Презентационни шаблони

/гл. ас. Димитър Благоев, гл. ас. Мая Стоева/

Съдържание

1. Въведение
2. Терминология
3. Model-View-Controller (MVC = Модел-Изглед-Контролер)
4. PureModel-View-Controller (PureMVC = Чист Модел-Изглед-Контролер)
5. Model-View-ViewMode (MVVM = Модел-Изглед-Изглед-на-Модела)
6. Model-View-Presenter (MVP)

Въведение

Повечето приложения се състоят от три слоя:

- Изглед, който е видим за потребителя;
- Данни, които са представени и могат да се манипулират;
- Логика, отговорна за работата на приложението.

Презентационните шаблони се опитват да обособят тези слоеве.

Те целят максимална гъвкавост на разработваното приложение.

В идеалния случай поддържат мултиплатформеност.

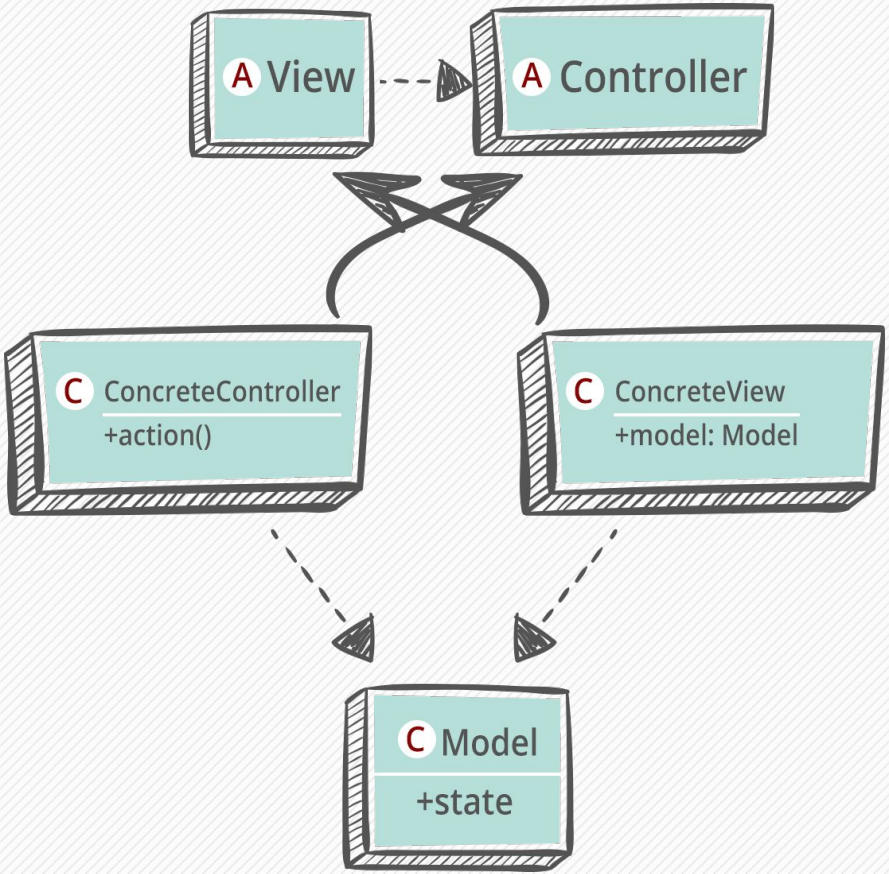
Терминология

- Класовете, съдържащи данни често се наричат модели.
- Моделът обикновено не съдържа логика.
- Шаблоните за дизайн се различават по това как са използвани моделите.
- Някои шаблони използват контролер, за да свържат изглед и модел.
- Други шаблони – презентатор – за активна промяна на изгледа с модела
- Съществуват и високо-динамични модели, които обединяват двата подхода, например MVVM.

Model-View-Controller

- В програмирането Модел-Изглед-Контролер (Model-View-Controller или MVC) е архитектурен шаблон за дизайн, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение.
- ***Контролерът*** *свързва модел и изглед.*
- ***Изгледът*** *използва модел, за да генерира изходната репрезентация.*
- ***Моделът*** *съдържа информация.*
- В **пасивното MVC** моделът е тих, но в **активното MVC**, при промяна моделът може да изпраща нотификации (напр. събития). Тези събития на свой ред е възможно да се използват от контролера или от изгледа.

