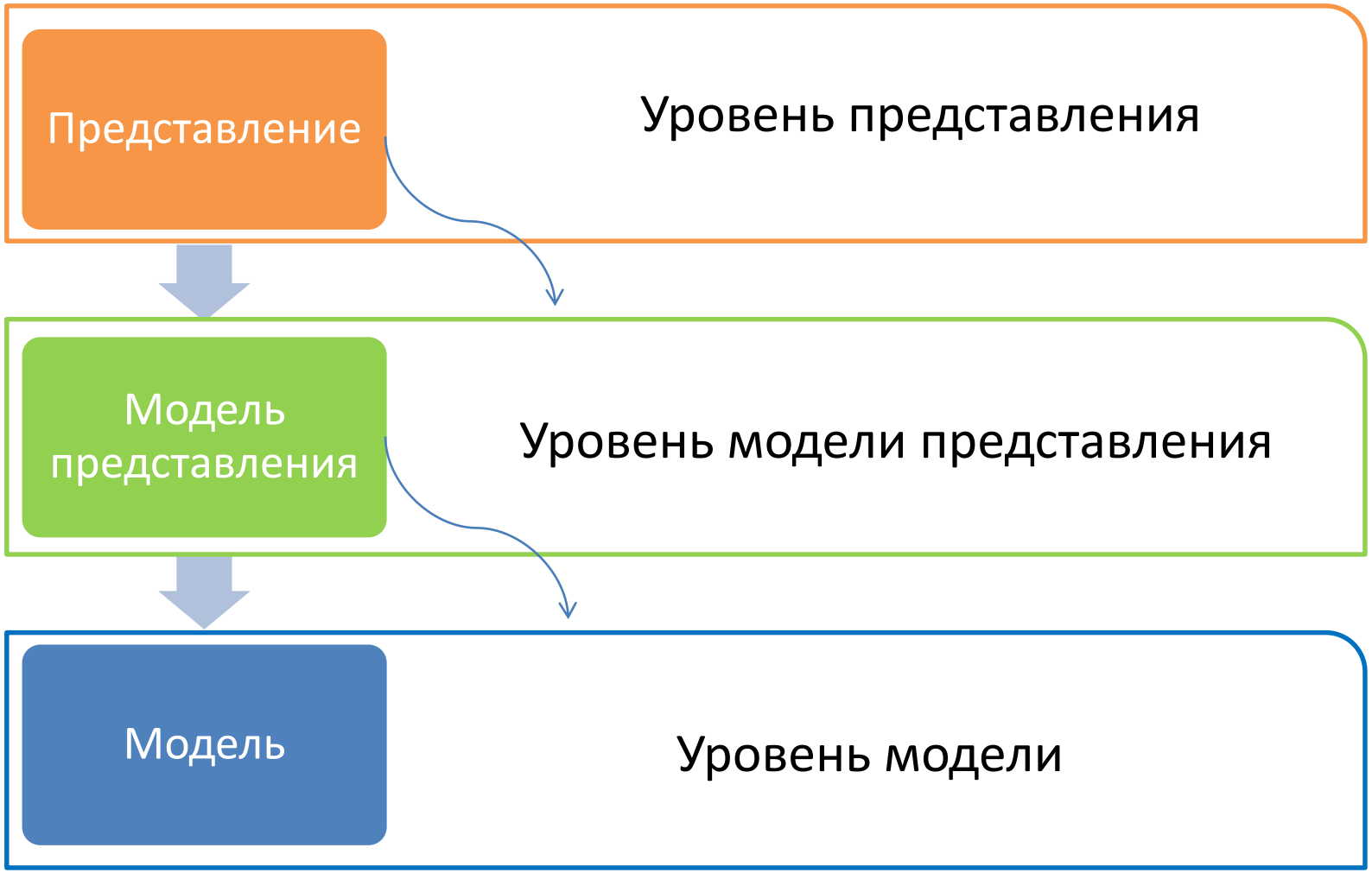
Укрощение MVVM с помощью Catel

Маренков Вадим Викторович

