АНАЛИЗ И СРАВНЕНИЕ АРХИТЕКТУРНЫХ ШАБЛОНОВ ПРОЕКТИРОВАНИЯ, ИСПОЛЬЗУЕМЫХ ПРИ РАЗРАБОТКЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ ПЛАТФОРМЫ iOS

Булыга И.М.

Булыга Игорь Михайлович - разработчик программного обеспечения для платформы iOS, Компания Booking.com, г. Амстердам, Нидерланды

Аннотация: *в статье анализируется и сравнивается различные шаблоны проектирования ПО при разработке под мобильную платформу iOS.*
Ключевые слова: *паттерн, ios, mvc, mvp, mvvm.*

В данной статье будут рассмотрены различные архитектурные шаблоны (паттерн, по-английски - patterns)проектирования, которые используются при разработки програмного обеспечения (сокращенно - ПО) для платформы iOS. Для начала будут освещены семейства архитектурных шаблонов проектирования, а затем конкретные реализации, такие как модель представление контроллер (по-английски – Model View Controller, сокращённо – MVC), модель представление ведущий (по-английски Model View Presenter, сокращенно – MVP), модель представление модель-представления (по-английски – Model View ViewModel, сокращенно – MVVM).

Семейства архитектурных шаблонов проектирования

Существует множество семейств архитектурных шаблонов проектирования, которые нашли свое применение в различных системах ПО. В рамках этой статьи мы подробнее остановимся на некоторых из них:

- Многоуровневые шаблоны (по-английски – Layered Architecture);
- Семейство шаблонов модель представление X, где в роли X могут быть представлены такие сущности, как контроллер, ведущий или модель представления.

Все вышеперечисленные семейства архитектурных шаблонов решают задачи по упрощению систем, их гибкости, при изменении требований к системе, масштабирования и поддержки. Для различных систем используются различные семейства, но они с легкостью могут быть скомбинированы при разработке одной системы.

Многоуровневые шаблоны

При разработке ПО для мобильной платформы iOS, данный подход используется повсеместно. Система раскладывается на слои, которые определяются техническими особенностями системы. Ограничений по количеству слоев и их типу не предусмотрено. Однако чаще всего используется архитектура, состоящая из трех слоев: слой представления, слой бизнес-логики, слой данных.

