

Module 7

"Introducing the MVVM Design Pattern"



TEKNOLOGISK
INSTITUT

Agenda

- ▶ **Introduction to MVVM**
- ▶ Creating an MVVM Application
- ▶ Common Techniques

ViewModel Pattern History

► Presentation Model

- Martin Fowler 2004
- Separation pattern
- Remove state and behavior from the view
- <http://www.martinfowler.com/eaDev/PresentationModel.html>



► Model-View-ViewModel

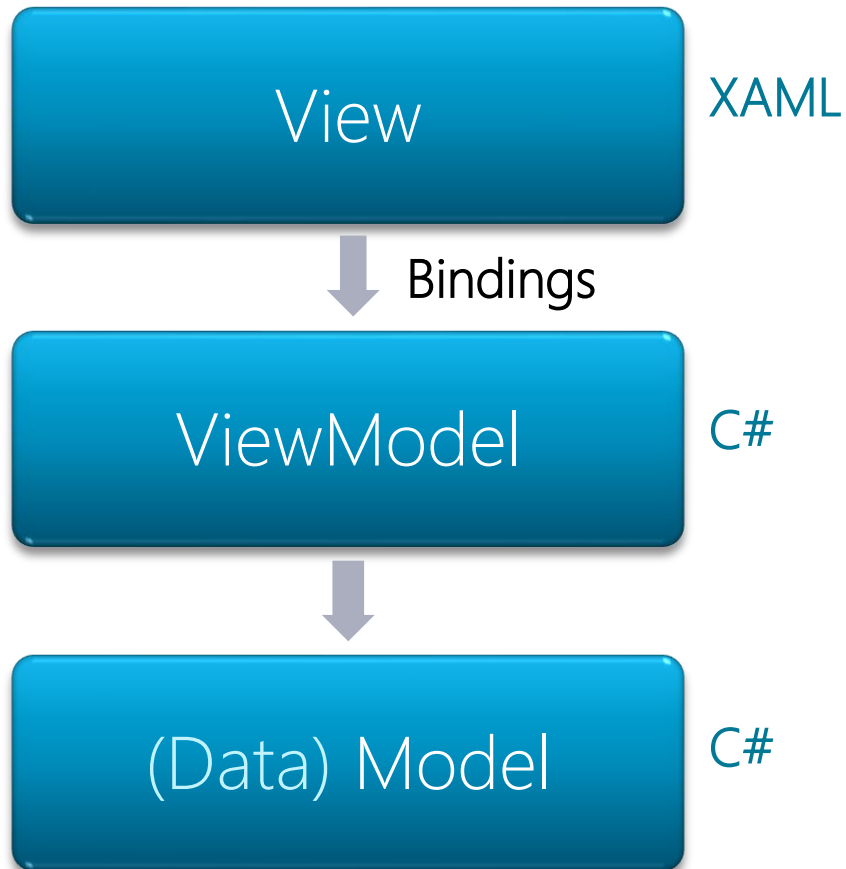
- John Gossman 2005
- Presentation Model specialized to XAML
- Crucially based on CLR data-binding
- <http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx>



Model-View-ViewModel Components

- ▶ Commonly known as MVVM
- ▶ Model
 - Business objects
 - Data
- ▶ View
 - Presentation
- ▶ ViewModel
 - Abstraction of view, adapter between Model and View
 - View's state and behavior

Model-View-ViewModel



- ▶ Separation between presentation and application logic
- ▶ The ViewModel is an abstraction of the View
- ▶ Depends heavily on **data binding** and **command binding**
- ▶ Reflects the architecture of WPF itself
- ▶ WPF, Blend etc. are built with MVVM

Goals of MVVM

- ▶ Separation of Concern
- ▶ Loose coupling
- ▶ Maintainability
- ▶ Testability
- ▶ “Blendability”
- ▶ ...