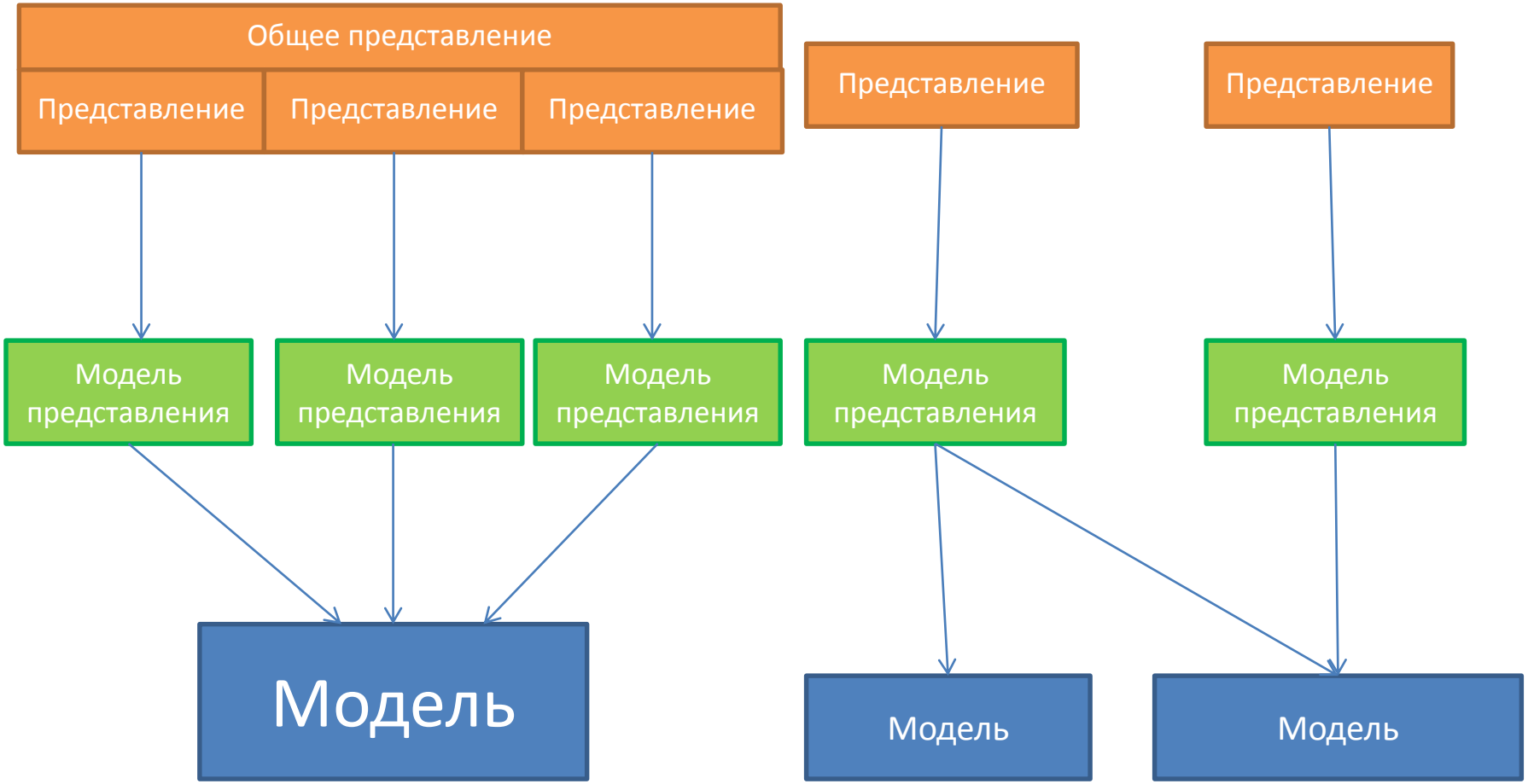
Паттерн MVVM
Model View ViewModel

Общетеоретическая часть.