Model-View-Controller

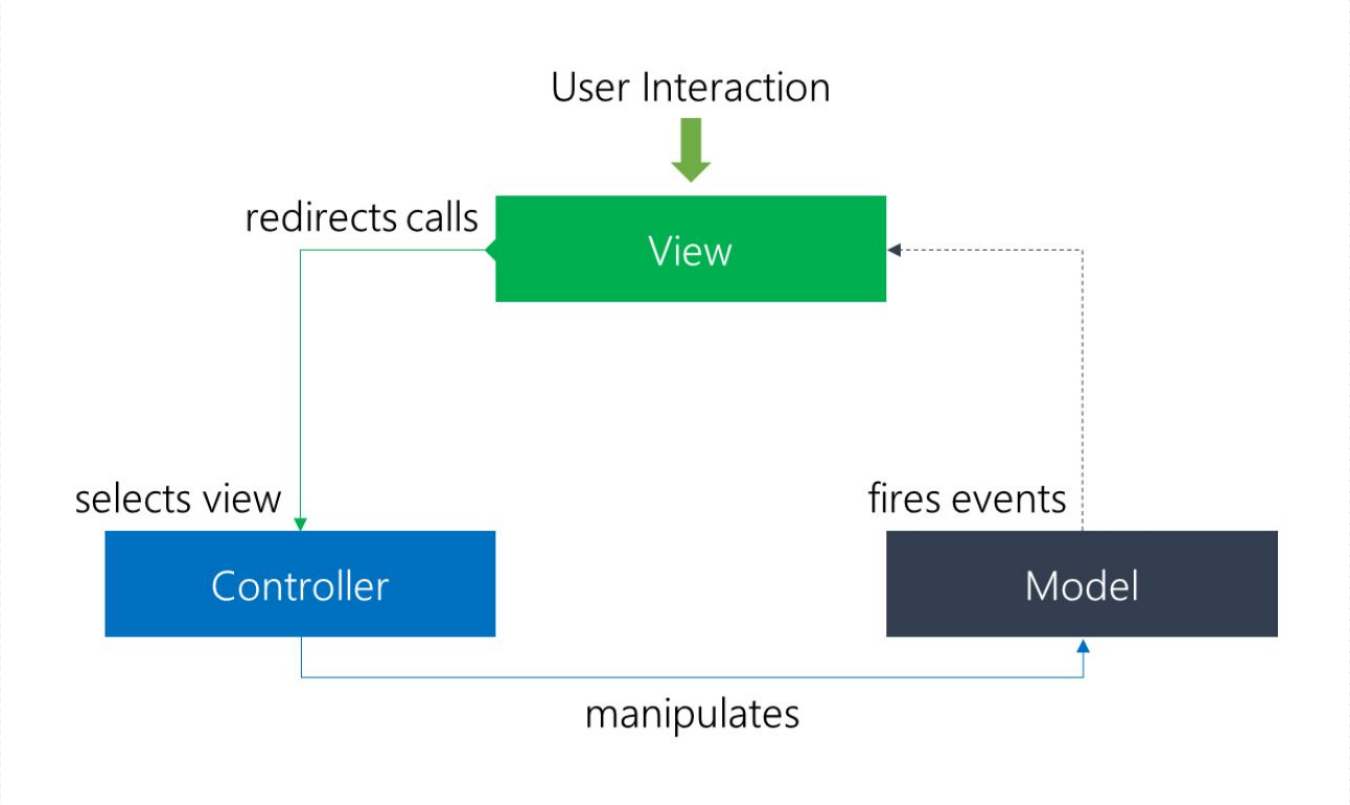


Model-View-Controller :: Забележки

Ползите от това разделение са:

- *Разделянето на отговорностите* (separation of concerns) е нужно за необвързване отделните части на продукта (decoupling).
- *Едно приложение може да има повече от една репрезентация* (клиент, конзола, уеб) и елементите на различните потребителските интерфейси е добре да се държат разделени от частите, които са общи за всички изгледи-репрезентации.
- Различни разработчици, с различни умения може да са *отговорни за различни аспекти на приложенията* (напр. дизайнери на интерфейси).

Model-View-Controller :: MVC в действие



Model-View-Controller :: Модел и изглед

Моделът:

- Репрезентира данни и правила, които управляват достъпа до него и обновлението му.
- Прости техники от моделирането в реалния свят могат да се прилагат, когато се дефинира моделът, защото той често е приближение на реалния свят.

Изгледът:

- Визуализира съдържанието на модела.
- Достъпва данните чрез модела и избира как да ги репрезентира.
- Отговорен за поддържане на консистентност, когато моделът се променя.
- Зависи от това какъв графичен интерфейс се прави и каква технология се използва.

