ЛЕКЦИЯ

Архитектурные шаблоны проектирования

Лектор Крамар Ю.М.

Содержание

1. Введение в архитектурные шаблоны проектирования
2. MVC
3. Document-View
4. MVP
5. MVVM
6. Применение MVC. Практика
7. Шаблон Репозиторий (Repository)
8. Шаблон Единица работы (Unit of Work – UoW)
9. Многоуровневая (многослойная) архитектура

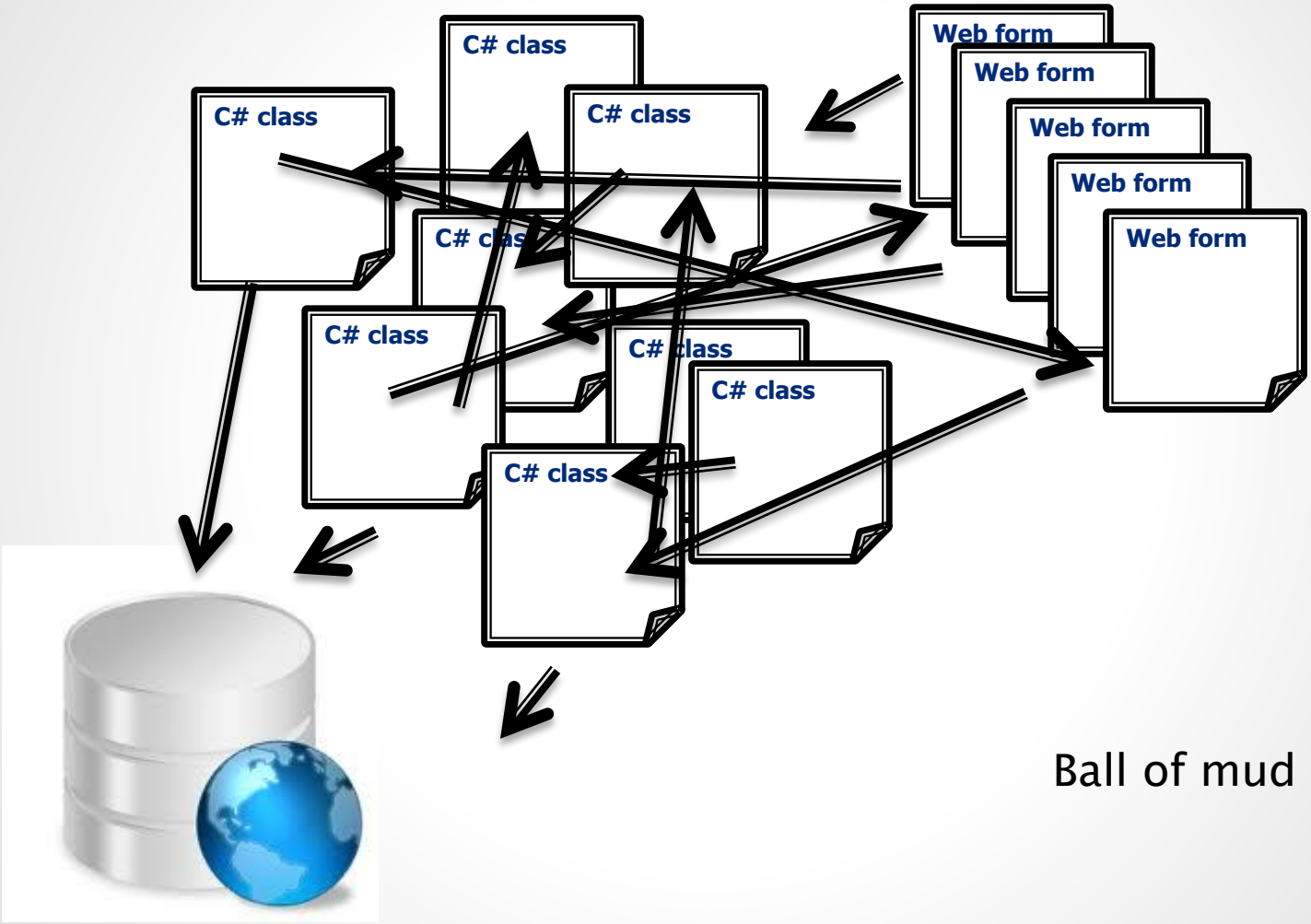
ВВЕДЕНИЕ В АРХИТЕКТУРНЫЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ

Введение

Очень многие программисты начинают реализацию проекта фактически сразу же после возникновения идеи.