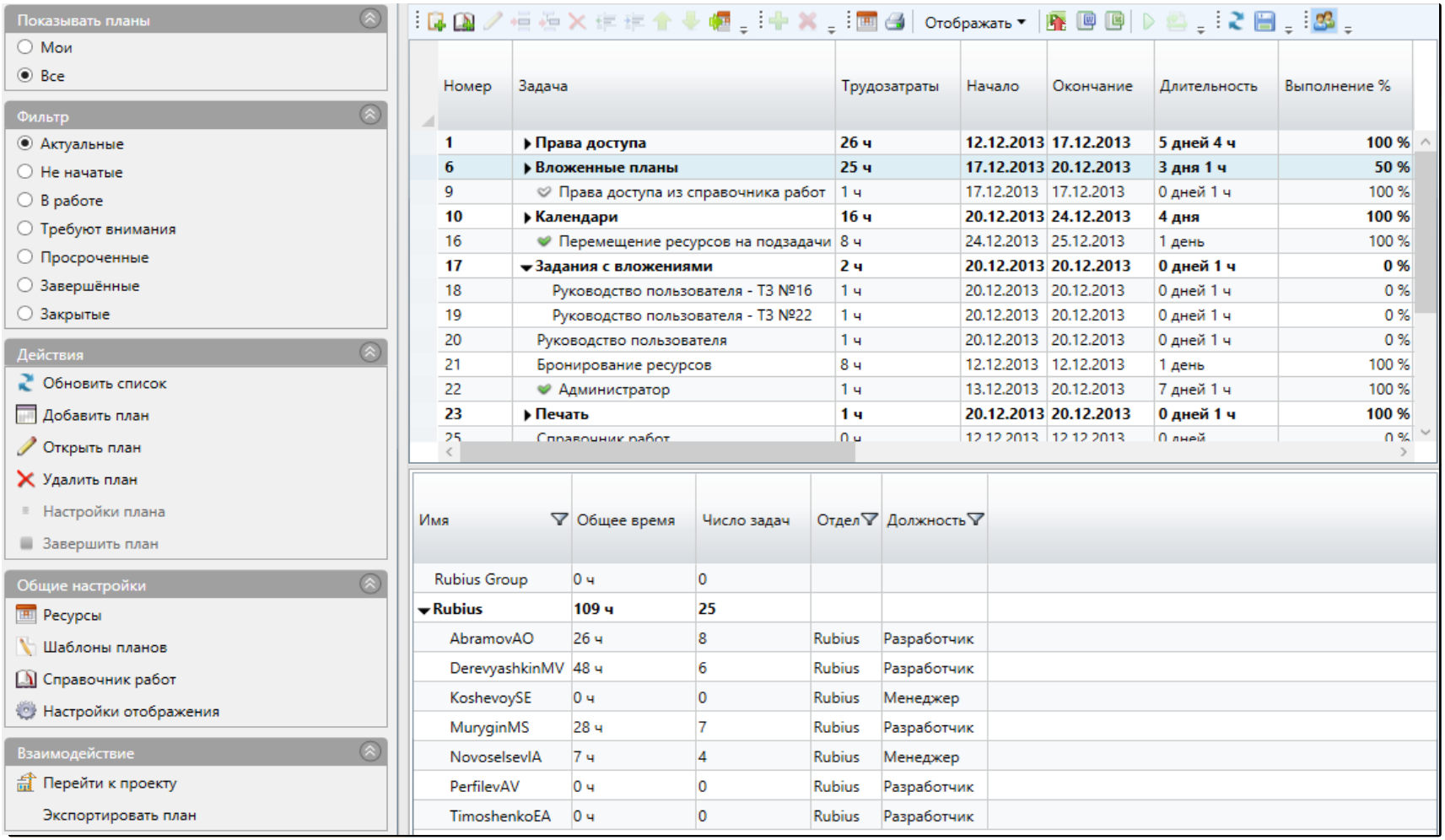
Паттерн MVVM



Варианты реализации.



Композиция ViewModel



Паттерн MVVM. Преимущества.

- Строгая формализация взаимодействия модель – представления.
- Пригодность визуальной логике (в виде VM) для автоматического тестирования.
- Гибкость. Простота (относительная) модификации интерфейса.
- Упрощение многоплатформенной разработки.
- Теоретическая возможность параллельной разработки GUI.

