Real world Windows Phone development

Igor Kulman
igor@kulman.sk
@igorkulman

About Igor Kulman

- Windows Phone and Microsoft Azure developer at Inmite
- Developing for Windows Phone since 2011

- web: http://www.kulman.sk, blog: http://blog.kulman.sk, github: https://github.com/igorkulman

Goal

Develop maintanable and testable Windows Phone applications

Agenda

- Model-View-ViewModel (MVVM)
- Caliburn.Micro
- INotifyPropertyChanged (Fody)
- Nuget
- Portable Class Libraries (PCL)
- Testing (MSTest, Moq)

Model-View-ViewModel

- Design pattern from Microsoft simillar to MVC
- Separating UI, app logic and business logic
- Data binding, messaging, commands
- Typically also DI
- Popular frameworks: MVVMLight, Caliburn.Micro, PRISM

Model-View-ViewModel

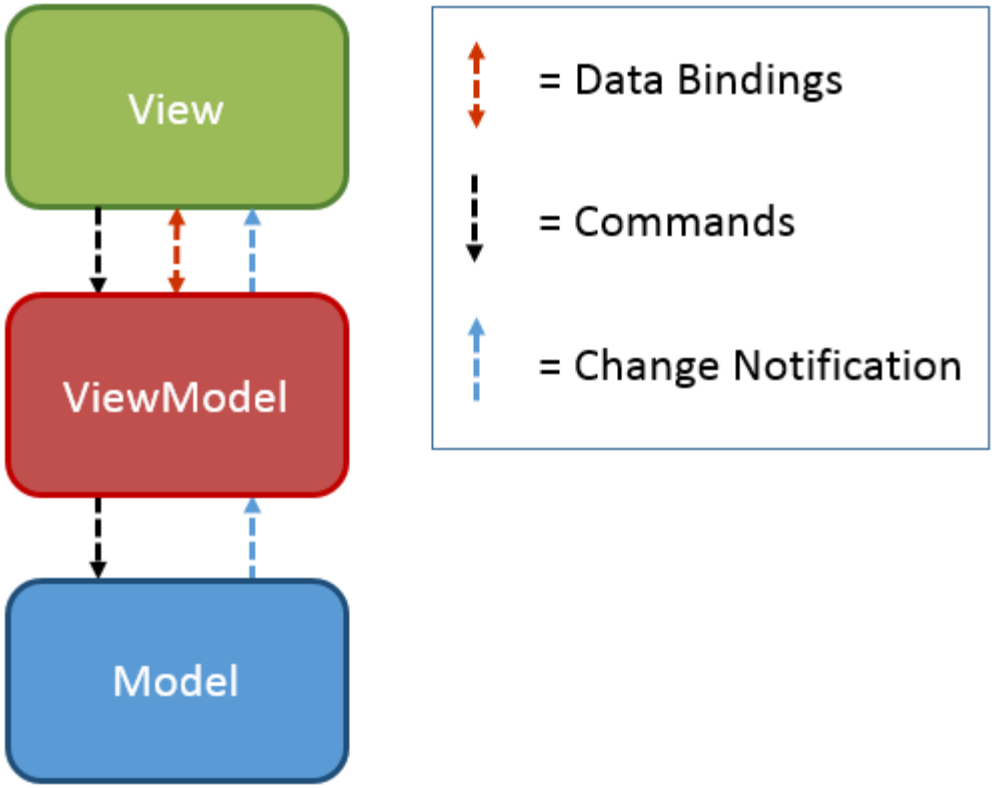
Advantages

- more testable
- more maintanable
- easier to do DI

Notes

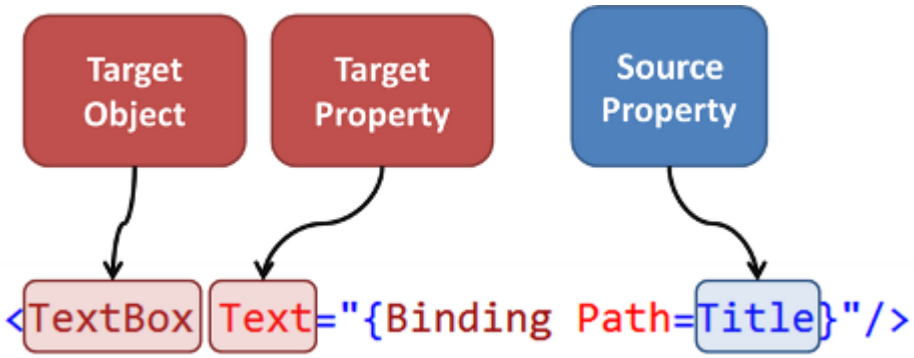
- View-specific stuff should be still handled in code-behind

Model-View-ViewModel



Bindings

- Referencing properties from ViewModel or ther componetns in View
- One-Way, Two-Way



Converters

- Convert data used in bindings from one format to another
- Keep your ViewModel "clean"
- Implement IValueConverter

```
public class DateTimeToLabelConverter: IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        var datetime = (DateTime) value;
        return datetime.Humanize();
    }
```

- Use as StaticResource

```
<phone:PhoneApplicationPage.Resources>
    <converters:DateTimeToLabelConverter x:Key="DateTimeToLabelConverter" />
</phone:PhoneApplicationPage.Resources>
```

INotifyPropertyChanged

- Interface telling the UI that a property in ViewModel has changed
- Necessary for data binding

```
private string _MyProperty;
public string MyProperty
{
    get { return _MyProperty; }
    set
    {
        if (value != _MyProperty)
        {