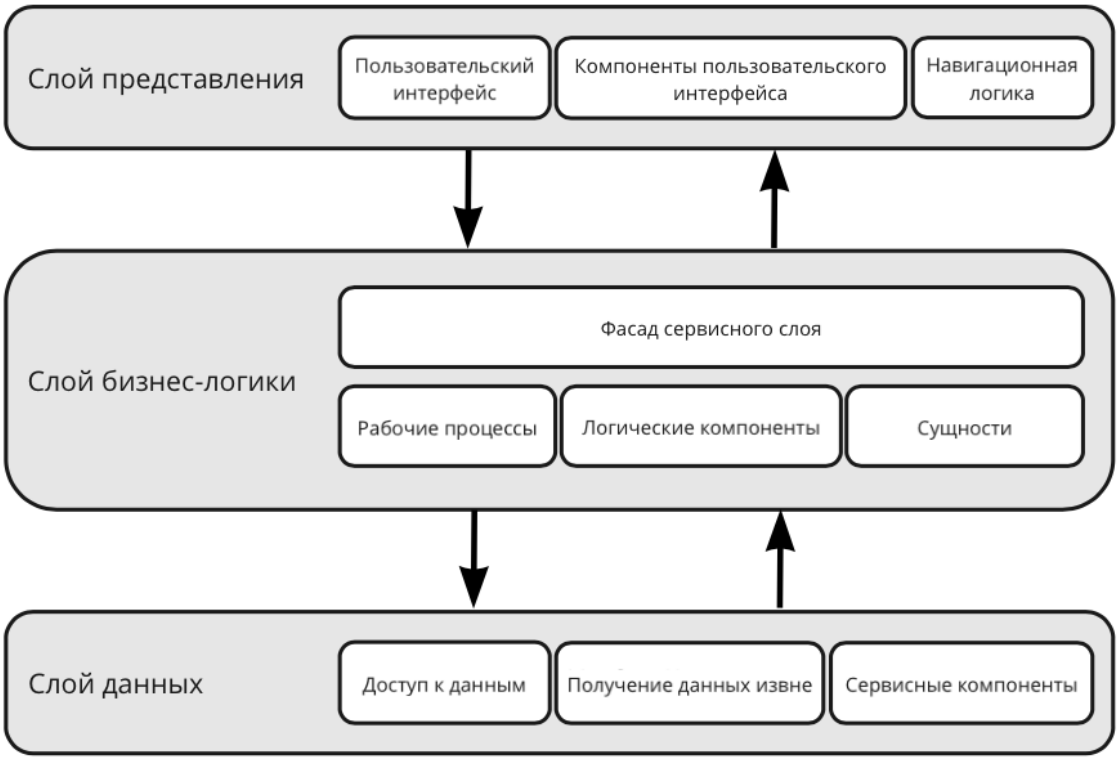

Рис. 1. Типовая схема многоуровневой архитектуры мобильного приложения

На рис. 1 приведена типовая схема проекта, состоящего их трех слоев, рассмотрим каждый слой подробнее.

- Слой представления – на этом слое располагаются компоненты, которые непосредственно предоставляют визуальное отображение данных, поступивших на этот слой из слоя бизнес-логики. Обрабатывают пользовательский ввод, например нажатия на кнопки, ввод с клавиатуры, а затем предают эти данные на более низкий слой бизнес-логики.
- Слой бизнес-логики – на этом слое располагаются компоненты, которые обеспечивают слой представления необходимыми данными для формирования отображения, а также реагируют на пользовательский ввод, и в случае необходимости запускают рабочие процессы, которые получают или отдают данные на более низкий слой данных.
- Слой данных – на этом слое располагаются компоненты, которые отвечают за хранение данных, получение данных из внешних источников, например сети интернет.

При использовании данного подхода разработчики легко могут отделить различные компоненты, и обеспечить слабую связанность между компонентами системы, при условии, что слои могут принимать и отдавать данные только слоям, находящимся на один уровень вверх или вниз.

Данный подход легко комбинируется с другими семействами шаблонов проектирования, например слой представления, может быть реализован с помощью семейства шаблонов модель представление X. А слой

Модель представление контроллер

Данный архитектурный шаблон при разработке мобильных приложений для платформы iOS состоит из трех компонентов.

1. модель – отвечает за данные, чаще всего она ограниченна доменной областью, которая содержит в себе ту или иную бизнес логику.
2. представление – отвечает за визуальную часть приложения, обработку пользовательского ввода, навигацию.
3. контроллер – обрабатывает запросы, манипулирует данными из модели и совершает другие операции.

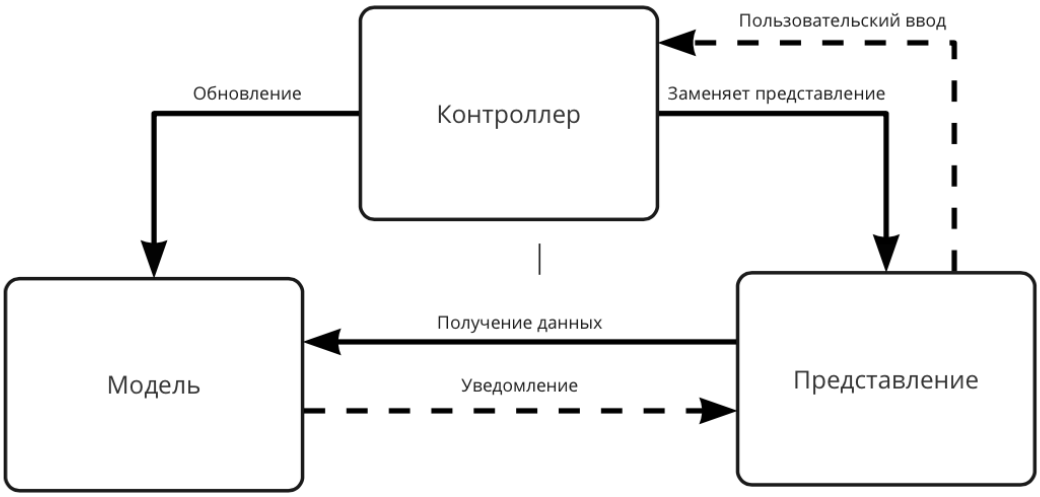

Рис. 2. Схема работы шаблона модель контроллер представление

На рисунке 2 представлена схема работы шаблона модель представление контроллер и как компоненты взаимодействуют друг с другом. Как видно из схемы, все компоненты тесно связаны друг с другом, и представление полностью перерисовывается, когда наши данные изменяются, или контроллер реагирует на пользовательский ввод. Такой подход может быть реализован на платформе iOS, но добиться этого будет очень тяжело и кроме того, мы не сможем переиспользовать компоненты и столкнемся с рядом ограничений, наложенных платформой.