Детали реализация паттерна.

Типичные сложности.

Уровень Model - ViewModel

Основные моменты.

- Связь Model с ViewModel.
- Организация сохранения изменений из VM в Model.
- Рутина.
  - Дублирование свойств модели.
  - Реализация INotifyPropertyChanged.
- Обертка иерархий/композиций в VM.

Создание VM.

- Агрегация модели.

```
var item = new ViewModel(model);
```

- Создание по модели.

```
var item = Manager.CreateVM<ViewModel>(model);
// Предпочтительный вариант для неполного
  отображения класса модели.
```

Сохранение изменений в модель.

- Ручное сохранение измененний.

```
Item.SaveChanges(); / Item.ResetChanges();
(+) Наиболее универсальное;
(-) Наиболее трудоемкое;
```

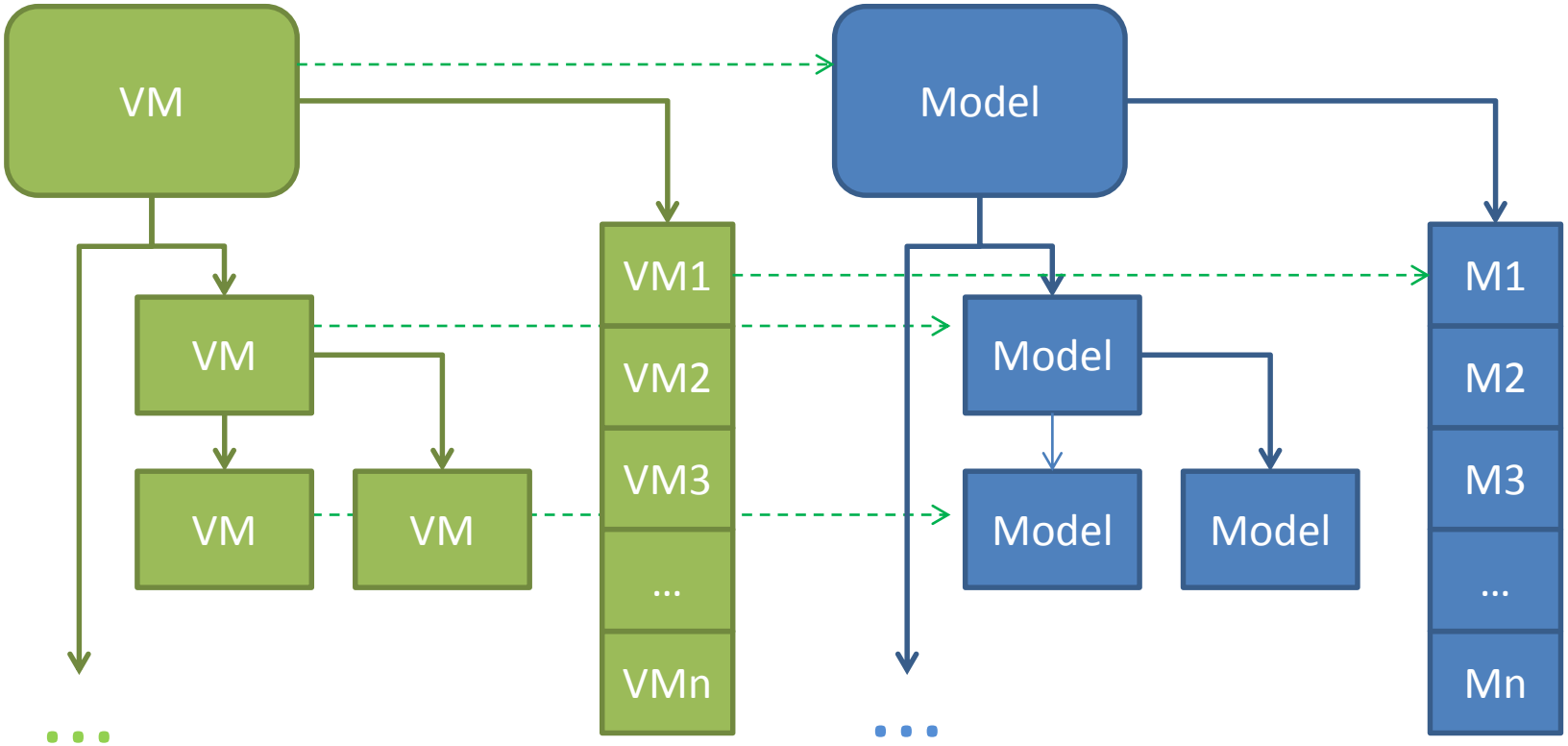
- Автоматическая синхронизация внутри класса VM

```
(+) Автоматическая синхронизация изменений
(-) Ручная реализация отката изменений
```