В случае работы над маленьким проектом, некоторые вопросы проектирования можно опустить. На первых порах все будет идти как по маслу, быстро и без проблем, однако с увеличением проекта он может превратиться в immutable **ball of mud**…

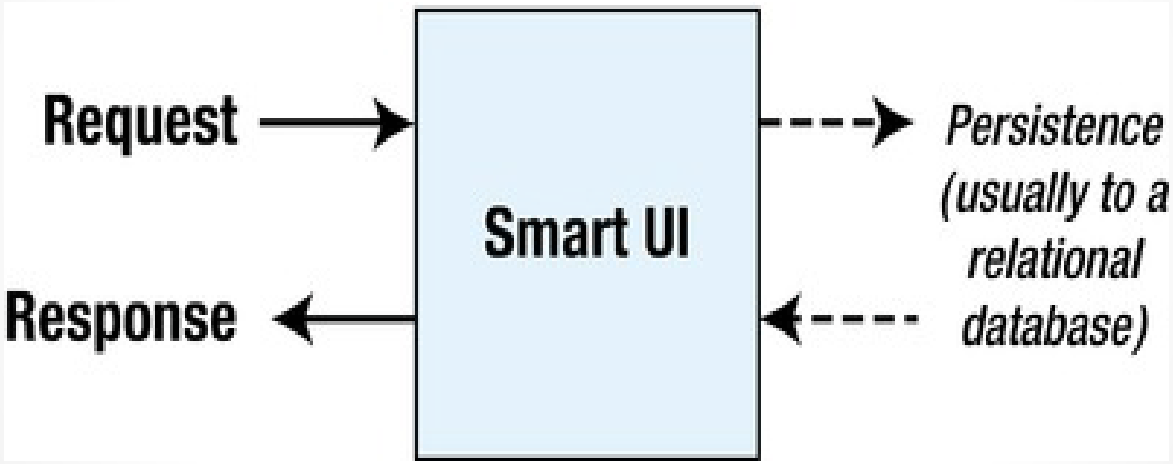
Введение



Ball of mud

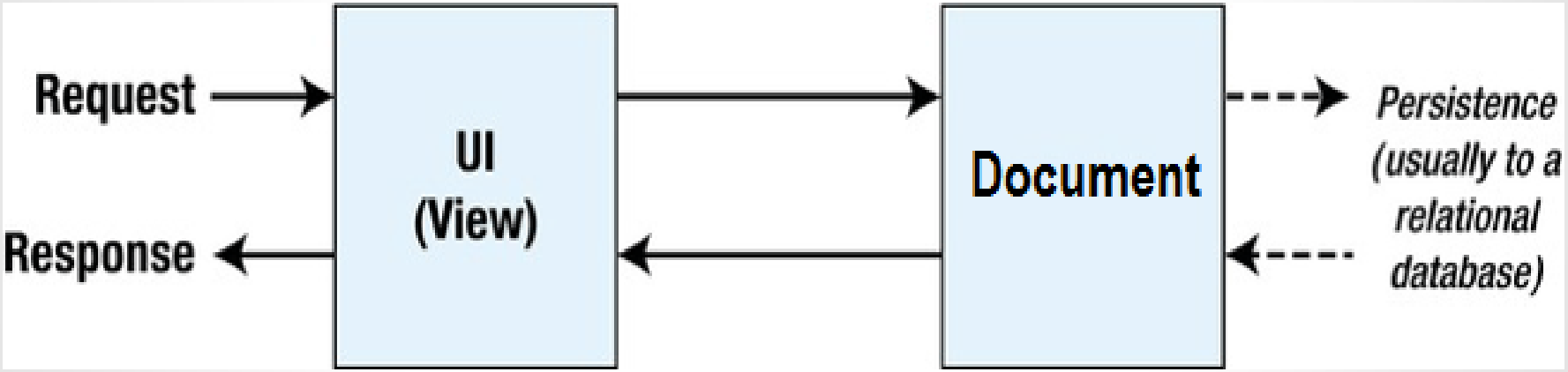
Введение

Smart UI



Введение

Интерфейс – Все остальное



Введение

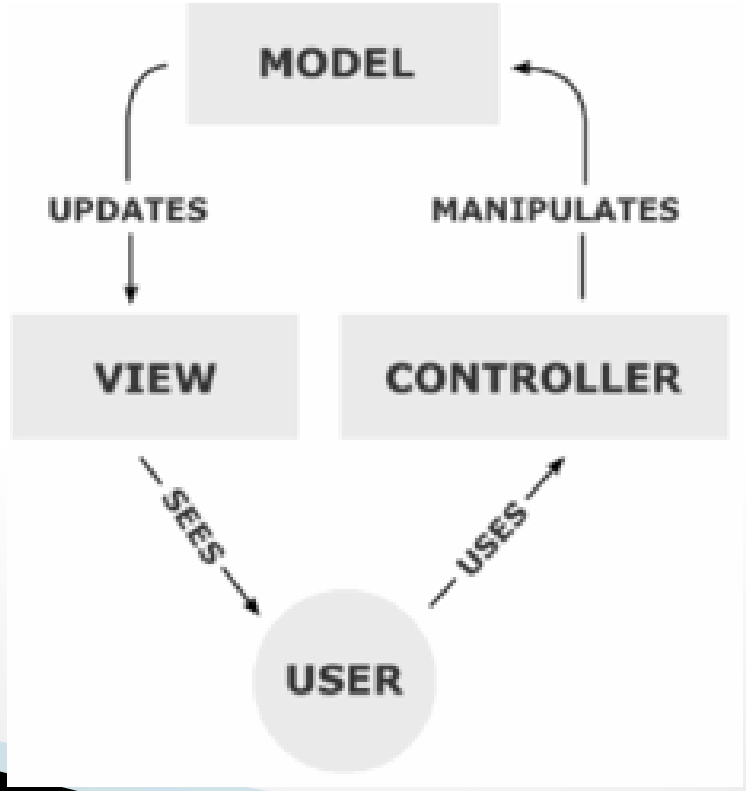
- **Шаблон проектирования** (англ. design pattern) — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Каковы бы ни были особенности современных паттернов, причина их создания, а отсюда их свойства и назначение применения таковы, что разработчики стремятся выделить, сделать как можно более независимыми отдельные компоненты кода, чтобы их можно было повторно использовать и легко заменять.

MVC

MVC

- Первым паттерном и родителем более поздних паттернов считается паттерн **model-view-controller (MVC).**



MVC

- Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента
- *Модель* (англ. *Model*). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.
- *Представление*, *вид* (англ. *View*). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.
- *Контроллер* (англ. *Controller*). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

MVC

- MVC появился в языке SmallTalk в конце семидесятых
- Цель: архитектурное решение, которое позволяло бы манипулировать графическими представлениями данных некоего приложения, таким образом, чтобы изменение *Представления* этих данных не влияло на бизнес-логику и данные (*Модель*) приложения, а также чтобы была возможность иметь несколько *Представлений* для одной *Модели*.

MVC. Model

- Под *Моделью* обычно понимается часть, содержащая в себе функциональную логику приложения, или то, что обычно называется «Business Layer», «Бизнес-слой» или «Слой бизнес логики».

MVC. Model

Признаки *Модели*:

- 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;

MVC. Model

- Основная цель паттерна – сделать так, чтобы *Модель* была полностью независима от остальных частей и практически ничего не знала об их существовании, что позволило бы менять и *Контроллер* и *Представление* модели, не трогая саму *Модель* и даже позволить функционирование нескольких экземпляров *Представлений* и *Контроллеров* с одной *Моделью* одновременно. Вследствие чего, *Модель* ни при каких условиях не может содержать ссылок на объекты *Представления* или *Контроллера*.