Компания Apple, немного изменила данный подход для своих платформ.

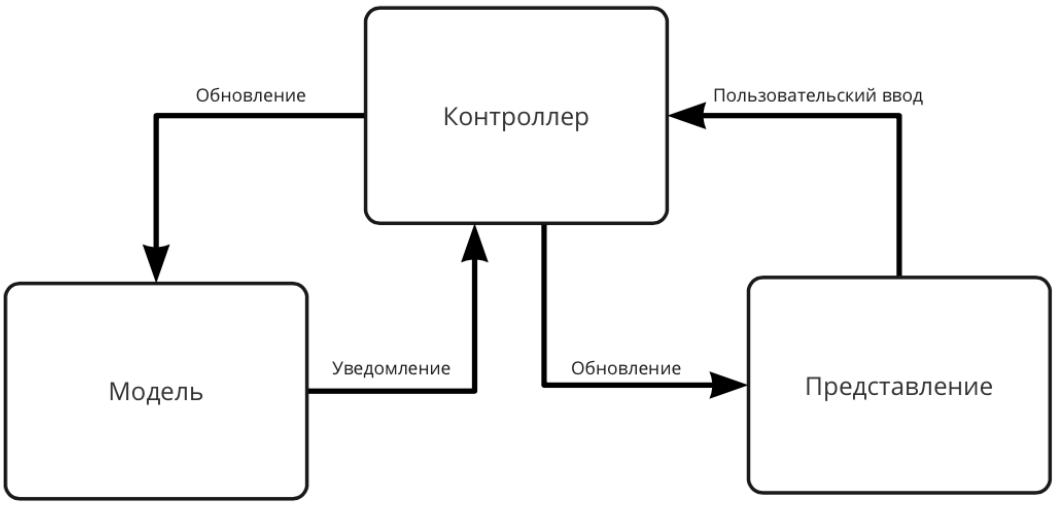

Рис. 3. Схема работы шаблона модель представление контроллер, реализованная компанией Apple

Как видно из схемы, представленной на рисунке 3, представление передает событие о пользовательском вводе в контроллер, тот обрабатывает этот пользовательский ввод и обновляет модель, модель реагирует на обновление и уведомляет об этом контроллер, который в свою очередь обновляет представление. Таким образом обеспечивается слабая связанность компонентов и разделение ответственности между компонентами системы, что позволяет их пере использовать.

Однако, в реализации данного подхода была допущена критическая ошибка, контроллер и представление были тесно связаны друг с другом, и контроллер помимо обработки пользовательского

ввода стал отвечать и за представление, что привело к его разрастанию, невозможности пере использования и масштабирования. Все дело в том, что компания Apple, назвала базовый класс для реализации экрана ViewController и добавила ему ненужные ответственности по обработке пользовательского ввода и управление жизненным циклом класса View. Таким образом, она связала представление и контроллер.

