Паттерны для новичков: MVC vs MVP vs MVVM

C#*, .NET*

Добрый день, уважаемые коллеги. В этой статье я бы хотел рассказать о своем аналитическом понимании различий паттернов MVC, MVP и MVVM. Написать эту статью меня побудило желание разобраться в современных подходах при разработке крупного программного обеспечения и соответствующих архитектурных особенностях. На текущем этапе своей карьерной лестницы я не являюсь непосредственным разработчиком, поэтому статья может содержать ошибки, неточности и недопонимание. Заинтригованы, как аналитики видят, что делают программисты и архитекторы? Тогда добро пожаловать под кат.

Ссылки

Первое, с чего я бы хотел начать — это ссылки на внешние материалы, которыми я руководствовался в процессе написания этой статьи:

- ru.wikipedia.org/wiki/Model-View-Controller
- ru.wikipedia.org/wiki/Model-View-Presenter
- ru.wikipedia.org/wiki/MVVM
- outcoldman.com/ru/blog/show/184
- rsdn.ru/article/patterns/ModelViewPresenter.xml

Введение

Во времена, когда солнце светило ярче, а трава была зеленее, на тот момент команда студентов, как автор этой статьи, разрабатывали программное обеспечение, писав сотни строк кода непосредственно в интерфейсе продукта. Иногда использовались сервисы и менеджеры для работы с данными и тогда решение получалось с использованием паттерна Document-View. Поддержка такого кода требовала колоссальных затрат, т. к. нового разработчика надо обучить (рассказать), какой код за что в продукте отвечает, и ни о каком модульном тестировании и речи не было. Команда разработки — это 4 человека, которые сидят в одной комнате.
Прошло время, менялась работа. Разрабатываемые приложения становились больше и сложнее, из одной сплоченной команды разработчиков стало много разных команд разработчиков, архитекторов, юзабилистов, дизайнеров и PMов. Теперь каждый ответственен за свою область: GUI, бизнес-логика, компоненты. Появился отдел анализа, тестирования, архитектуры. Стоимость разработки ПО возросла в сотни и даже тысячи раз. Такой подход к разработке требует наличие стойкой архитектуры, которая бы синхронизировала разные функциональные области продукта между собой.

Паттерны

Учитывая цель уменьшения трудозатрат на разработку сложного программного

обеспечения, предположим, что необходимо использовать готовые унифицированные решения. Ведь шаблонность действий облегчает коммуникацию между разработчиками, позволяет ссылаться на известные конструкции, снижает количество ошибок.
По словам Википедии, паттерн (англ. design pattern) — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Начнем с первого главного – Model-View-Controller. MVC — это фундаментальный паттерн, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам.

Впервые паттерн MVC появился в языке SmallTalk. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес логики, а бизнес логику от данных. Таким образом, в классическом варианте, MVC состоит из трех частей, которые и дали ему название. Рассмотрим их:

Модель

Под Моделью, обычно понимается часть содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель.

Модель обладает следующими признаками:

- Модель — это бизнес-логика приложения;
- Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
- Для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);
- Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения;

Представление (View)

В обязанности Представления входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Представление обладает следующими признаками:

- В представлении реализуется отображение данных, которые получаются от модели любым способом;

- *В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.*

Примеры представления: HTML-страница, WPF форма, Windows Form.

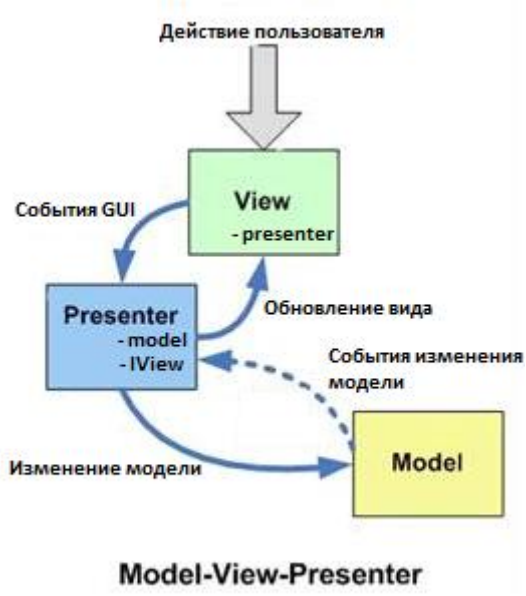
Различия MVP & MVVM & MVP

Наиболее распространенные виды MVC-паттерна, это:

- Model-View-Controller
- Model-View-Presenter
- Model-View-View Model

Рассмотрим и сравним каждый из них.

Model-View-Presenter



Model-View-Presenter

Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.