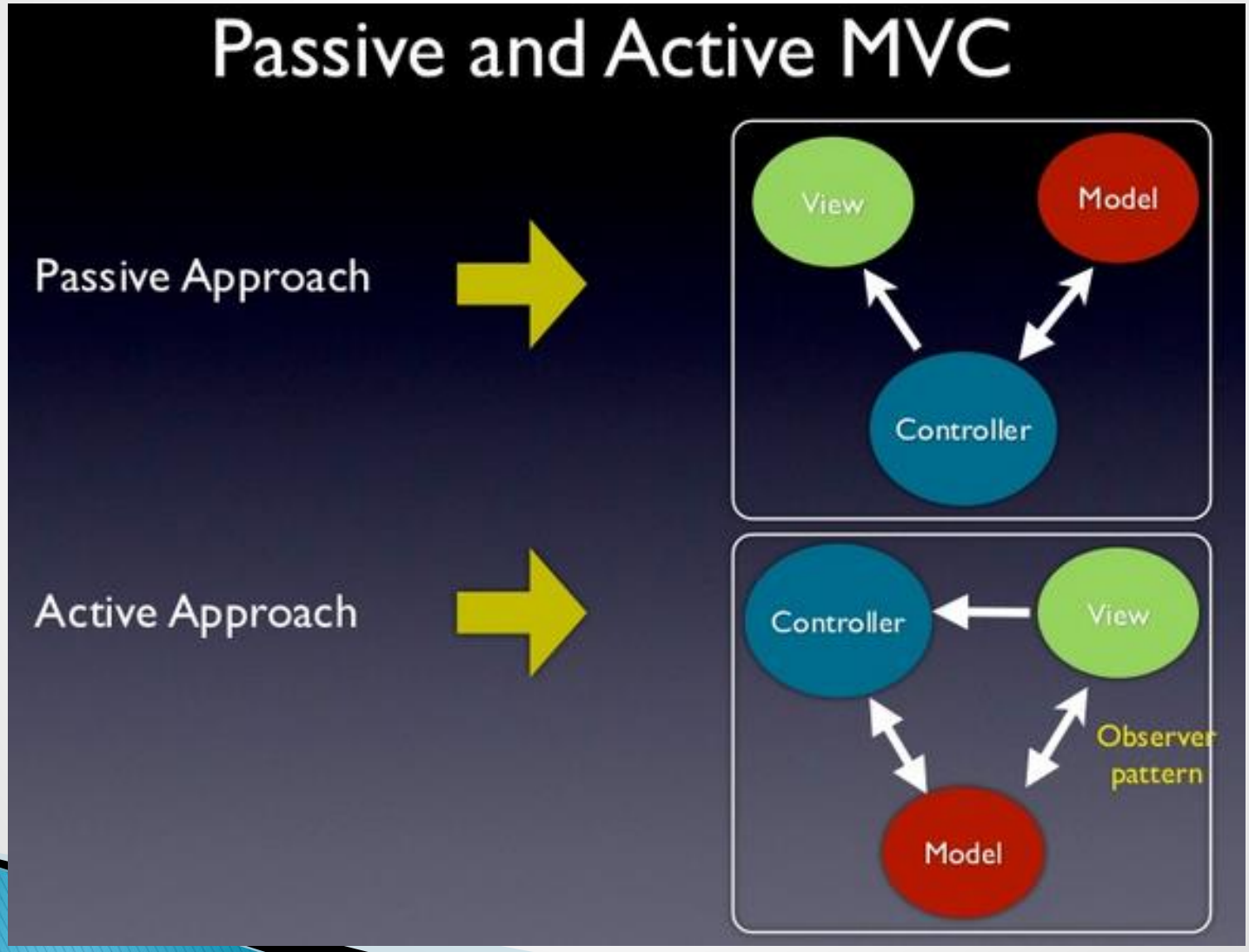
MVC. Model

- **Passive Model MVC** (пассивная модель) – *Модель* не имеет никаких способов воздействовать на *Представление* или *Контроллер* и только используется ими в качестве источника данных для отображения. Все изменения модели отслеживаются *Контроллером* и он же отвечает за перерисовку *Представления*, если это необходимо.

MVC. Model

- **Active Model MVC** (активная модель) – *Модель* имеет возможность оповестить *Представление* о том, что в ней произошли некие изменения, и *Представление* может эти изменения отобразить. Как правило, механизм оповещения реализуется на основе паттерна Observer (обозреватель), *Модель* просто бросает сообщение, а *Представления*, которые заинтересованы в оповещении, подписываются на эти сообщения

MVC. Model


Passive and Active MVC
Passive Approach
View
Model
Controller
Active Approach
Controller
View
Observer pattern
Model

MVC. Model

Часто неправильно трактуют архитектурную модель MVC как **пассивную модель** MVC. В этом случае *модель* выступает исключительно совокупностью функций для доступа к данным, а *контроллер* содержит бизнес-логику. В результате код *моделей* по факту является средством получения данных из СУБД а *контроллер* представляет собой типичный модуль, наполненный бизнес-логикой, или скрипт в терминологии веб-программирования.

MVC. Model

В объектно-ориентированном программировании используется **активная модель** MVC, где *модель* — это не только совокупность кода доступа к данным и СУБД, но и вся бизнес-логика. Следует отметить возможность *модели* инкапсулировать в себе другие модели. В свою очередь, *контроллеры* представляют собой лишь элементы системы, в чьи непосредственные обязанности входит приём данных из запроса и передача их другим элементам системы.

MVC. View

- В обязанности *Представления* входит отображение данных полученных от *Модели*. Обычно *Представление* имеет свободный доступ к *Модели* и может брать из нее данные, однако это доступ только на чтение, ничего менять в *Модели* или даже просто вызывать методы приводящие к изменению ее внутреннего состояния, *Представлению* позволять нельзя.

MVC. View

Признаки *Представления*:

- 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.**

MVC. View

- В случае активной *Модели Представление* может подписаться на события изменения *Модели* и перерисовываться, забрав измененные данные, при получении соответствующего оповещения. Для взаимодействия с *Контроллером* представление, как правило, реализует некий интерфейс, известный *Контроллеру*, что позволяет менять представления независимо и иметь несколько представлений на *Контроллер*.