- Для VM не зависящей от модели.

```
Manager.SaveChanges(model, item);
(+) Простейшая реализация отката изменений
```

Иерархии.



Уровень ViewModel - View

- Создание связи View – ViewModel.
- Доступ к функциональности View из ViewModel.

Создание взаимосвязи View – View Model.

- Direct.
- View first.
- ViewModel first.

Создание взаимосвязи View – View Model.

Direct

- Создать ViewModel
- Создать View
- Связать их

Варианты:

  - Установить DataContext.
  - Тоже самое через конструктор View.
  - Через Binding в xaml

View First

- View может сама создавать или получать для себя ViewModel.
- Логичнее всего делать это в конструкторе.
- Наиболее простой вариант.
- Применимо, когда ViewModel не зависит от других VM.
- Нет исходных данных для передачи во ViewModel.

Варианты реализации ViewModel First.

- Фреймверки.
  - Декларируется VM. Фреймворк сам сопоставляет необходимую View производит связывание.
- Использование DataTemplate

Использование DataTemplate

```
<DataTemplate DataType="{x:Type
  itemViewModel:ModelClassVM}">
  <viewVmRelation:ModelView />
</DataTemplate>
```

Доступ к View из View Model

- Неполная поддержка контролами MVVM паттерна.
- Необходимость вызовов методов View или ссылок на View в реализации методов ViewModel.

Как показать MessageBox с ошибкой из ViewModel?

Вызов методов View из VM. Варианты решения.

- Через интерфейс.
- С помощью событий.
- Через статический класс/Менеджер.

Вызов методов View из VM. Через интерфейс.

- Описать некий интерфейс IView с методами нужными для вызова во ViewModel.
- Завести свойства типа IView в VM.
- Инициализировать это свойство при создании VM, например, с помощью IoC контейнера.

(+) Простота использование;

(-) Захламление кода еще и интерфейсами (кроме V, VM и M)

(-) Жесткое существование привязанной к VM View.

Вызов методов View из VM. Через события.

- Создать события во ViewModel.
- Подписаться во View на события VM. Либо в конструкторе либо с помощью OnDataContextChange.
- Реализовать обработчики во View.

(+) Максимальная автономность VM;

(-) Много кода для вызова даже одного простого метода View.

(-) Необходимость следить за корректной отпиской от событий.

Catel Framework

Общее описание

Поддерживаемые платформы

- WPF
- Silverlight 4
- Silverlight 5
- Windows Phone 7
- Windows Phone 8
- Windows RT

Что предоставляет Framework?

- Базовые классы для Model и ViewModel.
- Catel Property.
- Система валидации.
- Система сообщений для ViewModel’ий.
- Вспомогательные сервисы.
- IoC собственный. Поддержка сторонних.
- Классы создания UI DataWindow и UserControl.
- Behaviors для создания UI на базе стандартных компонент.

Базовые классы

- ModelBase
  - INotifyPropertyChanged, IDataErrorInfo, IEditableObject
- ViewModelBase
- Command

Catel Property.

```
public string SimpleProperty
{
  get { return GetValue<string>(SimplePropertyProperty);}
  set { SetValue(SimplePropertyProperty, value); }
}
```

```
public static readonly PropertyData SimplePropertyProperty =
   RegisterProperty("SimpleProperty", typeof(string), "Простое
   свойство");
```

Ускорение разработки

- Шаблоны.
- Code snippet.
- Плагин Resharp’ера.
- Пробрасывание свойств через атрибуты.