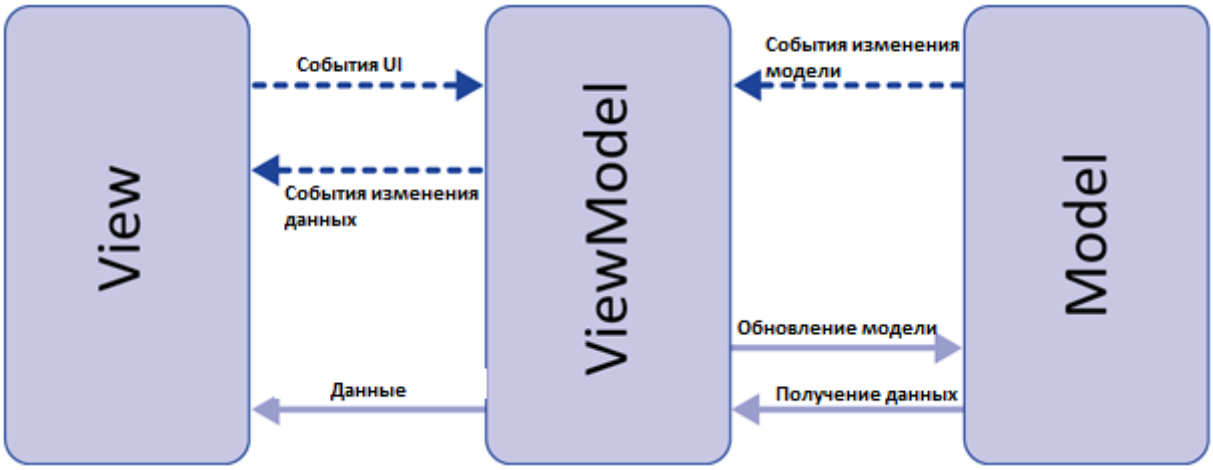
Признаки презентера:

- Двухсторонняя коммуникация с представлением;
- Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
- Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
- *Один экземпляр презентера связан с одним отображением.*

Реализация:
Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, **IView**.ShowErrorMessage(string msg)). Презентер должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе.
Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и практически никогда не обрабатываются логикой представления (в т.ч. создания других представлений).

Пример использования: **Windows Forms.**

Model-View-View Model



Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.

Признаки View-модели:

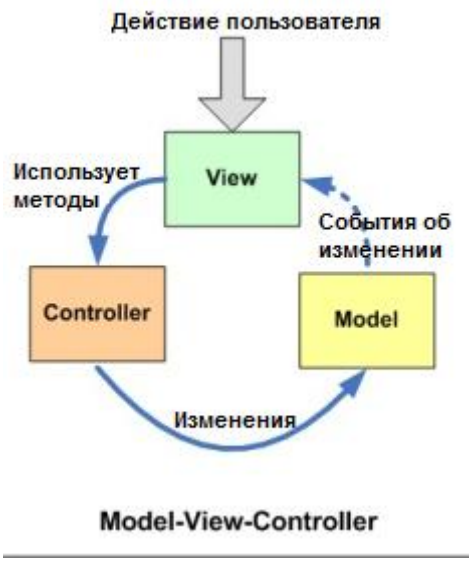
- Двухсторонняя коммуникация с представлением;
- View-модель — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели
- View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)
- *Один экземпляр View-модели связан с одним отображением.*

Реализация:
При использовании этого паттерна, представление не реализует соответствующий интерфейс (IView).
Представление должно иметь ссылку на источник данных (DataContex), которым в данном случае является View-модель. Элементы представления связаны (Bind) с

соответствующими свойствами и событиями View-модели.
В свою очередь, View-модель реализует специальный интерфейс, который используется для автоматического обновления элементов представления. Примером такого интерфейса в WPF может быть INotifyPropertyChanged.

Пример использования: **WPF**

Model-View-Controller



Model-View-Controller

Основная идея этого паттерна в том, что и контроллер и представление зависят от модели, но модель никак не зависит от этих двух компонент.

Признаки контроллера

- Контроллер определяет, какие представление должно быть отображено в данный момент;
- События представления могут повлиять только на контроллер.контроллер может повлиять на модель и определить другое представление.
- Возможно несколько представлений только для одного контроллера;

Реализация:
Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя Модель, посредством вызова соответствующего метода. После изменения Модель использует событие о том что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

Пример использования: **MVC ASP.NET**

Резюме

Реализация MVVM и MVP-паттернов, на первый взгляд, выглядит достаточно простой схожей. Однако, для MVVM связывание представления с View-моделью осуществляется автоматически, а для MVP — необходимо программировать
MVC, по-видимому, имеет больше возможностей по управлению представлением.

Общие правила выбора паттерна

MVVM

- Используется в ситуации, когда возможно связывание данных без необходимости ввода специальных интерфейсов представления (т.е. отсутствует необходимость реализовывать IView);
- Частым примером является технология WPF.

MVP

- Используется в ситуации, когда невозможно связывание данных (нельзя использовать Binding);
- Частым примером может быть использование Windows Forms.

MVC

- Используется в ситуации, когда связь между представление и другими частями приложения невозможна (и Вы не можете использовать MVVM или MVP);
- Частым примером использования может служить ASP.NET MVC.

Заключение

В заключении, автор этой статьи хотел бы отметить, что строго придерживаться только одному паттерну — не всегда лучший выбор. Например, представьте, что Вы хотели бы использовать MVVM для разработки приложений с использованием Windows Forms через свойство контролов Bindings. Ваша цель — это отделить представление от бизнес логики и логики, которая их связывает. Приложение должно быть легко тестируемым и поддерживаемым, а для аналитиков — понятным (ведь на вопрос «в чем измеряется работа жесткого диска» существует единственный правильный ответ — в Джоулях (абстрактный пример Модели -> Представления)).

Большое спасибо за уделенное время, приятного чтения!