MVC. Controller

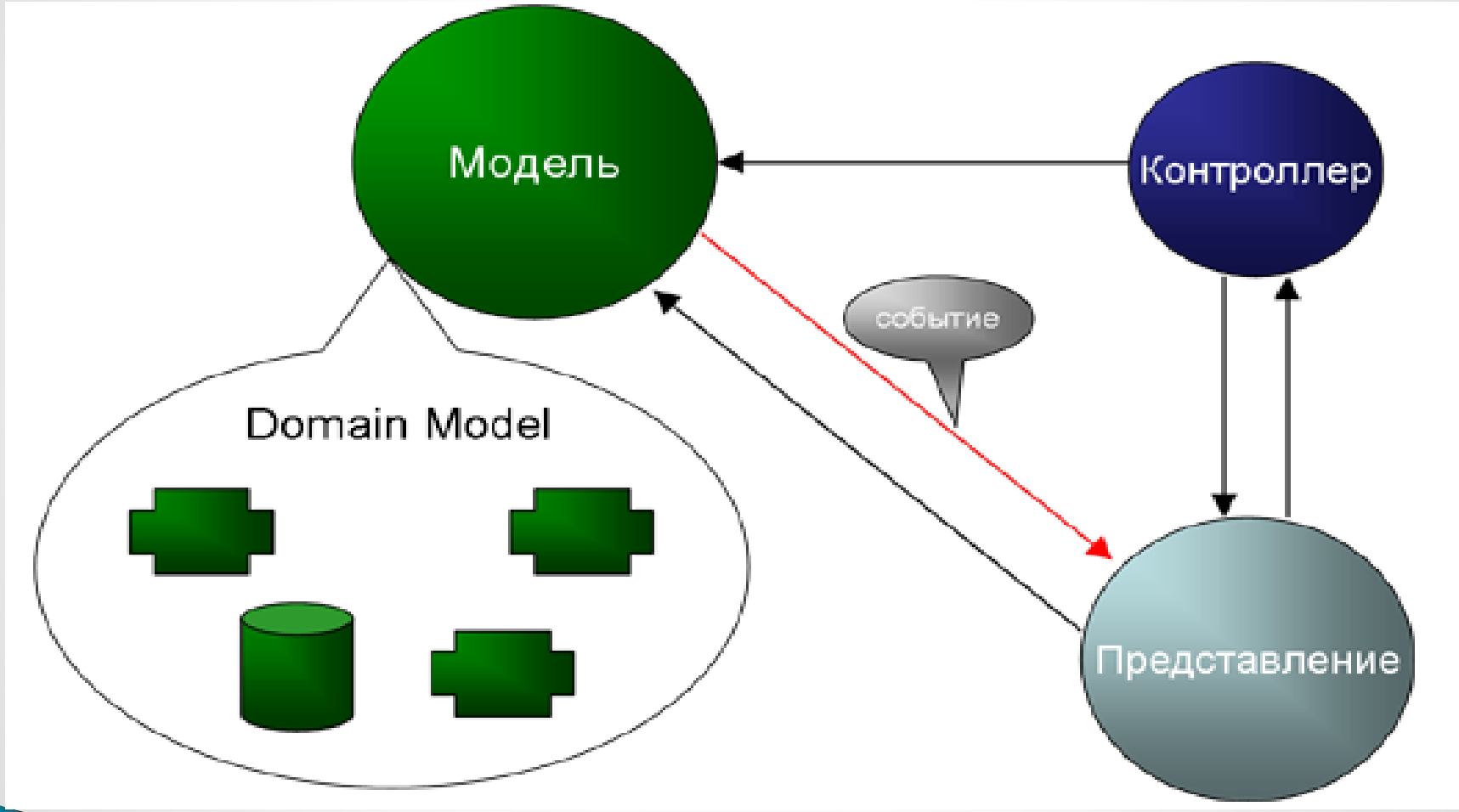
- В задачи *Контроллера* входит реакция на внешние раздражители и изменение *Модели* и/или *Представления* в соответствии с заложенной в него логикой. Один *Контроллер* может работать с несколькими *Представлениями* в зависимости от ситуации, взаимодействуя с ними через некий заранее известный интерфейс, который эти *Представления* реализуют.

MVC. Controller

Признаки *Контроллера:*

- *Контроллер* определяет, какое *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*

MVC



MVC. Особенности

- выделение отдельного *Контроллера* не так важно как отделение *Представления* от *Модели,* и *Контроллер* вполне может быть интегрирован в *Представление*, тем более что в классическом варианте MVC логики в *Контроллере* не очень много.

MVC. Особенности

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**

MVC

Наиболее распространенные виды MVC-паттерна, это:

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

DOCUMENT-VIEW

Document-View

- Следующим этапом развития MVC стал паттерн Document-View, хорошо известный по таким библиотекам как Turbo Vision (Pascal 6.0), Microsoft Foundation Class Library и многих других, вплоть до WinForms.