Model-View-Controller :: Контролер

Контролерът транслира взаимодействията с изгледа в действия:

- Тези действия се изпълняват от модела. Например в самостоятелно графично приложения, потребителските взаимодействия може да са кликания на бутони, докато в уеб приложение те да са HTTP-заявки.
- Действията, изпълнявани от модела, включват активирането на бизнес процеси или промяната на състоянието на модела.
- На база на взаимодействията и резултата от действията на модела, контролерът отговаря като избере подходящ изглед.

Model-View-Controller :: Уеб имплементация

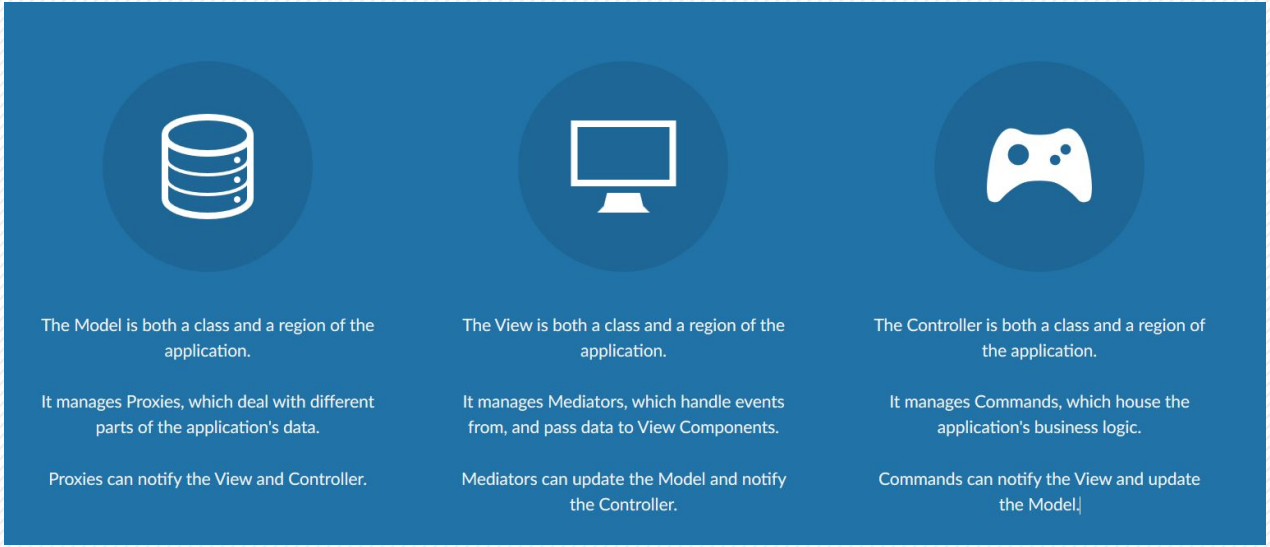
- Клас, (*наречен рутер*) е нужен, за да се интерпретират входящите заявки и да се насочат към съответния контролер.
- Извиква се отговарящият метод, (*наричан действие – action*).
- *Контролерът* може да обнови модела, на база на подадената заявка.
- След това се избира отговор, под формата на изглед, който може да е HTML.

Model-View-Controller :: Практически съвети

- Днес MVC се използва предимно в разработката на уеб приложения.
- Полулярни работни рамки съдържат много полезни базови имлпементации:
  - *ASP.NET MVC (.NET)*
  - *Rails (Ruby)*
  - *Spring (Java)*
  - *AngularJS (JavaScript)*
  - *CakePHP(PHP)*
- Обикновено писането от нулата на MVC-работна рамка не се препоръчва, (*освен за обучителни цели или при комерсиална нужда от това*).