Примеры

Система взаимодействия VM

```
[InterestedIn(typeof(FamilyViewModel))]
public class PersonViewModel : ViewModelBase


protected override void OnViewModelPropertyChanged(IViewModel
   viewModel, string propertyName)
{
   …
}
protected override void OnViewModelCommandExecuted (IViewModel
   viewModel, ICatelCommand command, object commandParameter)
{
   …
}
```

Сервисы.

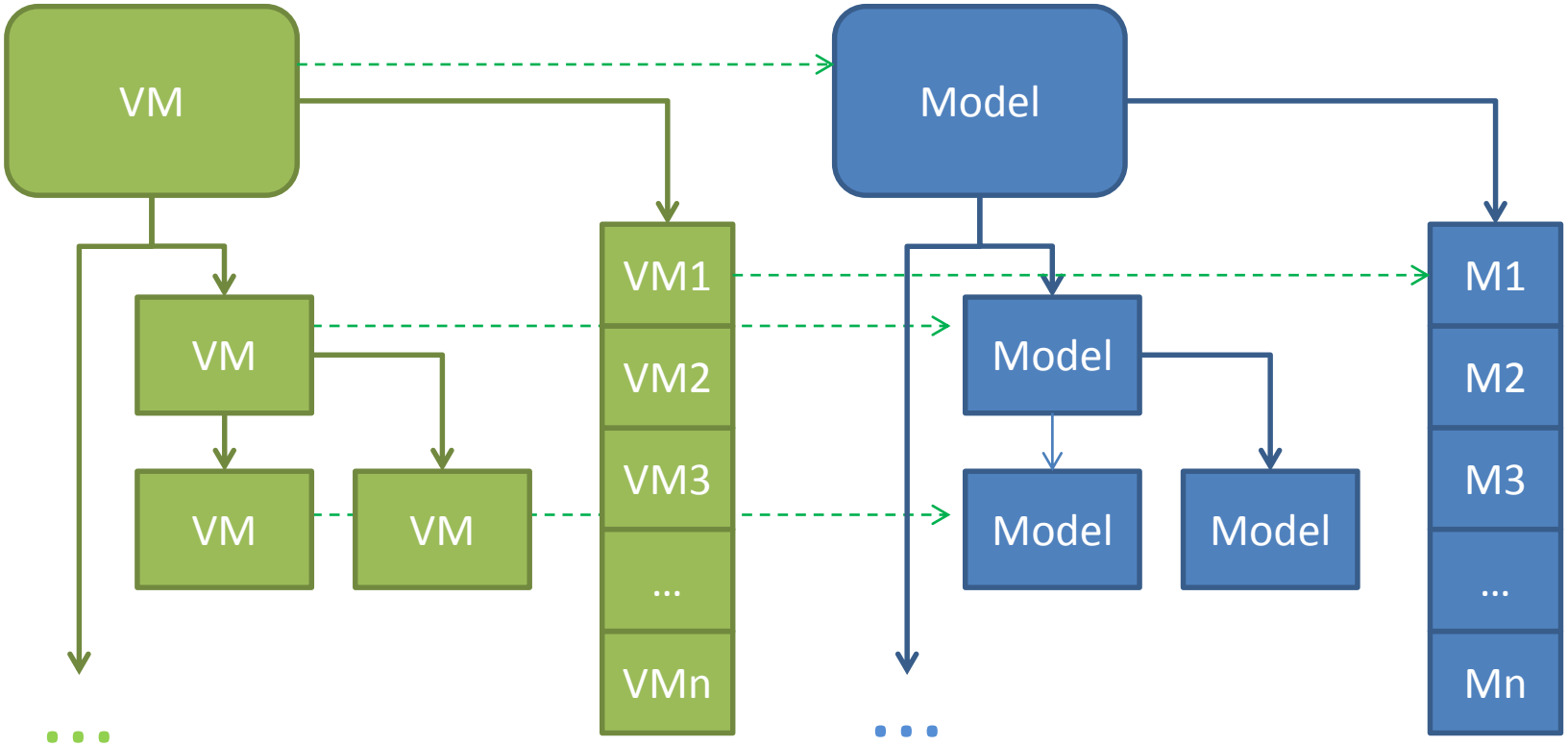
- IAccelerometerService
- ICameraService
- ICompassService
- IGyroscopeService
- ILocationService
- IMessageService
- INavigationService
- IOpenFileService
- IPleaseWaitService
- IProcessService
- ISaveFileService
- IUIVisualizerService
- IVibrateService