Document-View

- В этой версии MVC *Контроллер* интегрирован в *Представление:*
- Так как прежде всего, отделение *Контроллера* от *Представления*, не ключевая часть паттерна;
- Появились графические оболочки, встроенные в ОС, что позволяло не рисовать графические элементы пользовательского интерфейса под каждый проект, а использовать готовые, предоставляемые платформой посредством соответствующего API, при этом, в этих оболочках функции *Контроллера* уже были интегрированы в «контроллы» (которые и являются *Представлениями* или же его частями)

MVP

MVP

- **Model-View-Presenter (MVP)** — шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса.

Эффективно отделяет модель от ее представлений.

Позволяет пользоваться дизайнером форм и имеющимися библиотеками, без ограничений.

Позволяет тестировать логику *Контроллера* независимо от *Представления* и сводит логику *Представления* к минимуму.

Позволяет избегать лишних обращений к *Модели*.

MVP. Presenter

- Элемент *Presenter* в данном шаблоне, берет на себя функциональность посредника (аналогично *Контроллеру* в MVC), и так же отвечает за управление событиями пользовательского интерфейса (такими, как использование мыши), как в других шаблонах обычно отвечает *Представление*.

MVP. Presenter

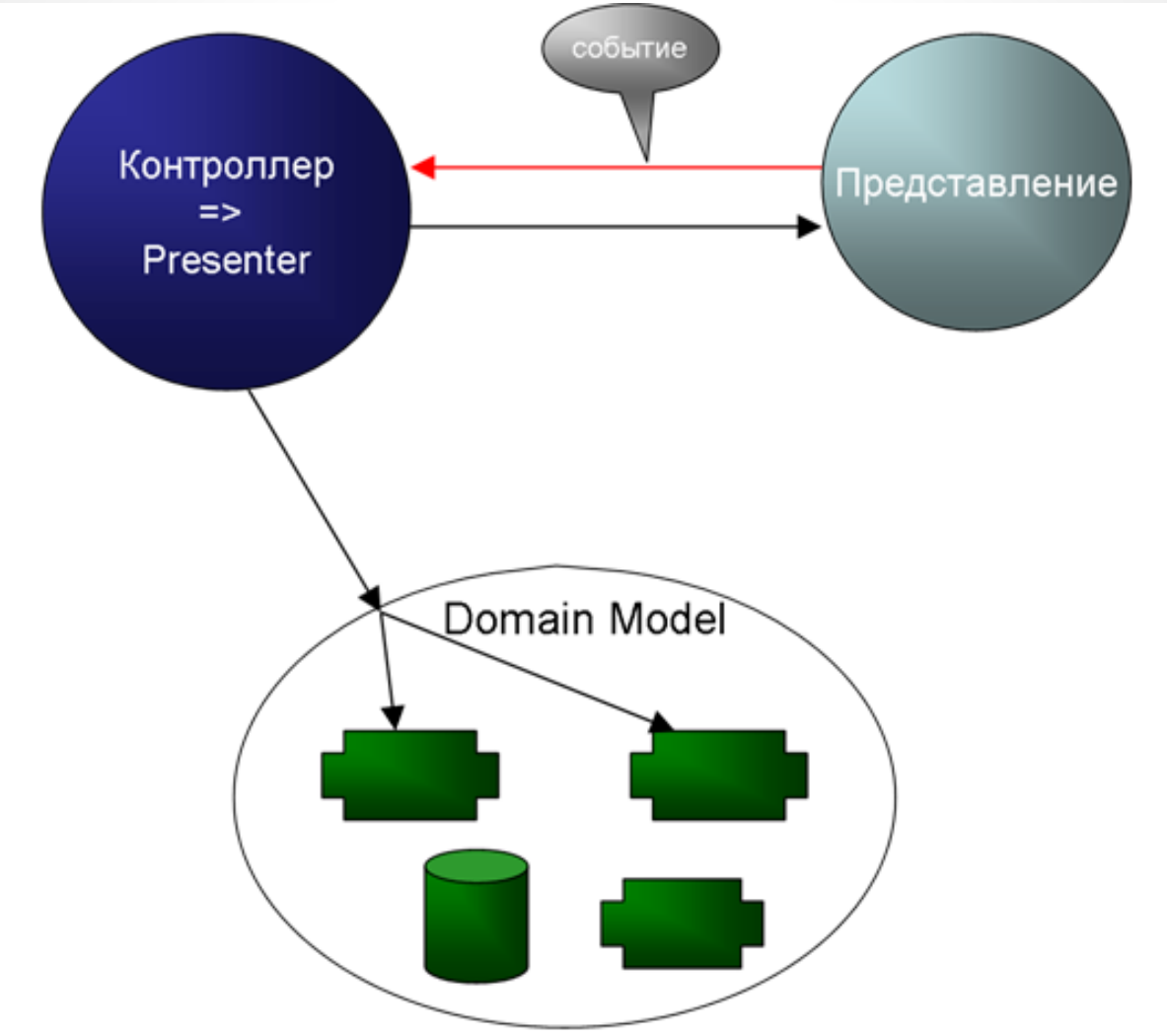
Признаки *Презентера*:

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

MVP. View

- экземпляр *Представления* создаёт экземпляр *Presenter*-а, передавая ему ссылку на себя. При этом *Presenter* работает с *Представлением* в абстрактном виде, через его интерфейс. Когда вызывается событие *Представления*, оно вызывает конкретный метод *Presenter'а*, не имеющего ни параметров, ни возвращаемого значения. *Presenter* получает необходимые для работы метода данные о состоянии пользовательского интерфейса через интерфейс *Представления*, и через него же передаёт в *Представление* данные из *Модели* и другие результаты своей работы.

