dotnetconf

11-я конференция .NET разработчиков
31 октября 2015
dotnetconf.ru

Разработка WPF приложений в стиле ViewModel First

Денис Цветцих
АстроСофт
http://www.astrosoft.ru/

byndyusoft

APP GRA NULA

Почему это актуально

- WPF все ещё жив ☺️
- MVVM – тема множества докладов и статей
- Множество разных реализаций
  - от MVVM.Light
  - до Prism с мануалом на 250 страниц

В чем проблема?

Много ***разных*** реализаций MVVM

- Непонятно, чем они отличаются
- Непонятно, какую из них использовать и в каких случаях

Почему я здесь

- Накоплен интересный опыт участия в WPF проектах
- Есть опыт, связанный с фреймворками:
  - Использование сторонних
  - Доработка сторонних
  - Изобретение своего MVVM фреймворка ☺️

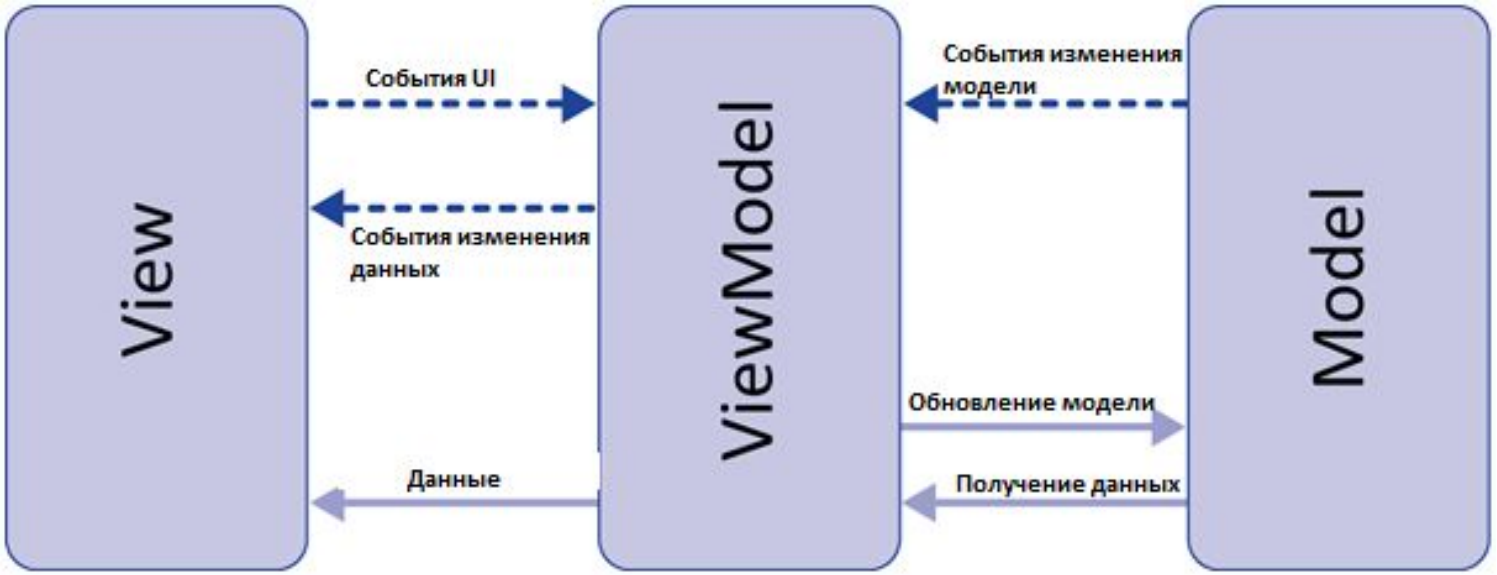
Опрос

- Кто собирается написать WPF приложение?
- Кто при этом использовал какие-нибудь MVVM библиотеки/фреймворки?
- Кто их вас изобрел свой собственный MVVM ~~велс~~ мворк?

О чем мы поговорим?

- Что такое MVVM?
- Какими бывают подходы к его реализации?
- Как отличаются организация CompositeUI и дочерних окон в разных подходах?
- Какой подход лучше использовать?
- Где найти реализацию этого подхода?
- Рекомендации на основе нашего опыта

Model-View-ViewModel



Что со всем этим делать?

Как решать типовые задачи?

- CompositeUI (MasterDetail форма)
- Навигация (дочерние окна)

Нам нужен MVVM фреймворк!