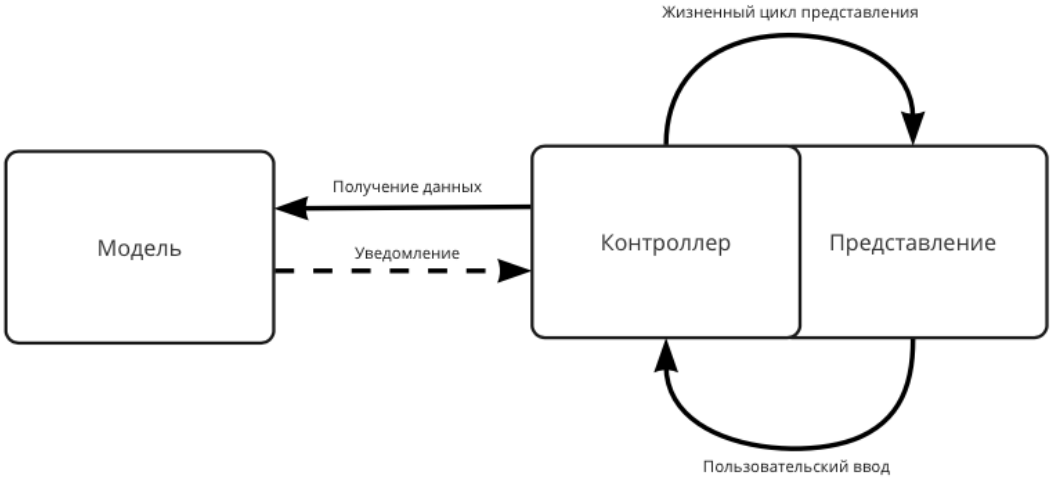

Рис. 4. Реализация шаблона модель представление контроллер, компанией Apple

Но у такого подхода есть и свои плюсы, например, очень быстрая скорость разработки.

Модель представление ведущий.

Основное отличие шаблона модель представление ведущий от модель представление контроллер, разработанного компанией Apple, в том, что роль контроллера заменил ведущий, и теперь классы View и ViewController, точно так же могут оставаться связанными, но теперь это не имеет значения, так как вся логика по обновление модели и представления вынесена в отдельный компонент – ведущий.

Как видно из схемы, представленной на рисунке 5, ведущий полностью заменяет контроллер. Весь пользовательский ввод, события жизненного цикла передаются от представления, в нашем случае от View и ViewController, ведущему, который обрабатывает его и в случае необходимости обновляет модель, а модель в ответ уведомляет ведущего об изменениях, которые в дальнейшем будут переданы в представление и отобразятся на экране.

Можно заметить, что данный шаблон очень сильно напоминает шаблон модель представление контроллер, так оно и есть, но с одним очень большим отличием – вся логика скрыта за ведущим. Таким образом, это позволяет нам обеспечить слабую связанность компонентов между собой, улучшить тестируемость кода и дальнейшее переиспользование и расширение.

Но также это накладывает и свои ограничения и добавляет очень много однотипного кода, для общения представления и ведущего. Также необходимо решать проблему навигации и со сборкой модулей.

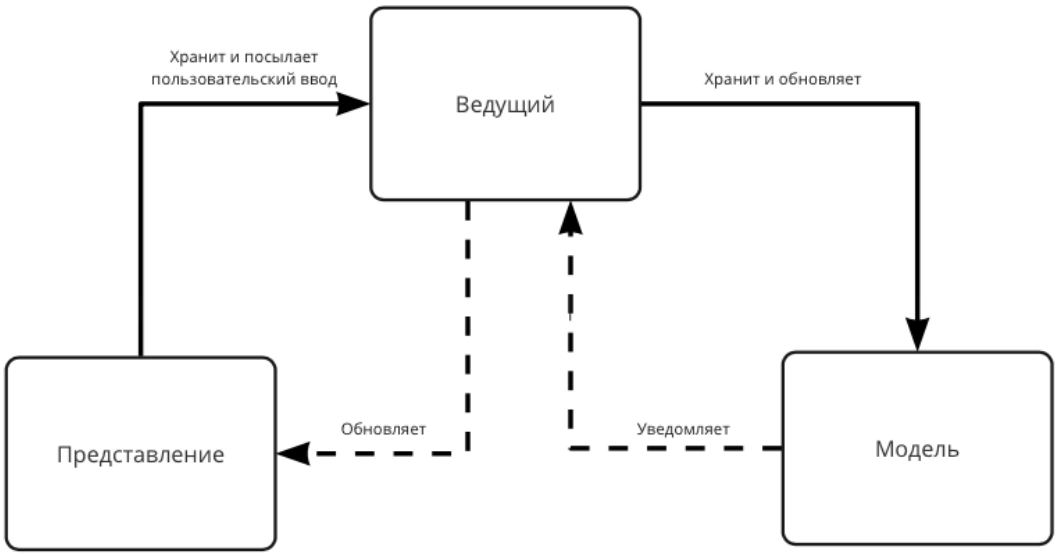

Рис. 5. Схема работы шаблона модель представление ведущий

Модель представление модель-представления

Данный шаблон выглядит самым перспективным из всего списка модель представление X, так как он решает практически все проблемы, с которыми сталкиваются модель представление контроллер и модель представление ведущий. В данном шаблоне точно так же представлены компоненты представления и модели, но компонент модель представления появляется впервые. Этот компонент независим от пользовательских компонентов и оперирует только данными и хранит состояние представления. Так же модель-представления изменяет модель и реагирует на ее обновления.

