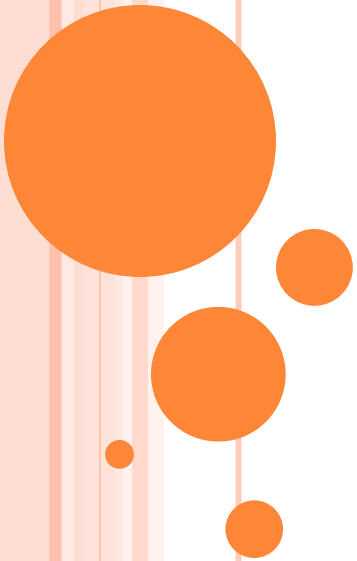


MVVM

Применение паттерна MVVM как оптимального при проектировании WPF и Silverlight приложений

Расчленение системы на слои предоставляет целый ряд преимуществ.

- Отдельный слой можно воспринимать как единое самодостаточное целое, не особенно заботясь о наличии других слоев.
- Можно выбирать альтернативную реализацию базовых слоев.
- Зависимость между слоями можно свести к минимуму.
- Каждый слой является удачным кандидатом на стандартизацию.
- Созданный слой может служить основой для нескольких различных слоев более высокого уровня. В противном случае для каждого протокола высокого уровня пришлось бы изобретать собственный протокол низкого уровня.

Схема расслоения обладает и определенными недостатками.

- Слои способны удачно инкапсулировать многое, но не все: модификация одного слоя подчас связана с необходимостью внесения каскадных изменений в остальные слои.
- Наличие избыточных слоев нередко снижает производительность системы. При переходе от слоя к слою моделируемые сущности обычно подвергаются преобразованиям из одного представления в другое.

Несмотря на это, инкапсуляция нижележащих функций зачастую позволяет достичь весьма существенного преимущества.

Паттерн Model-View-ViewModel (MVVM)

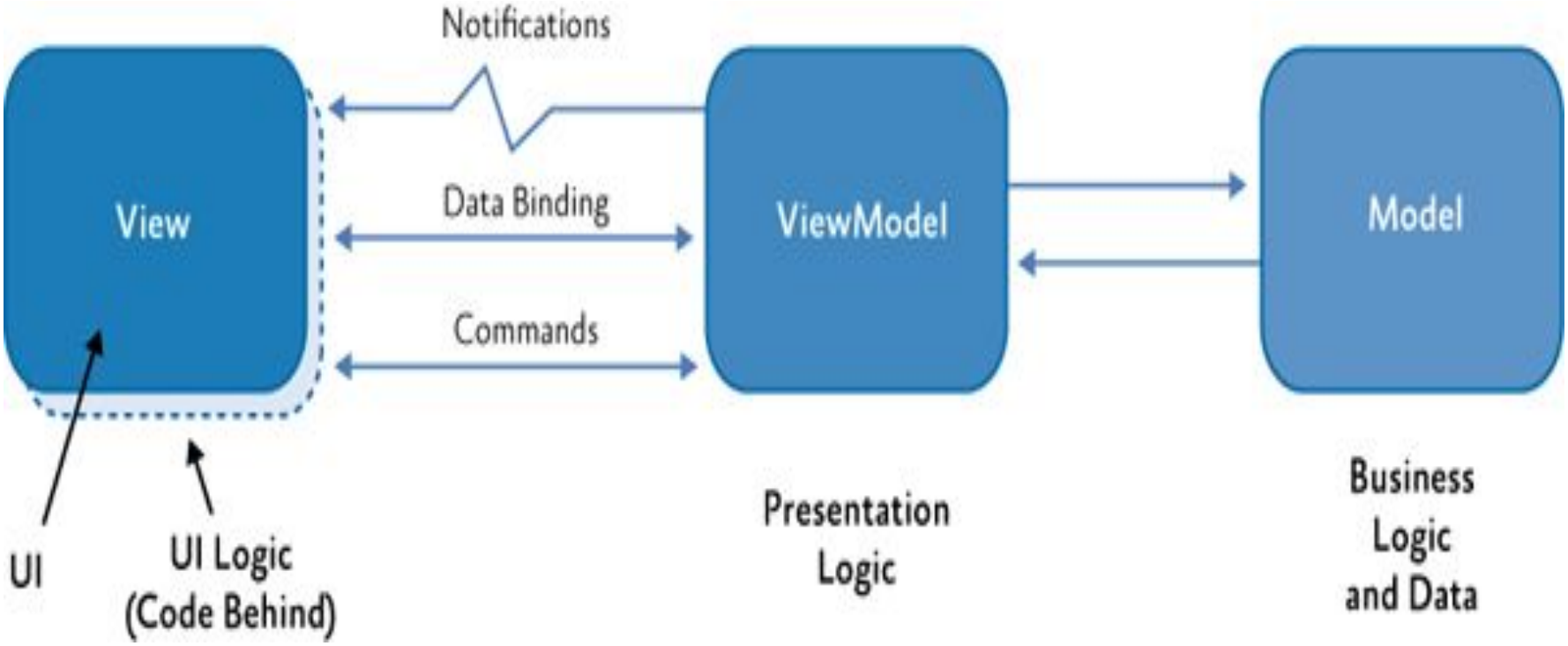
MVVM поможет вам разделить бизнес-логику и логику представления приложения от его пользовательского **интерфейса (UI)**.

Паттерн MVVM:

- ***вид***, который инкапсулирует UI и его логику;
- ***модель представления***, которая инкапсулирует логику представления и состояния;
- ***модель***, *которая инкапсулирует бизнес-логику приложения и данные.*

Однако **самое трудное** при использовании архитектурных слоев - это **определение содержимого и границ** ответственности каждого слоя.