Бывает 2 типов:

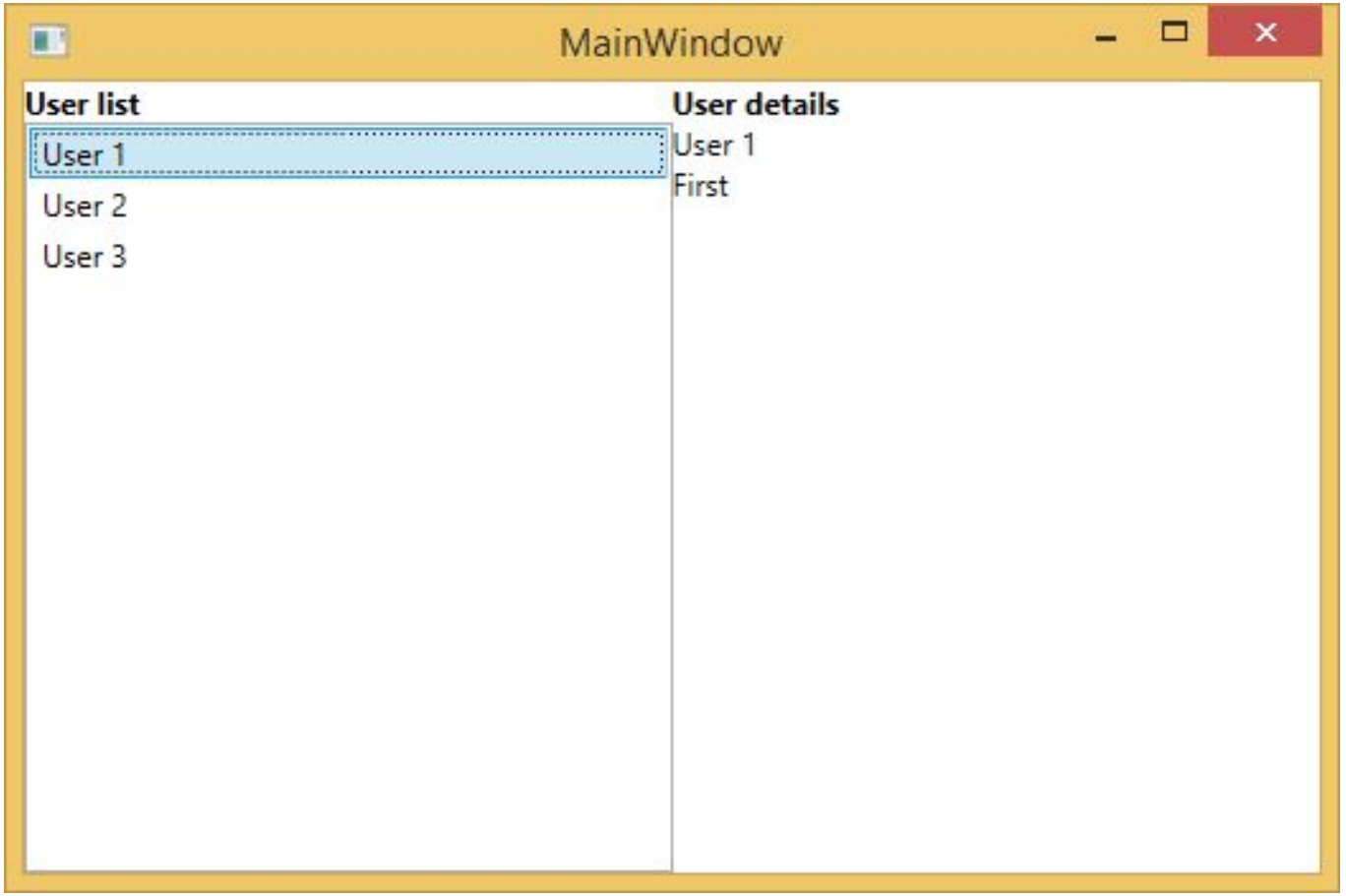
- ViewFirst
- ViewModelFirst

Это не разные реализации паттерна MVVM

Это разные подходы к решению типовых задач с использованием MVVM

COMPOSITE UI

Master Detail



ViewFirst (Prism)

Отображение нового региона:

1) Создать View
2) Создать ViewModel для View
3) View.DataContext = ViewModel
4) Инициализировать ViewModel