Как видно из схемы, представленной на рисунке 6, связь между представлением и модель-представления осуществляется с помощью двухстороннего связывания. Таким образом представление ничего не знает о модели-представления, что обеспечивает слабую связанность компонентов и позволяет пере использовать модель-представления в других системах и компонентах. Такой механизм позволяет моментально обновлять представление, если состояние модели-представления изменилось.

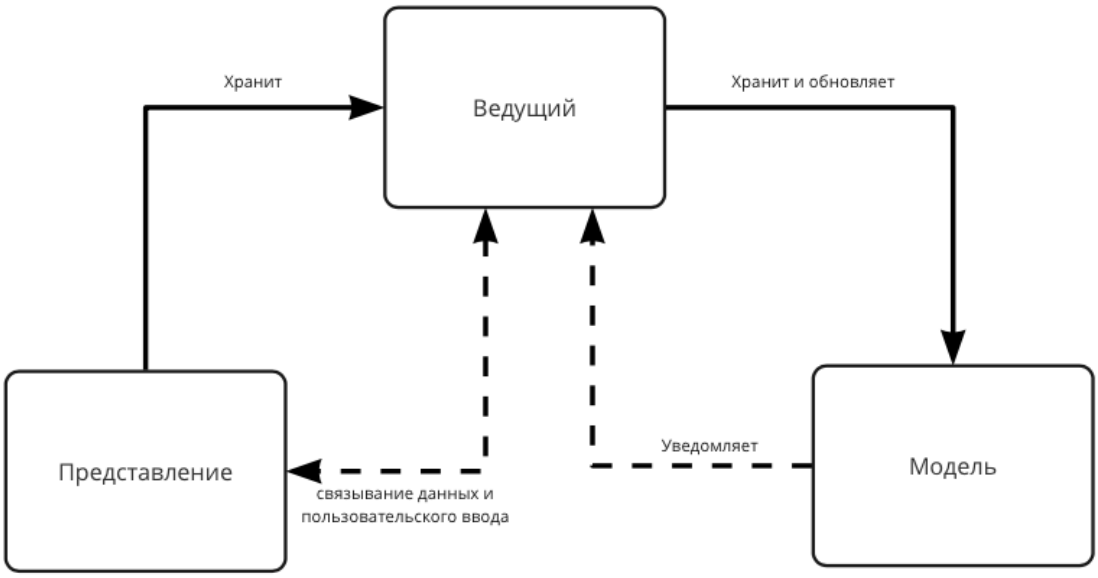

Рис. 6. Схема работы шаблона модель представление модель-представления

Связывание

Данный механизм позволяет связать те или иные данные и действия между собой. Реализовать его можно несколькими способами:

- KVO и KVC – key-value-observing и key value coding системный механизм наблюдения и изменения данных
- Функциональное реактивное программирование – данный подход реализован с помощью таких библиотек как RxSwift, Combine.

С помощью связывания убирается проблема написания однотипного кода для обновления представления, которая присутствовала в шаблоне модель представление ведущий. Также мы в любой момент можем получить состояние нашего представления, так как оно представлено в модели-представления.

Однако, использования связывания накладывает свои ограничения, такие как сложность отладки. Потому что не всегда понятно, откуда пришло событие обновления данных, и тяжело пробраться через стэк вызовов.

Заключение

В статье я разобрал основные архитектурные паттерны, используемые для разработки мобильных приложений для платформы iOS. Также я попытался сравнить их и выделить плюсы и минусы каждой из них. Существует неограниченное количество архитектурных паттернов, и каждый из них решает свою задачу.

Список литературы

1. *Ричардс Марк.* Паттерны проектирования программного обеспечения. [Электронный ресурс], 2015. Режим доступа: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ (дата обращения: 21.12.2021).
2. Архивная документация компании Apple, Model-View-Controller. [Электронный ресурс], 2018. Режим доступа: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html/ (дата обращения: 21.12.2021).
3. Microsoft Application Architecture Guide, 2nd Edition. [Электронный ресурс], 2009. Режим доступа: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658117(v=pandp.10)?redirectedfrom=MSDN/ (дата обращения: 21.12.2021).