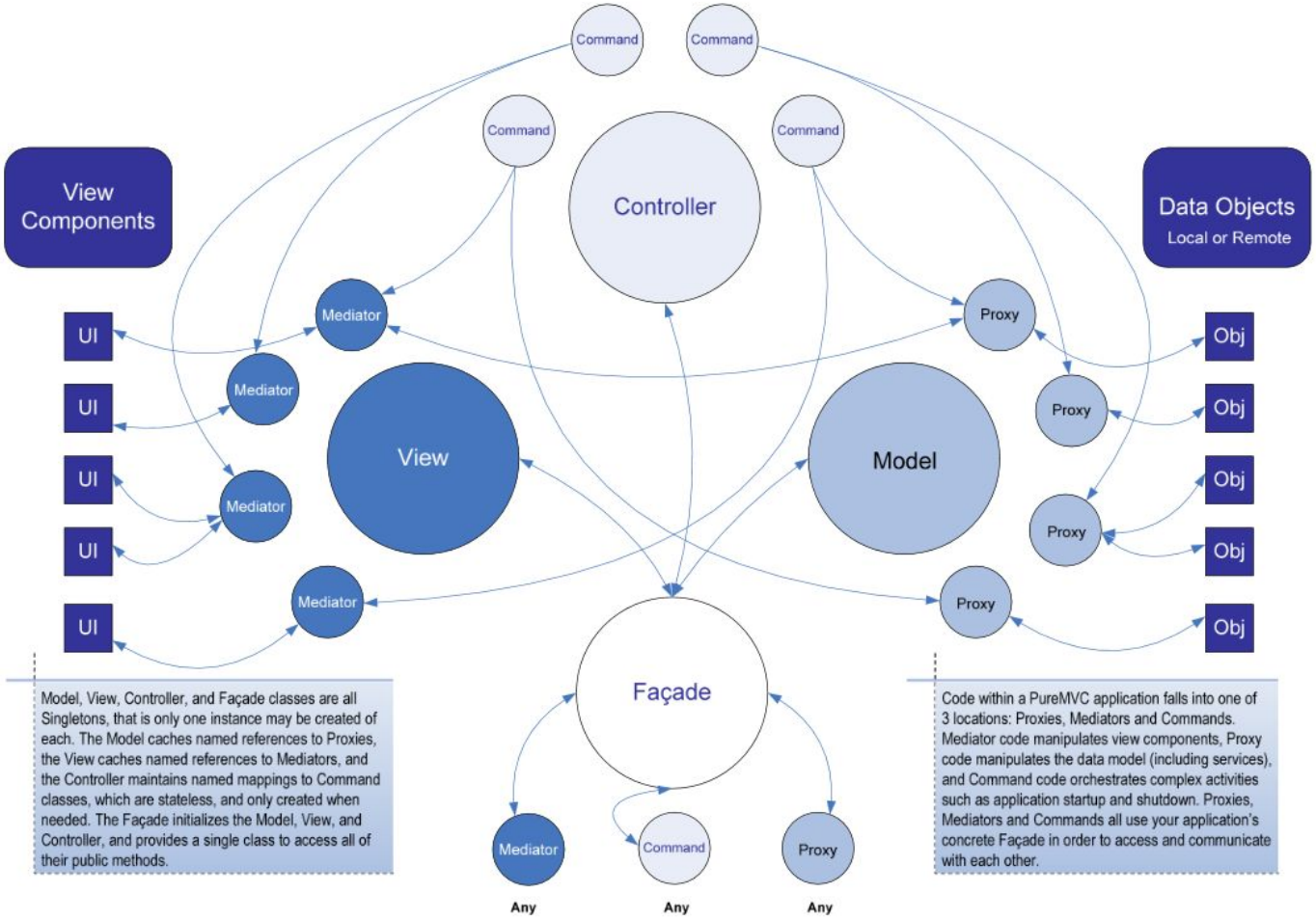
PureModel-View-Controller (PureMVC)

PureMVC е олекотена програмна рамка за създаване на приложения, базирани на класическия шаблон за дизайна Model-View-Controller. Тя поддържа модулно програмиране чрез използване на Multiton Core вместо Singletons, характерни за стандартната версия.



puremvc framework

CONCEPTUAL DIAGRAM AND INTRODUCTION



PureModel-View-Controller (PureMVC)

PureMVC е рамка за създаване на приложения въз основа на утвърдения модел на дизайн-изглед-контролер (MVC). Безплатната рамка с отворен код първоначално е била приложена в езика ActionScript 3, в комбинация с Adobe Flex, Flash Builder и AIR и оттогава се пренесе за почти всички големи платформи за уеб програмиране.

- *Data objects* данните могат да са разположени локално или отдалечено и се управляват от помощници (*Proxies*).
- Компонентите на *View*-то, които отговарят за потребителския интерфейс, се управляват от Медиаторите (*Mediators*).
- *Командите* (*Commands*) могат да взаимодействат с *Proxy*, Mediators, както и да задействат или изпълняват други команди (Commands).

Model-View-ViewMode (MVVM)

Модел-Изглед-Изглед-на-Модела (*Model-View-ViewModel*) – не само представлява реализация на MVC модела, но и го разширява. По своята същност е многослоен архитектурен шаблон, най-често използван с технологиите WPF, Silverlight и Windows Apps. Последно време навлиза в употреба при разработката на приложения с HTML и JavaScript. *Model-View-ViewModel* разделя представителната логика (визуализацията) от бизнес логиката (функционалността) на приложенията. Така бъдещата промяна и в двете става по-лесна за дизайнери и програмисти. Всички изменения могат да се направят независимо.

