画效果
处理点击反馈
界面元素表达

M

- 什么才是好的架构
- MVC
- MVVM
- VIPER
- Riblets


WOTA
51CTO
Thank you !