MVP


событие
Контроллер
=>
Presenter
Представление
Domain Model

MVP. Особенности

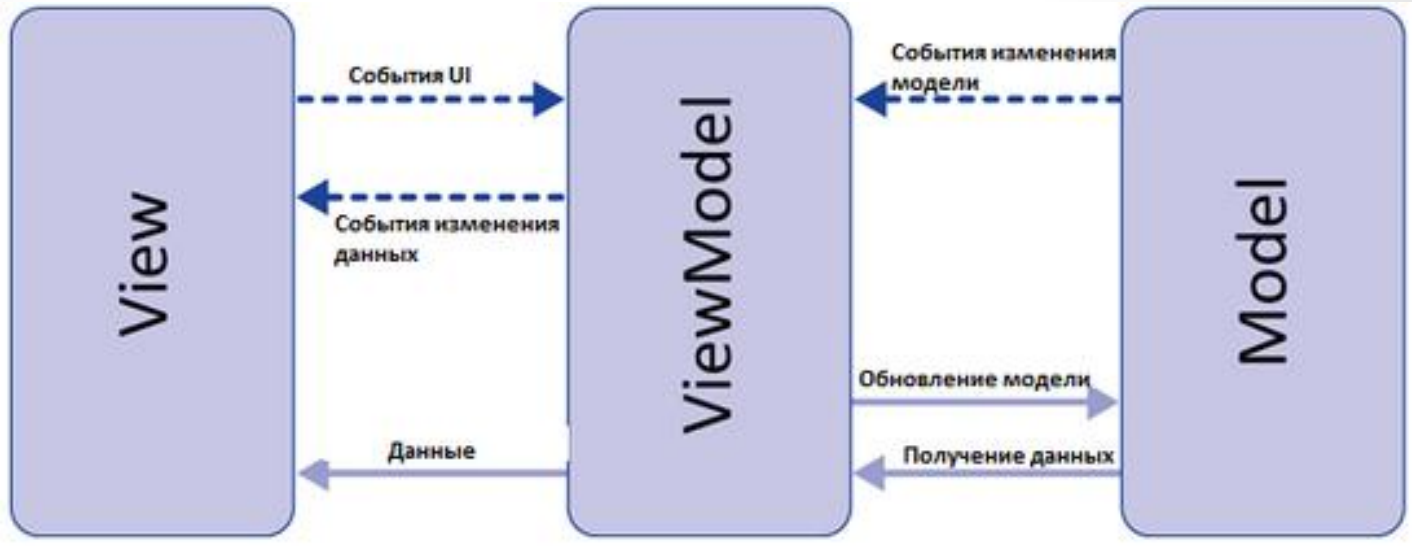
Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, **IView**.ShowErrorMessage(string msg)). Презентер должен иметь ссылку на реализацию соответствующего интерфейса, 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**

MVVM

MVVM

- 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.



MVVM

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.