MainWinow

```
<Grid>

    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <view:UserListView Grid.Column="0" />

    <view:UserDetailsView Grid.Column="1" />

</Grid>
```

UserListView

```
<Grid>

    <TextBlock Grid.Row="0" Text="User list" FontWeight="Bold" />

    <ListBox Grid.Row="1" x:Name="UserListBox"
        SelectedItem="{Binding SelectedUser, Mode=TwoWay}"
        ItemsSource="{Binding Users}">

        <ListBox.ItemTemplate>
            <DataTemplate>
                <TextBlock Text="{Binding FirstName}" />
            </DataTemplate>
        </ListBox.ItemTemplate>

    </ListBox>
</Grid>

public UserListView()
{
    DataContext = new UserListViewModel();
}
```

UserDetailsView

```
<StackPanel Orientation="Vertical">

    <TextBlock Text="User details" FontWeight="Bold" />

    <TextBlock Text="{Binding User.FirstName}" />
    <TextBlock Text="{Binding User.LastName}" />
</StackPanel>

public UserDetailsView()
{
    DataContext = new UserDetailsViewModel();
}
```

UserListViewModel

```
public class UserListViewModel : ViewModel
{
    private User _selectedUser;

    public IEnumerable<User> Users { get; } // инициализация

    public User SelectedUser
    {
        get { return _selectedUser; }
        set
        {
            _selectedUser = value;
            OnPropertyChanged();
        }
    }
}
```

UserDetailsViewModel

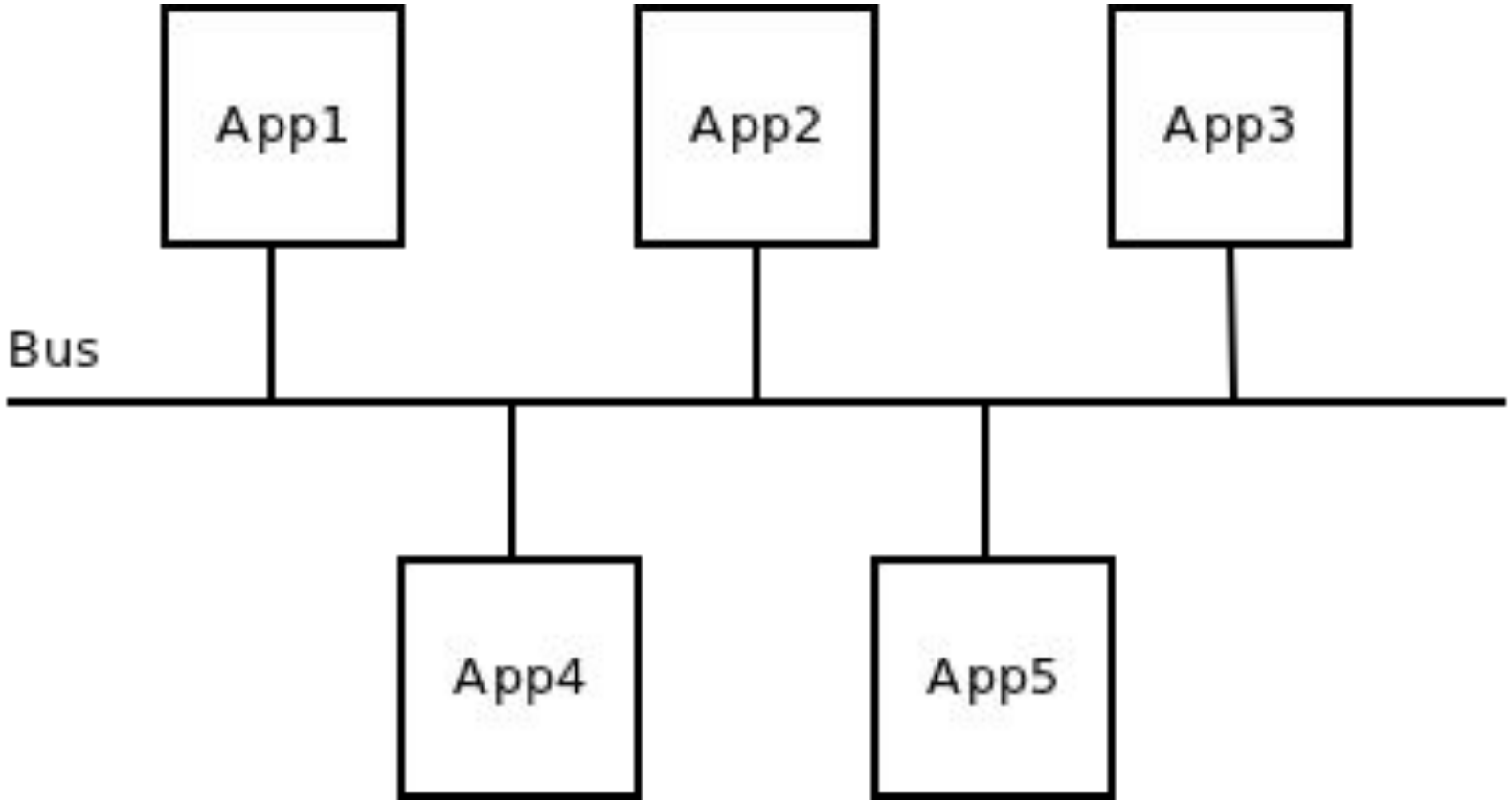
```
public class UserDetailsViewModel : ViewModel
{
    private User _user;

    public User User
    {
        get { return _user; }
        set
        {
            _user = value;
            OnPropertyChanged();
        }
    }
}
```