Model-View-ViewMode (MVVM)

Технологиите за разработка на приложения като Windows Forms, WPF, Silverlight и Windows Phone осигуряват интуитивна работа с програмната среда. Това позволява на разработчика да борави със стандартните компоненти за изграждане на GUI само с провлачване върху дизайнерска повърхност, след което записва кода в отделни файлове за визия и функционалност. Тъй като такива приложения нарастват по размер и обхват и често се променят, то възникват проблеми със сложната поддръжка. Тези проблеми включват тясното свързване между контролите на потребителския интерфейс и бизнес логиката, което увеличава разходите за извършване на промени в потребителския интерфейс и води до трудности при unit testing-а.

Model-View-ViewMode (MVVM)

Основните мотиви за развоя на приложенията чрез използването на модела MVVM, са следните:

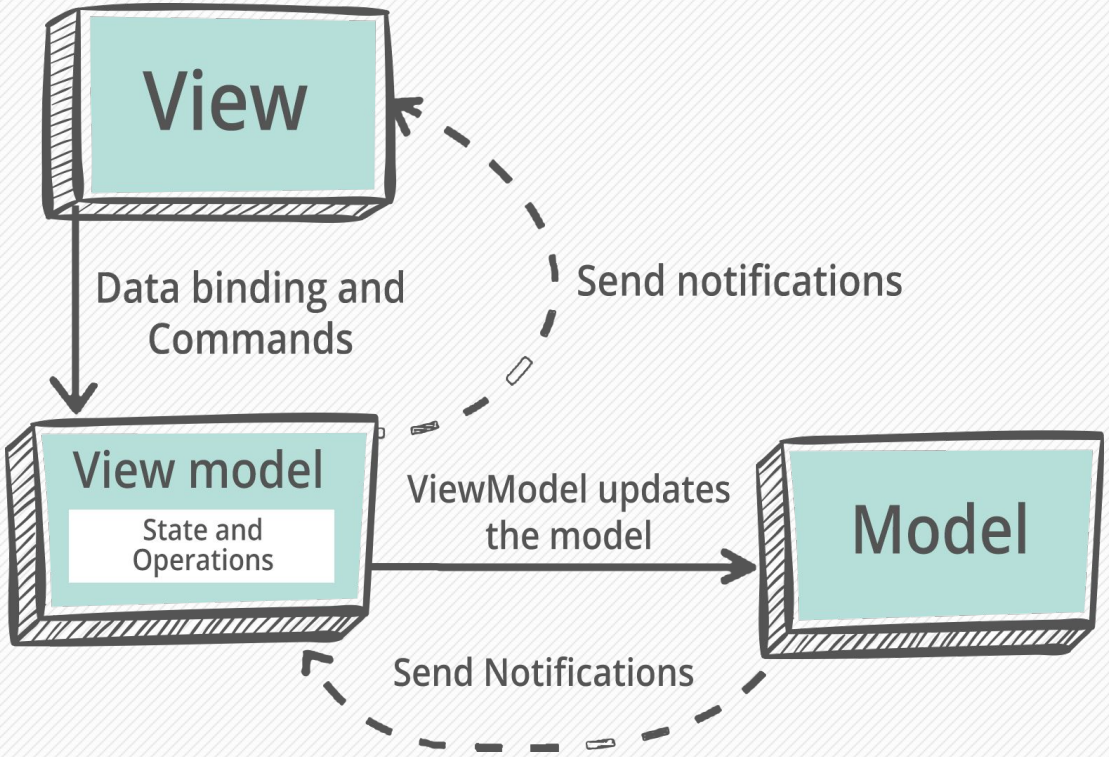
- *Осигурява разделение на логиката. Плътно обвързан, устойчив на промените, смесен код, води до всякакъв вид дългосрочни проблеми с поддръжката, които в крайна сметка причинява неудовлетвореност в клиента на доставения софтуер.*
- *Чистото разделение между логиката на приложението и потребителския интерфейс ще направи приложението по-лесно за тестване, поддръжка и развой. То подобрява възможностите за преизползваемост на кода и улеснява работния процес на разработчик/дизайнер.*

Model-View-ViewMode (MVVM)

Основните мотиви за развоя на приложенията чрез използването на модела MVVM, са следните:

- *Ускорява работният процес на разработчика и дизайнера. Когато XAML кодът на интерфейса не е плътно свързан с функционалността (code-behind), дизайнерите лесно могат да реализират своите идеи.*
- *Увеличава възможността за лесно тестване на приложенията. Преместването на логиката на потребителския интерфейс в отделен клас прави тестването на птделните модулни единици много по-лесна.*

Model-View-ViewMode (MVVM)