Обязанности и характеристики классов



Класс представления (View)

- Ответственность представления состоит в том, чтобы **определить структуру и появление того, что пользователь видит на экране**.
- **В некоторых случаях, code-behind может содержать код логики UI**, который реализует визуальное поведение, являющееся трудным или неэффективным для выражения в XAML.
- **Недопустимо помещать код логики, нуждающийся в тестировании, в представление**. Как правило, код логики в code-behind представления может быть протестирован через автоматизацию UI.

У представления есть следующие ключевые характеристики:

- Представление определяет элементы управления, их компоновку и стиль.
- Представление ссылается на модель представления через свое свойство DataContext. Элементы управления в представлении привязаны к свойствам и командам модели представления.
- Представление может настроить поведение привязки данных между представлением и моделью представления.
- Представление задаёт и обрабатывает визуальное поведение UI, такое как анимации или переходы, которые могут быть инициированы изменением состояния модели представления или через взаимодействие пользователя с UI.
- Code-behind представления может определить логику UI, чтобы реализовать визуальное поведение, которое трудно выразить в XAML, или которое требует прямых ссылок на определенные элементы управления UI, определенные в представлении.

Класс модели представления (View Model)

Модель представления в паттерне MVVM инкапсулирует логику представления и данные для отображения. **У него нет никаких прямых ссылок на представление** или любое знание о реализации или типе представления.

Как правило, **модель представления определяет команды или действия**, которые могут быть представлены в UI и вызваны пользователем. Типичный примером является то, когда модель представления предоставляет команду **Submit**, которая позволяет пользователю передать данные **веб-сервису или репозитарию данных**.

МОДЕЛЬ ПРЕДСТАВЛЕНИЯ ИМЕЕТ СЛЕДУЮЩИЕ КЛЮЧЕВЫЕ ХАРАКТЕРИСТИКИ

- Модель представления является неотображаемым классом и не наследуется ни от какого базового класса WPF или Silverlight. Она инкапсулирует логику представления, необходимую для поддержки пользовательских действий в приложении. Модель представления является тестируемой независимо от представления и модели.
- Модель представления обычно непосредственно не ссылается на представление. Она реализует свойства и команды, к которыми представление может привязать данные. Она уведомляет представление о любых изменениях состояния через события уведомления через интерфейсы INotifyPropertyChanged и INotifyCollectionChanged.
- Модель представления **координирует взаимодействие представления с моделью**. Она может преобразовать или управлять данными так, чтобы они могли быть легко использованы представлением, и может реализовать дополнительные свойства, которые, возможно, не присутствуют в модели. **Она может также реализовать валидацию данных через интерфейсы IDataErrorInfo или INotifyDataErrorInfo.**
- Модель представления может определить логические состояния, которые представление может визуально представить пользователю.

Представление или Модель Представления?

Часто определение того, где должна быть реализована определенная функциональность, не очевидно.

Общее правило гласит: Что-либо касающееся определенного **визуального отображения** UI на экране и что может быть модернизировано позже (даже если вы в настоящий момент не планируете модернизировать это), должно **войти в представление**; что-либо, что важно для **логического поведения** приложения, должно **войти в модель представления**.

Например, цвет выделения выбранного пункта в поле списка должен быть определен в представлении, но список элементов для отображения, и ссылка на выбранный пункт непосредственно, должны быть определены моделью представления.

Класс модели (Model).

Модель в паттерне MVVM инкапсулирует бизнес-логику и данные. Бизнес-логика определяется как любая логика приложения, которая касается **извлечения и управления** данными приложения и для того, чтобы удостовериться, что налагаются бизнес-правила, которые гарантируют **непротиворечивость и валидность данных**.

Как правило, модель реализует средства, которые облегчают привязку к представлению. Это обычно означает, что поддерживаются уведомления об изменениях свойств или коллекций через интерфейсы **INotifyPropertyChanged** и **INotifyCollectionChanged**. Классы моделей, которые предоставляют наборы объектов обычно наследуются **от класса ObservableCollection<T>**, который обеспечивает реализацию интерфейса INotifyCollectionChanged.

Ключевые характеристики

- Классы модели **являются не визуальными классами**, которые инкапсулируют данные приложения и бизнес-логику. Они ответственны за управление данными приложения.
- **Классы модели** непосредственно **не ссылаются на классы представления** или модели представления и не имеют никакой зависимости от того, как эти классы реализуются.
- Классы модели обычно **предоставляют уведомления об изменении свойств или коллекций** через интерфейсы INotifyPropertyChanged и INotifyCollectionChanged. Это позволяет им быть легко **привязанными к представлению**.
- Классы модели обычно обеспечивают **валидацию данных** и **сообщение об ошибках** через интерфейсы IDataErrorInfo или INotifyDataErrorInfo.
- **Классы модели** обычно используются **вместе со службой или репозитарием**, который инкапсулирует **доступ к данным и кэширование.**

Взаимодействие классов.

Паттерн **MVVM обеспечивает чистое разделение** между **пользовательским интерфейсом** вашего приложения, **его логикой представления**, и его **бизнес-логикой и данными**, разделяя каждый элемент на отдельные классы

Привязка данных

Привязка данных играет очень важную роль в паттерне MVVM.

Ваша модель представления и (в идеале) ваши классы модели должны быть разработаны так, чтобы поддерживать привязку данных. Как правило, это означает, что **они должны реализовать корректные интерфейсы**.