            _MyProperty = value;
            NotifyPropertyChanged("MyProperty");
        }
    }
}
```

Fody.PropertyChanged

- Fody is a il-weaver for .NET
- Fody.PropertyChanged adds to INotifyPropertyChanged every property so you do not have to

```
[ImplementPropertyChanged]
public class Photo
{
    public string Title { get; set; }
    public string Description { get; set; }
    public string Image { get; set; }
```

Commands

Commands provide a separation between a user interface and logic.

- A command is an object that implements the ICommand interface.
- Generally, it is associated with a function in some code.
- User interface elements bind to commands - when they are activated by the user the command is fired - which calls the associated function.
- Commands know if they are enabled or not.
- A function can disable the command object - automatically disabling any user interface elements associated with it.

Commands

Implement ICommand

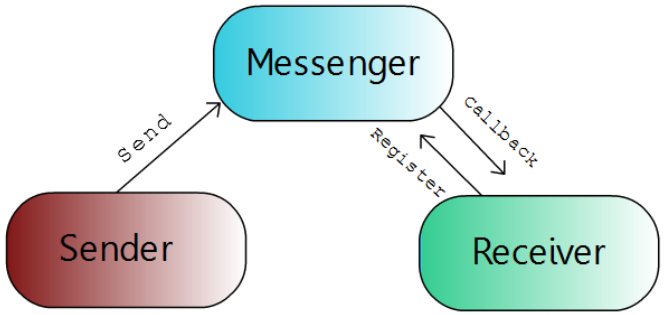
- Execute - method to execute
- CanExecute - determines whether the command can execute in its current state

Binding in XAML

```
<Button Command="{Binding Execute}" Content="Execute me" />
```

Messaging

- Sending messages between ViewModels and Views avoid tight coupling
- Communication between View and ViewModel when a binding cannot be used (e.g: animations)



Nuget

- package manager for .NET project
- open to everyone (my packages)
- web repository, search
- Visual Studio and PowerShell integration
- handling dependencies, upgrades
- restoring packages at build

DEMO

Creating a new Windows Phone project with packages from Nuget

Portable Class Libraries

- Libraries that target more than one .NET Framework platform
- WP8, WP81, WinRT, Silverlight, XBOX,
- Support for Mono (Xamarin.iOS / Android)
- No platform specific stuff (I/O, Settings, etc.)