Model-View-Presenter

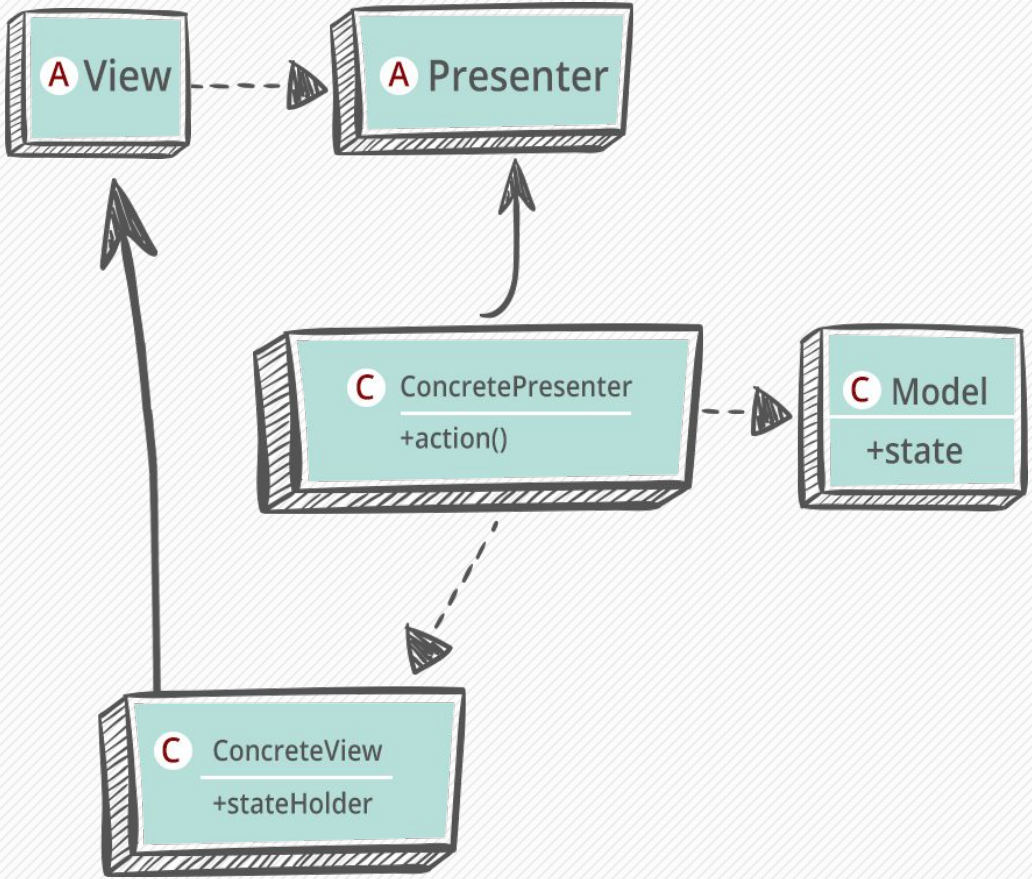
Модел-Изглед-Презентатор (*Model-View-Presenter – MVP*) – отново наследява MVC модела. Този архитектурен шаблон се използва най-вече при изграждането на потребителски интерфейси. Тук моделът е представен като абстракция, която предоставя достъп до данните на едно приложение посредством множество интерфейси. Изгледът съдържа програмния код, необходим за визуализацията. Той може да включва YAML или PHP шаблони, JSP или ASP страници и други, в зависимост от графичния потребителски интерфейс и платформата, за която ще се използва. Презентаторът дава постоянна, двупосочна връзка с изгледа чрез функции и методи.
За всеки определен изглед е налична само една инстанция на презентатора. *При MVC шаблона за един контролер могат да са налични множество изгледи.*

Model-View-Presenter

Обратно на MVC, при MVP цялата репрезентационна логика е изнесена в презентатора:

- Модел е интерфейсът дефиниращ данните, които ще се показват.
- Изглед е пасивен интерфейс, който показва данните (модела) и рутира потребителските команди (събития) към презентатора, където се предприемат съответните действия.
- Презентаторът действа върху модела и изгледа.
- Презентаторът форматира данните, които се показват в изгледа.
- Съществува един единствен презентатор за всеки изглед.