Использование сервисов

```
public class PersonViewModel : ViewModelBase
{
  private readonly IUIVisualizerService uiVisualizerService;

  public PersonViewModel(Person person,
         IUIVisualizerService uiVisualizerService)
  {
    this.uiVisualizerService = uiVisualizerService;
  }
  public void OnShow()
  {
    uiVisualizerService.ShowDialog(newViewModel);
  }
}
```

Иерархии.



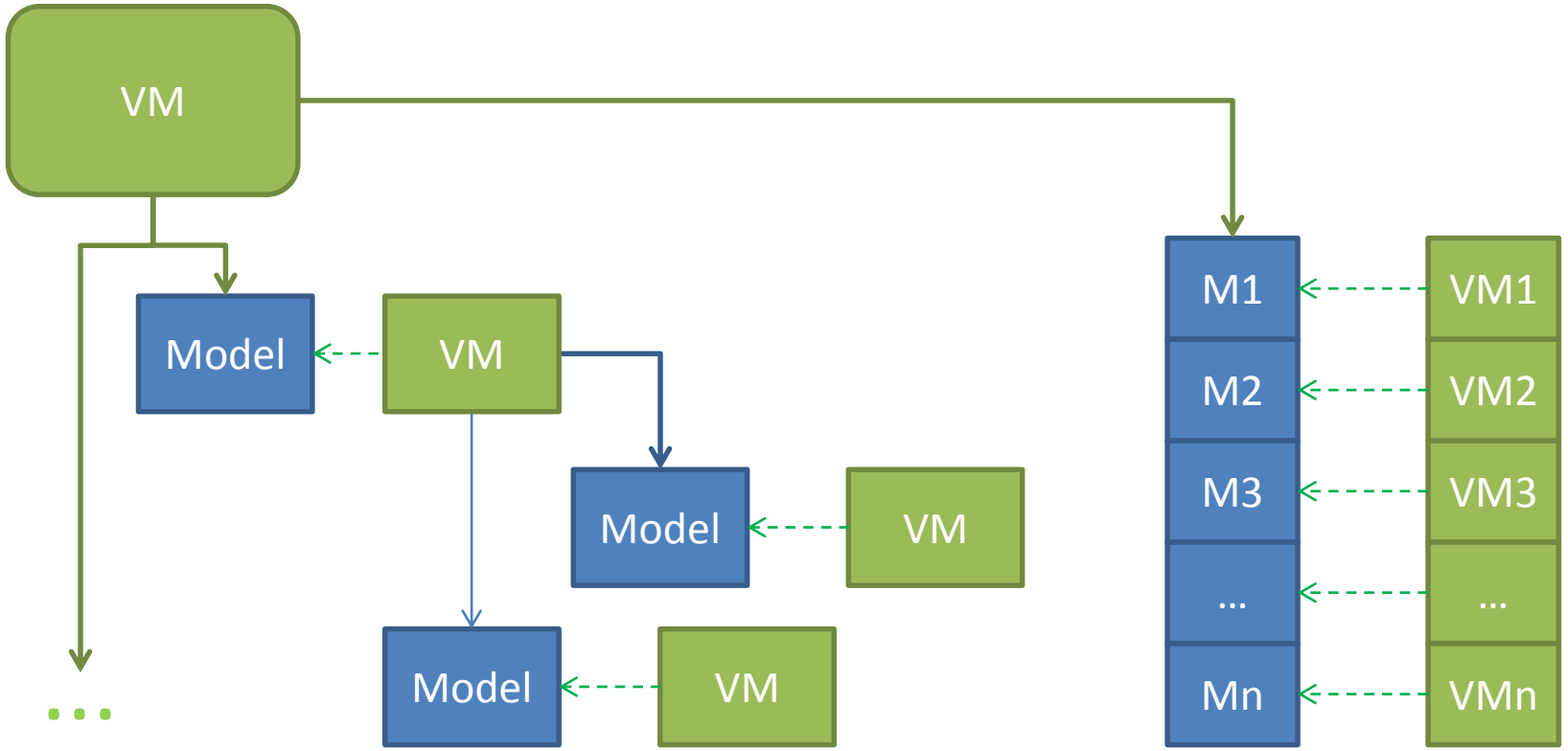
Проблема вложенных контролов.