Вопрос

Как сделать так, чтобы
UserListViewModel.**SelectedUser**
синхронизировалось с
UserDetailsViewModel.**User**?

Ответ в стиле ViewFirst – MessageBus

MessageBus



MessageBus

```
public class MessageBus
{
    public static MessageBus Instance = new MessageBus();

    public event EventHandler<UserChangedEventArgs> SelectedUserChanged;

    public void OnSelectedUserChanged(User user)
    {
        SelectedUserChanged?.Invoke(this, new UserChangedEventArgs(user));
    }
}

public class UserChangedEventArgs : EventArgs
{
    public UserChangedEventArgs(User user)
    {
        User = user;
    }
    public User User { get; }
}
```

UserListViewModel

```
public class UserListViewModel : ViewModel
{
    public User SelectedUser
    {
        get { return _selectedUser; }
        set
        {
            _selectedUser = value;
            OnPropertyChanged();

            MessageBus.Instance.OnSelectedUserChanged(value);
        }
    }
}
```

UserDetailsViewModel

```
public class UserDetailsViewModel : ViewModel
{
    public UserDetailsViewModel()
    {
        MessageBus.Instance.SelectedUserChanged +=
        (s, e) => User = e.User;
    }
}
```

Недостатки

1) Используется MessageBus, предназначенный для интеграции систем
2) На широковещательное событие может подписаться любой объект
3) Поведение системы становится запутанным и неочевидным

ViewModelFirst (энтузиасты)

Отображение нового региона:

1) Создать ViewModel

2) Инициализировать ViewModel

3) Создать View для ViewModel

4) View.DataContext = ViewModel

Чистим CodeBehind

```
public UserListView()
{
    DataContext = new UserListViewModel();
}
public UserDetailsView()
{
    DataContext = new UserDetailsViewModel();
}
```

Убираем MessageBus

```
public class UserListViewModel : ViewModel
{
    public User SelectedUser
    {
        get { return _selectedUser; }
        set
        {
            _selectedUser = value;
            OnPropertyChanged();

            MessageBus.Instance.OnSelectedUserChanged(value);
        }
    }
}
```

Убираем MessageBus

```
public class UserDetailsViewModel : ViewModel
{
    public UserDetailsViewModel()
    {
        MessageBus.Instance.SelectedUserChanged +=
        (s, e) => User = e.User;
    }
}
```

Вопрос

Как сделать так, чтобы
UserListViewModel.**SelectedUser**
синхронизировалось с
UserDetailsViewModel.**User**?