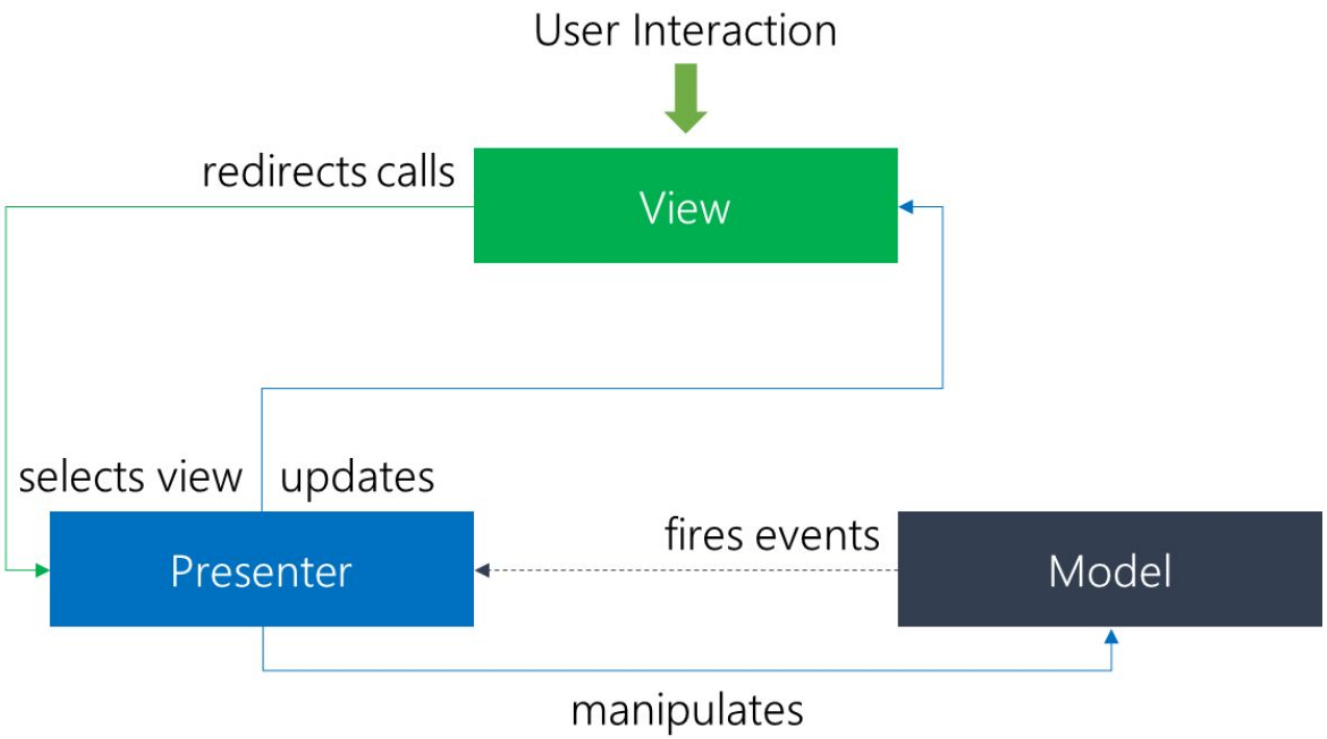
Model-View-Presenter :: Диаграма



Model-View-Presenter :: Диаграма

- Съществува истинска друпосочна комуникация с изгледа.
- Всеки изглед имплементира определен View-интерфейс.
- В изгледа се държи референция към инстанция на презентатора.
- Събитията се пренасочват към презентатора от изгледа.
- Изгледът никога не подава специфични за него неща, (например графични контроли) на презентатора.

Model-View-Presenter :: MVP в действие



Model-View-Presenter :: Модел и изглед

Моделът:

- Комуникира със слоя за връзка с Бази Данни.
- Генерира съответните събития, когато се променят данните.

Изгледът:

- Визуализира данните.
- Получава събитията и репрезентира данните.
- Притежава основна валидация, (например невалиден имейл и др.)

Model-View-Presenter :: Презентаторът

Презентаторът:

- Разделя конкретен изглед от модела.
- Поддържа изгледа при взимането на комплексни решения.
- Комуникира с модела.
- Комплексни валидации, (например включващи други източници на данни).
- Изпълнява заявки над модела.
- Извлича информация от модела, форматира я и я изпраща на изгледа.
- Изгледът се обновява посредством същия събитиен подход, използван от MVC.

Model-View-Presenter :: Практически съвети

- Днес MVP се използва предимно в разработката на десктоп клиенти.
- Полулярни работни рамки съдържат много полезни базови имплементации:
  - MVC# (.NET)
  - GWT (Java for web development)
- На практика разликите с MVC са малки.
- Понякога общият дизайн изглежда като MVP, когато всъщност е MVC.
- Обикновено писането на MVP-работна рамка от нулата не се препоръчва (освен за обучителни цели или при комерсиална нужда от това), както е при MVC.
- Днес MVVM е по-популярен за разработка на десктоп клиенти.