- Уникальная способность Catel User Control создавать ViewModel динамически из Model.

Алгоритм работы:

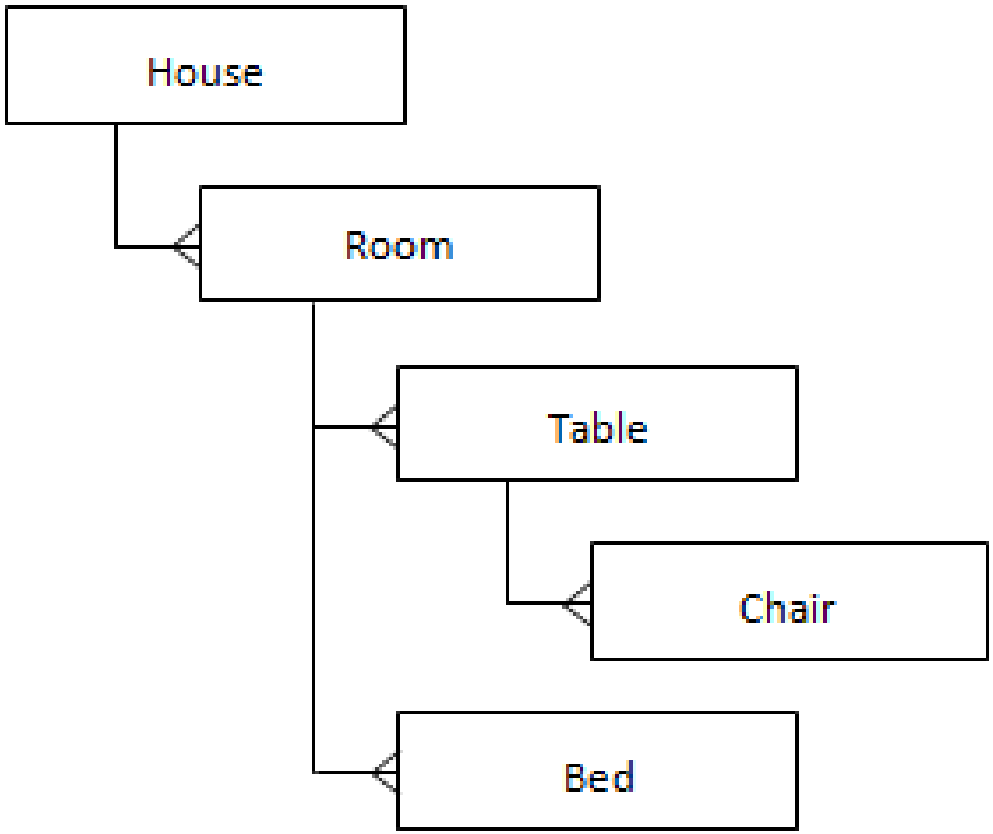
- Задается тип ViewModel для контрола.
- Если у контрола нет DataContext он создает его по типу.
- Если есть, проверяет соответсвует ли он типу VM.
- Если не соответсвует пытается найти конструктор с помощью которого можно создать VM используя объект DataContext как параметр.

Иерархия с Catel



Пример приложения.

- Небольшой пример приложения Catel



Заключение.

- MVVM может очень полезный.
- Использование паттерна не снимает необходимость правильного проектирования.
- Хороший фреймворк может серьезно снизить сложность реализации паттерна.

Спасибо за внимание!