

Module 13

"MVVM Problems and Patterns"



TEKNOLOGISK
INSTITUT

Agenda

- ▶ **MVVM Frameworks**
- ▶ Patterns for Common Problems
- ▶ Concluding Remarks

The Jungle of MVVM Frameworks

- ▶ Caliburn
- ▶ Catel
- ▶ Cinch
- ▶ ClientUI
- ▶ Excalibur
- ▶ **MVVM Light**
- ▶ Prism
- ▶ Simple MVVM
- ▶ Vidyano
- ▶ ... + many more

MVVM Light Toolkit

- ▶ Developed by Laurent Bugnion of GalaSoft
 - <http://www.galasoft.ch/mvvm/>
- ▶ Patterns
 - ViewModel Locator / IoC
 - Stateful and Stateless ViewModel
 - Message Bus
 - View Service
 - ...
- ▶ Provides
 - Project Templates
 - Infrastructure and Helper Classes
 - Snippets
 - NuGet packages



Agenda

- ▶ MVVM Frameworks
- ▶ **Patterns for Common Problems**
- ▶ Concluding Remarks

Stateful ViewModel Pattern

- ▶ Controls do not have state – ViewModel does!
- ▶ Easier to test
- ▶ Decouples business logic from UI
- ▶ MVVM Light Toolkit provides NuGet package
 - **ViewModelBase** + snippets

```
public class MainViewModel : ViewModelBase
{
    public string FirstName
    {
        get => firstName;
        set
        {
            ...
            RaisePropertyChanged();
        }
    }
}
```

Message Bus Pattern

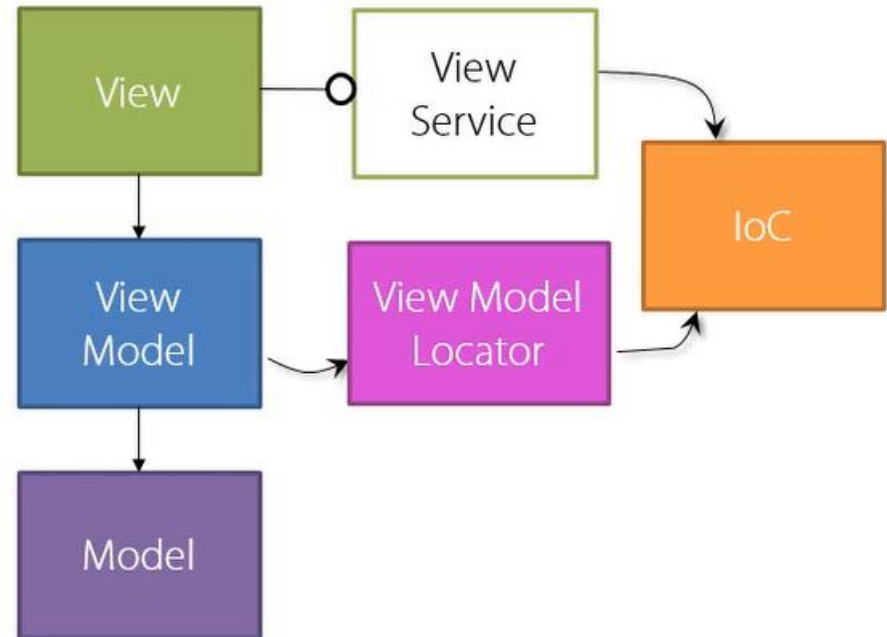
- ▶ Need to synchronize viewmodels
 - Without tightly coupling them!
 - **Messenger** provides loosely coupled communication
- ▶ Note that WPF 4.5 provides "live shaping"
 - **CollectionViewSource.**
 - **IsLiveSortingRequested**
 - **IsLiveFilteringRequested**
 - **IsLiveGroupingRequested**

Inversion of Control Pattern

- ▶ It is really helpful to make use of an IoC container
 - Unity
 - Simpleloc
 - ...
- ▶ This will allow simple setup of many nice features, e.g.
 - Design-mode vs. runtime models, view models and services
 - View Service Pattern
 - ...
- ▶ MVVM Light Toolkit provides the **SimpleIoc** helper

View Services Pattern

- ▶ Cardinal rule of MVVM
 - Never look "up"...
- ▶ ViewModel should never call View methods, e.g.
 - Prompts
 - Message boxes etc.
- ▶ Solution is to create View Services to break potential circular dependency between View and ViewModel



Attached Behaviors Pattern

- ▶ Nikhil Kothari
 - In 2008 discovered the sheer power of attached properties
 - Attaches “unavailable” behavior to UI elements, e.g. **Command**
- ▶ Blend Team went to improve it to **Behavior<T>** class
 - In **System.Windows.Interactivity** namespace
 - **EventTrigger**
 - **InvokeCommandAction**
- ▶ Examples may include
 - Providing **Command** property
 - Converting events to commands
 - Drag ‘n Drop
 - Setting focus to elements
 - ... + many more
- ▶ MVVM Light provides **EventToCommand** behavior similar to **InvokeCommandAction**

Agenda

- ▶ MVVM Frameworks
- ▶ Patterns for Common Problems
- ▶ **Concluding Remarks**

Summarizing...

- ▶ Many variations and alternative sub-patterns exist
- ▶ MVVM keeps a clear and clean separation
- ▶ MVVM is in many ways like “re-learning” WPF
- ▶ MVVM is often difficult upon new first encounters
- ▶ MVVM frameworks support sets of sub-patterns

- ▶ Choose the approach which suits YOU
- ▶ But stay as consistent as possible...
 - Uncle Google might be your enemy here

MVVM is "going global..." 😊

► Works for any XAML-based formalism

- WPF
- UWP
- Xamarin.Forms

► Adapted by

- KnockoutJS
- Angular

Choose a ticket class:

```
<select data-bind="options: tickets,  
optionsCaption: 'Choose...',  
optionsText: 'name',  
value: chosenTicket"></select>
```

Binding attributes
declaratively link
DOM elements
with model
properties

```
<button data-bind="enable: chosenTicket,  
click: resetTicket">Clear</button>
```

```
<p data-bind="with: chosenTicket">  
You have chosen <b data-bind="text: name"></b>  
( $<span data-bind="text: price"></span> )  
</p>
```

```
<script>
```

```
function TicketsViewModel() {  
    this.tickets = [  
        { name: "Economy", price: 199.95 },  
        { name: "Business", price: 449.22 },  
        { name: "First Class", price: 1199.99 }  
    ];  
    this.chosenTicket = ko.observable();  
    this.resetTicket = function() { this.chosenTicket(null) }  
}
```

Your view model
holds the UI's
underlying data
and behaviors

```
    ko.applyBindings(new TicketsViewModel());  
</script>
```

Activates Knockout

Summary

- ▶ MVVM Frameworks
- ▶ Patterns for Common Problems
- ▶ Concluding Remarks



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Ringgårdsvej 4A

8270 Højbjerg

Denmark