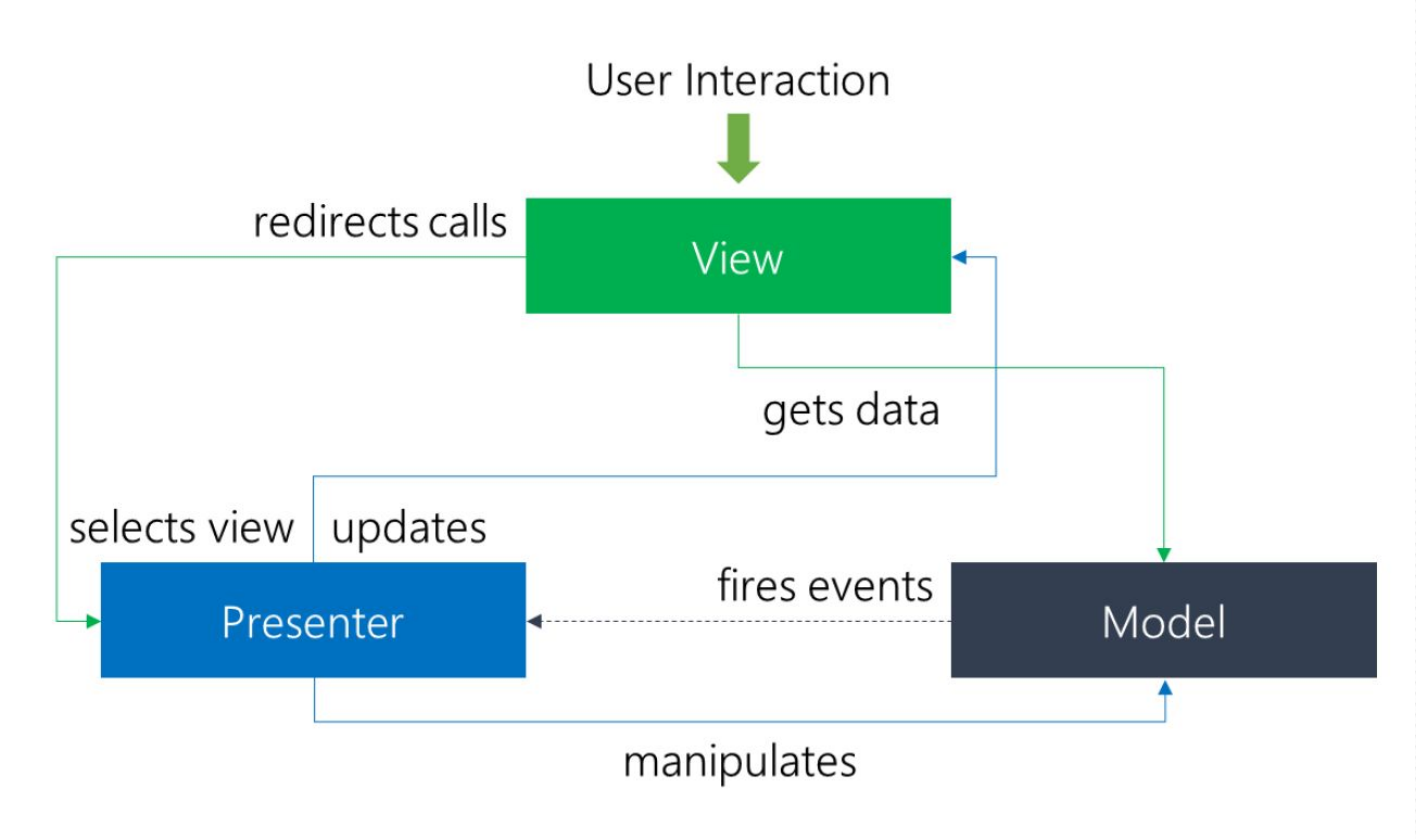
Model-View-Presenter :: MVP модели

Има два основни режима в MVP:

1. *Пасивен изглед* - Взаимодействието с продукта се обработва от презентатора. Изгледът се обновява само и единствено от презентатора.

2. *Надзиравщ контролер* - Изгледът взаимодейства с модела посредством просто обвързване (binding). Изгледът се обновява от презентатора чрез обвързване на данните (data-binding).

Model-View-Presenter :: MVP с data-binding



Литература

Гама Е., Хелм Р., Джонсън Р., Влисидес Д., *Шаблони за дизайн*, СофтПрес, 2005, ISBN: 9546853526.

http://puremvc.org
http://puremvc.org/docs/PureMVC_Conceptual_and_Intro.pdf

https://msdn.microsoft.com/en-us/library/hh848246.aspx