Apps part I

Rasmus Lystrøm
External Associate Professor
ITU

Agenda

XAML and the Universal Windows Platform

XAML = eXtensible Application Markup Language

Windows Desktop (WPF)

Windows Universal (anything)

Xamarin.Forms (iOS, Android, Windows)

Silverlight (web)

XAML

Markup language for declaratively designing and creating application UIs

XAML maps XML markup to objects in the .NET Framework

Every tag maps to a class and every attribute to a property

Markup and procedural code are peers in functionality and performance

Code and markup are both first class citizens

Consistent model between UI, documents, and media

Compiled to code

XAML Markup vs. Code

```
<Button Width="100">OK
    <Button.Background>
        Purple
    </Button.Background>
</Button>
```


```
var button = new Button();
button.Content = "OK";
button.Background = new SolidColorBrush(Colors.Purple);
button.Width = 100;
```

MainPage.xaml

```
<Page
    x:Class="App.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid>
        <StackPanel>
            <Ellipse Name="Light" Fill="Red"
                     Height="200" Width="200" Margin="50" />
            <Button Width="150"
                    Content="Change Lights"
                    HorizontalAlignment="Center"
                    Click="Button_Click" />
        </StackPanel>
    </Grid>
</Page>
```

MainPage.xaml.cs

```
namespace App
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            var current = Light.Fill as SolidColorBrush;

            if (current.Color == Colors.Red)
                Light.Fill = new SolidColorBrush(Colors.Green);
            else
                Light.Fill = new SolidColorBrush(Colors.Red);
        }
    }
}
```

XAML DEMO

XAML + code-behind

```
    {
        Light.Fill = new SolidColorBrush(Colors.Red);
    }
}
```

Image source: http://lazergaze.tumblr.com/post/26333564955

The Model-View-ViewModel Pattern

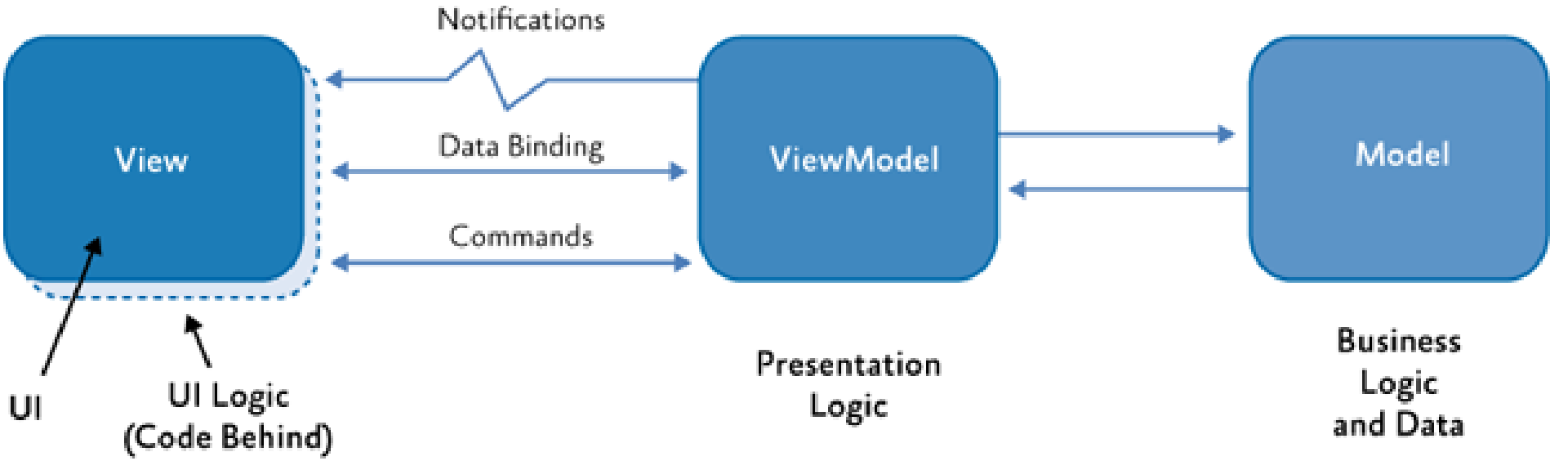
Separation of logic and presentation

Having event handlers in the code-behind is bad for testing, since you cannot mock away the view

Changing the design of the view often also requires changes in the code, since every element has it's different event handlers

The logic is tightly bound to the view. It's not possible to reuse the logic in an other view

MVVM


Notifications
View
Data Binding
ViewModel
Model
Commands
UI
UI Logic
(Code Behind)
Presentation
Logic
Business
Logic
and Data

MVVM DEMO

MVVM: Pieces of the puzzle

There is conceptually only ever one MODEL

Code in code-behind should be ABSOLUTELY MINIMAL

A ViewModel should ALWAYS implement INotifyPropertyChanged

A ViewModel may be used for more than one view

MVVM: Tips

Do

M

Te

Image source: https://dirtyhands.wordpress.com