Portable Class Libraries

When to start with PCL

- more platforms
- no platform specific stuff

Recommendation

- start with a Windows Phone Class Library
- convert to PCL later

DEMO

Adding PCL with business logic to the project

Testing the PCL

- Testing your business logic is a good practice
- Many testing frameworks: MSTest, NUnit, xUnit
- Testing async method is not a problem

MSTest

- built into Visual Studio
- run from Visual Studio, Resharper, FAKE
- decorating methods with attributes

```
[TestClass]
public class FlickrServiceTests
{
    private IFlickrService _flickrService;

    [TestInitialize]
    public void Init()
    {
        _flickrService=new FlickrService();
    }

    [TestMethod]
    public async Task SomePhotosShouldBeFound()
    {
        var res = await _flickrService.Search("prague");
        Assert.IsNotNull(res);
        Assert.IsTrue(res.Any());
    }
}
```

DEMO

Add test project and tests for PCL

Caliburn.Micro

- mature MVVM framework for WPF, Sliverlight, Windows Phone, Windows 8
- based on conventions
- simple DI container
- messaging model based on System.Interactivity (Blend support)
- works with Fody.PropertyChanged

Caliburn.Micro

Bootstrapper

- registered in App.xaml
- app strating point
- register ViewModels and Services
- override things like frame creation

Caliburn.Micro

Conventions

- Views directory with {X}View
- ViewModels directory with {X}ViewModel
- Binding button click event to methods by name
- Binding ListBox item source and selected item by name
- TextBox two way binding by name

Caliburn.Micro

Actions

- used instead of Commands for iteractions from View
- can be bound to any event

```
<Button Content="Remove"
        cal:Message.Attach="[Event Click] = [Action Remove($dataContext)]" />
```

- Parameters: $eventArgs, $dataContext, $source, $view, $executionContext, $this

Caliburn.Micro

Services

- INavigationService

```
_navigationService.UriFor<DetailViewModel>().WithParam(l=>l.Url,SelectedPhoto.Url).Navigate();
```

- IEventAggregator

```
_eventAggregator.Publish(new StartAnimationMessage
                         {
                             From = HeadingDifference,
                             To = 360+res
                         });
```

INavigationService

- No need to register, always available
- ViewModel first
- (Multiple) navigation parameters

```
_navigationService.UriFor<DetailViewModel>().WithParam(l=>l.Url,SelectedPhoto.Url).Navigate();
```

IEventAggregator

- No need to register, always available
- Publishing messages

```
_eventAggregator.Publish(new StartAnimationMessage
                         {
                             From = HeadingDifference,
                             To = 360+res
                         });
```

- Need to subscribe for handling, implement IHandle<T>

```
public partial class CompassExplorerView : PhoneApplicationPage, IHandle<StartAnimationMessage>
{
    public void Handle(StartAnimationMessage message)
    {
        RotateCompass(message.From,message.To);
    }
```

DEMO

Add Caliburn.Micro to project, implement Views and ViewModels

Testing the ViewModel

- Testing ViewModels is "nice to have"
- Mocking services, testing just the ViewModel
- Mocking framework: Moq

Moq

- Mocking an interface

```
var navigationServive = Mock.Of<INavigationService>();
```

- Mocking with distinct behaviour

```
var flickrService = new Mock<IFlickrService>();
flickrService.Setup(s => s.Search("brno")).Returns(Task.FromResult(new List<Photo>()));
```

DEMO

Add tests for ViewModel

And design matters!

Do not be one of the boring black pivot apps

Resources

Demo source code:

- https://github.com/igorkulman/muni-talk-2014

Fody

- http://blog.kulman.sk/inotifypropertychanged-the-easy-way-in-windows-phone-and-windows-8/

Caliburn.Micro

- http://wp.qmatteoq.com/first-steps-with-caliburn-micro-in-windows-phone-8-the-theory/