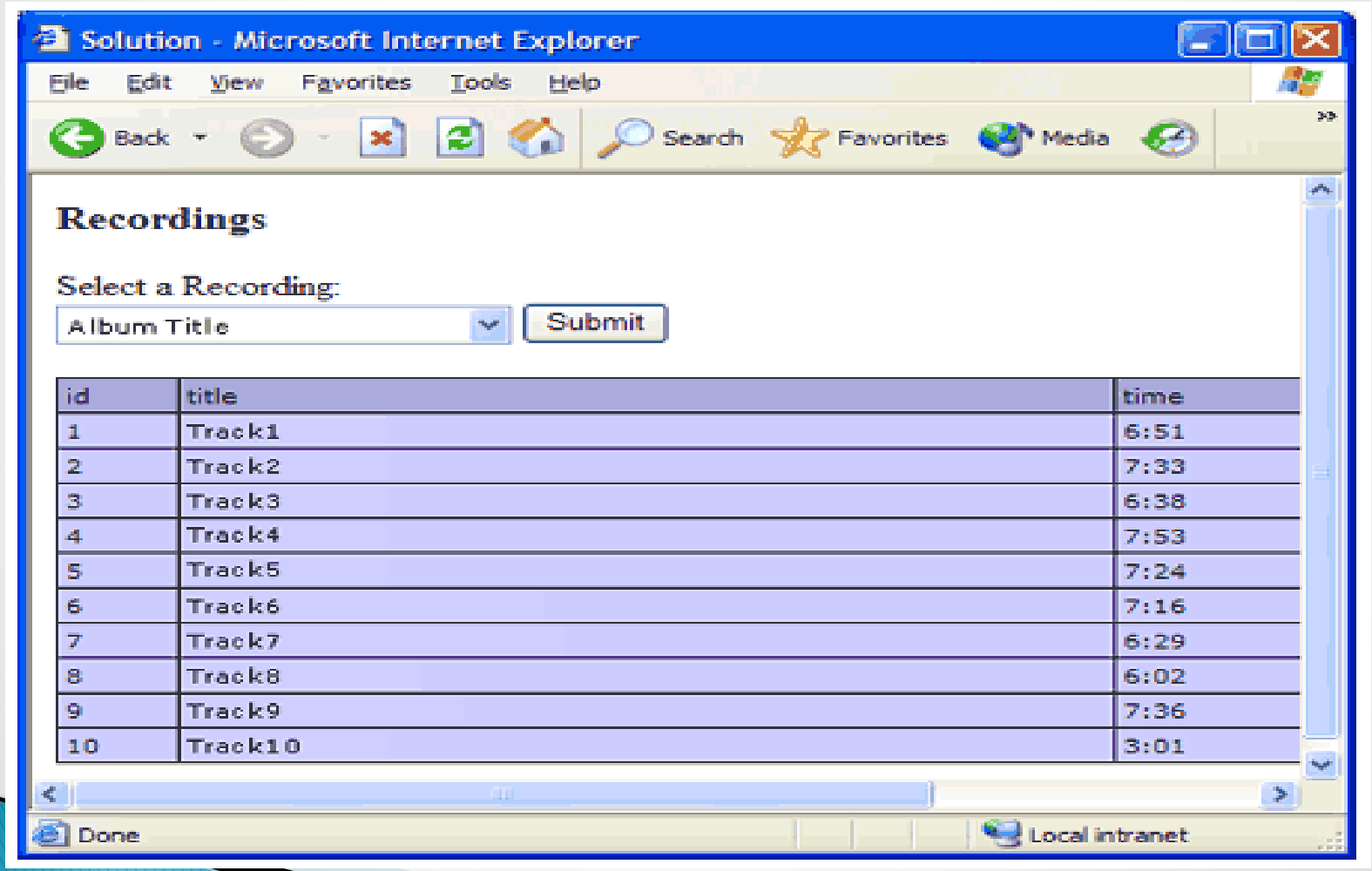
MVVM. Особенности

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**

АРХИТЕКТУРНЫЕ ШАБЛОНЫ. ПРАКТИКА

Применение MVC к проекту ASP.NET



Применение MVC к проекту ASP.NET

1. Single ASP.NET Page

- Простейшая реализация страницы в одном файле.
- Представление и модель намертво склеены. Невозможно работать над кодом командой. Невозможно отлаживать и тестировать. Невозможно использовать код модели повторно.

Generated code

Применение MVC к проекту ASP.NET

2 Code-Behind Refactoring

- Code-behind технология Microsoft Visual Studio .NET – разработки позволяет облегчить задачу деления кода на представления и модели в различные файлы. При этом модель и контроллер пока описываются одним файлом:
- View – .aspx

Generated code

- Model-Controller – .cs

Generated code

Применение MVC к проекту ASP.NET

3 Model-View-Controller Refactoring

- Разнесение кода модели и контроллера в различные файлы. Модель связана с источниками данных и абсолютно не связана с представлением
- Model – .cs

Generated code

- Controller – .cs

Generated code

Применение MVC к проекту ASP.NET

4 Tests

- Разделяемая модель кода в ASP.NET позволяет сделать тестирование более простым. Потому что тестирование модели не требует исполнения кода в ASP.NET . Вместо этого можно использовать средства юнит-тестирования как VS, так и сторонние, например, NUnit
- Test – .cs

Generated code

Применение MVC к проекту ASP.NET

Преимущества MVC

- Уменьшение зависимостей
- Сокращение дублирования кода
- Разделение обязанностей между разработчиками
- Оптимизация и увеличение производительности кода
- Тестируемость кода

Архитектурные паттерны проектирования

Ресурсы:

http://blog.webferia.ru/web/asp-net/pattern-mvc-mvp-mvvm/

https://ru.wikipedia.org/wiki/Model-View-Controller

https://habrahabr.ru/post/215605/

http://metanit.com/sharp/articles/mvc/11.php

http://metanit.com/sharp/mvc5/23.3.php