Ответ в стиле ViewModelFirst – нам нужна родительская ViewModel

MainWindowViewModel

```
public class MainWindowViewModel : ViewModel
{
    public UserDetailsViewModel UserDetailsViewModel { get; private set; }
    public UserListViewModel UserListViewModel { get; private set; }

    public void Initialize()
    {
        UserListViewModel.PropertyChanged += (s, e) =>
        {
            if (e.PropertyName == "SelectedUser")
                UserDetailsViewModel.User = UserListViewModel.SelectedUser;
        };
    }
}
```

MainWindow

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <views:UserListView
        DataContext="{Binding UserListViewModel}"
        Grid.Column="0" />

    <views:UserDetailsView
        DataContext="{Binding UserDetailsViewModel}"
        Grid.Column="1" />
</Grid>
```

ViewFirst vs ViewModelFirst

	ViewFirst	ViewModelFitst
CompositeUI	да	да
Взаимодействие ViewModel	MessageBus	Родетельская ViewModel

НАВИГАЦИЯ

ViewFirst: показать новый элемент

Дочернее окно, новый таб:

```
Navigation.Show<ViewModel>(Value);
```

или

```
Navigation.Show("View", Value);
```

Аналогично вебу: http://address.ru/?arg=value

ViewFirst предлагает в WPF организовать навигацию аналогично веб-приложению

ViewModelFirst: показать новый элемент

```
var vm = Navigation.Get<ViewModel>();
vm.Arg = Value;
vm.Show();
```

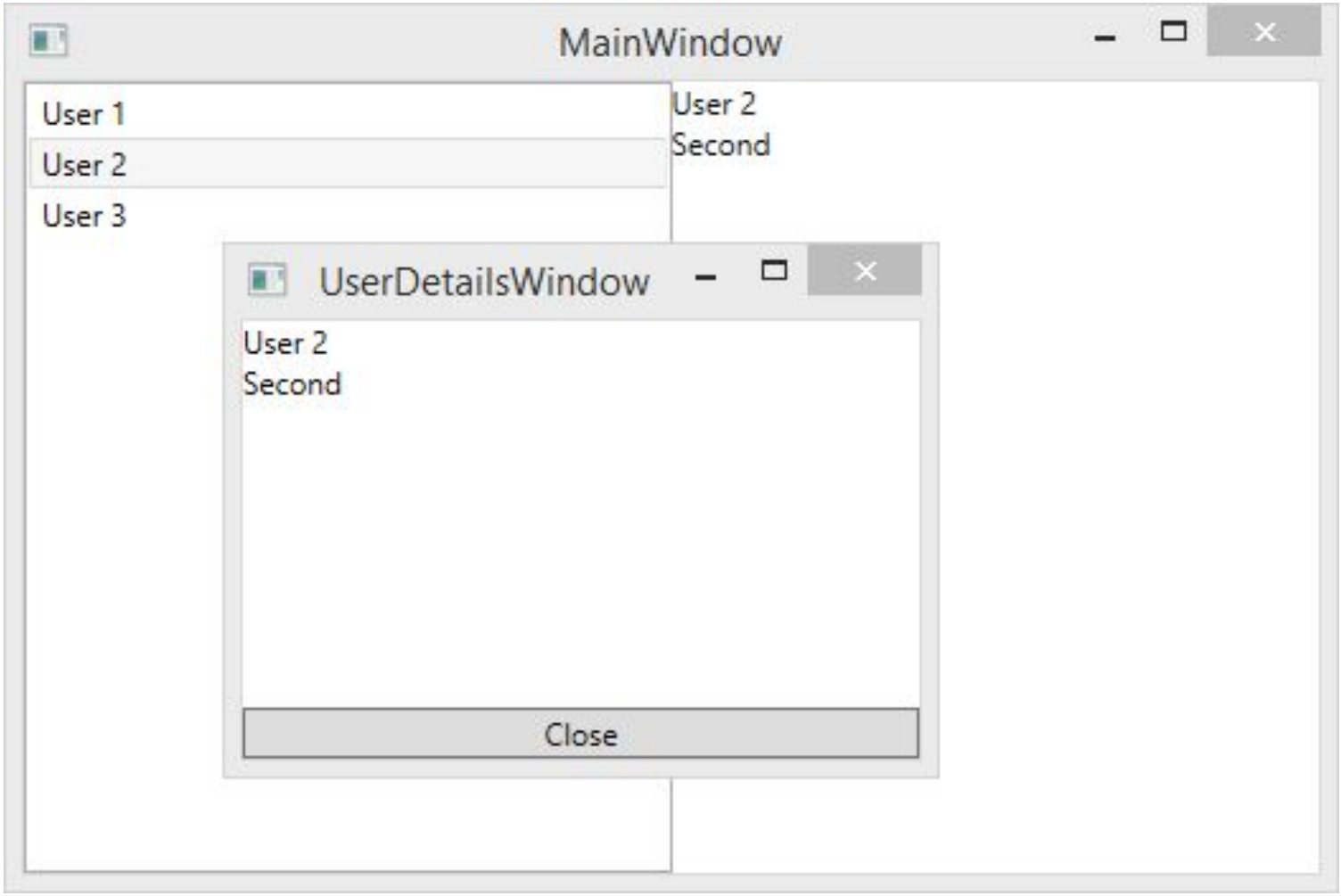
Аналогично окну WPF:

```
var wnd = new Window();
wnd.Arg1 = Value1;
wnd.Show();
```

ViewFirst vs ViewModelFirst

ViewFirst	ViewModelFirst
Создать View	Создать ViewModel
Создать ViewModel	Инициализировать ViewModel
Инициализировать ViewModel	Создать View
View.DataContext = ViewModel	View.DataContext = ViewModel

Дочернее окно



Отображение дочернего окна

```
private void OnShowDetails(User user)
{
    var detailsVM =
   Factory.Resolve<UserDetailsWindowViewModel>();

    detailsVM.User = user;
    detailsVM.Closed +=
        (s, e) => { /* обработка e.DialogResult */ };

    detailsVM.Show();
}
```

ChildViewModel

```
public abstract class ChildViewModel : ViewModel, IChildViewModel
{
    [Dependency]
    public IChildViewModelManager ChildViewModelManager { private get; set;
    }

    public bool IsClosed { get; private set; } // уже закрыли или нет?

    protected void Close()
    {
        if (IsClosed) throw  new InvalidOperationException(“closed");

        IsClosed = true;

       ChildViewModelManager.Close(this);
    }

    public void Show()
    {
        ChildViewModelManager.Show(this);
    }
}
```

ChildViewModelManager

```
public class ChildViewModelManager : IChildViewModelManager
{
    // открытые окна
    private readonly Dictionary<Type, Window> _openedWindows
    = new Dictionary<Type, Window>();

    // по типу ViewModel возвращает View
    [Dependency]
    public IViewTypeResolver ViewTypeResolver
        { private get; set; }

  }
```

ChildViewModelManager: Show

```
private void Show(IChildViewModel viewModel)
{
    // получить тип окна, которое будем открывать
    var windowType = ViewTypeResolver.
     ResolveViewType(viewModel.GetType());

    // создать экземпляр окна
    var window =
     (Window)Activator.CreateInstance(windowType);

     // запомнить, какое окно открываем
     _openedWindows.Add(viewModel.GetType(), window);

    window.DataContext = viewModel;

     // показать окно
     window.Show();
}
```

ChildViewModelManager: Close

```
public void Close(IChildViewModel viewModel)
{
     // какое окно закрываем
     var window = _openedWindows[viewModel.GetType()];

     // убираем из списка открытых
     _openedWindows.Remove(viewModel.GetType());

     // закрываем
     Application.Current.Dispatcher.BeginInvoke(
    new Action(() => window.Close()), null);
}
```

ChildWindow

```
public abstract class ChildWindow : Window
{
    protected override void OnClosing(CancelEventArgs e)
    {
        base.OnClosing(e);

        var viewModel = (ChildViewModel)DataContext;

    // если ViewModel на находится в состоянии «Закрыто»
        if (!viewModel.IsClosed)
        {
            e.Cancel = true; // не закрываем окно

       // запрашиваем изменение состояния ViewModel
       viewModel.Close();
        }
    }
}
```

Особенности ViewModelFirst

Достоинства

- Позволяет реализовать CompositeUI
- Не требует реализации MessageBus
  - Взаимодействие ViewModel более очевидное
  - Нет MessageBus – нет его использования не по назначению
- Позволяет удобно реализовать поддержку дочерних окон

Недостатки

- Не имеет вендорской поддержки

Наш рецепт

- ViewModelFirst
  - Свой велосипед
  - Mugen MVVM Toolkit
- IoC контейнер
- ReactiveUI (ограничено)
  - ReactiveCommand
  - ObservableForProperty
- Отдельная сборка для ViewModel

Материалы по ViewModelFirst

Материалы доклада на GitHub:
https://github.com/denis-tsv/ViewFirst-vs-ViewModelFirst

Курс «Методология синхронной разработки приложений в Microsoft Visual Studio 2010»
www.intuit.ru/studies/courses/2322/622/info

Mugen MVVM Toolkit
http://habrahabr.ru/post/236745/

Спасибо за внимание

Денис Цветцих

АстроСофт

den.tsvettsih@yandex.ru