Family of Patterns

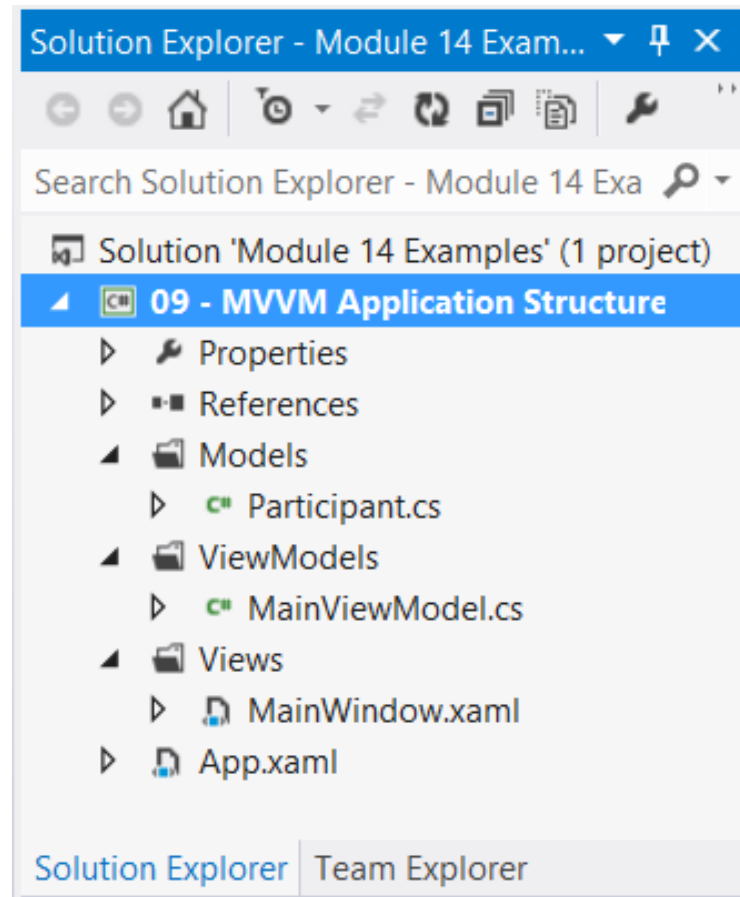
- ▶ MVVM is a set of guidelines open for interpretation
- ▶ How to pair view and viewmodel?
- ▶ How to process events?
- ▶ How to communicate between viewmodels?
- ▶ ...
- ▶ Purists vs. Pragmatics
- ▶ MVVM frameworks support these guidelines

Agenda

- ▶ Introduction to MVVM
- ▶ **Creating an MVVM Application**
- ▶ Common Techniques

MVVM Application Structure

- ▶ Create a new WPF application
 - Models
 - ViewModels
 - Views



The Model

- ▶ Purpose
 - Storing and represent application data and domain objects
 - Raise change notifications
 - Perform validation

```
class Participant : INotifyPropertyChanged, IDataErrorInfo
{
    public string FirstName { ... }
    public string LastName { ... }
    ...
}
```

- ▶ Implement
 - **INotifyPropertyChanged**
 - **IDataErrorInfo** (or **INotifyDataErrorInfo**), if needed



The ViewModel

► Purpose

- Expose data to view by presentation and manipulation
- Facilitate application interaction logic
- Respond to user interaction

```
class MainViewModel : INotifyPropertyChanged
{
    public Participant ModelParticipant { ... }
    ...
}
```

► Implement

- Properties and commands
- **INotifyPropertyChanged** on the exposed properties
 - E.g. the model object



The View

- ▶ Purpose
 - Provide user interface controls only
 - Agnostic of data origin

```
<Grid DataContext="{StaticResource MainViewModel}">
    ...
    <TextBlock Grid.Row="0" Grid.Column="0">First Name:</TextBlock>
    <TextBox Grid.Row="0" Grid.Column="1"
        Text="{Binding ModelParticipant.FirstName,
            ValidatesOnDataErrors=True}"></TextBox>
    ...
</Grid>
```

- ▶ Gets updated through **DataContext** bindings



Pairing up the View and ViewModel

- ▶ Different approaches exist

- XAML
- Code-behind
- Data Template
- ViewModel Locator
- Inversion of Control container
- ...

"View First"

"ViewModel First"

- ▶ Many MVVM frameworks use variations of
ViewModel Locator

Communication Between View and ViewModel

- ▶ Communication between view and view model is facilitated by binding
 - Events
 - Commands

```
<Button Command="{Binding SaveParticipantCommand}">Save</Button>
```

```
class MainViewModel : INotifyPropertyChanged
{
    public ICommand SaveParticipantCommand
    {
        get { return _saveParticipantCommand; }
    }
    private ICommand _saveParticipantCommand;
    ...
}
```



Agenda

- ▶ Introduction to MVVM
- ▶ Creating an MVVM Application
- ▶ **Common Techniques**

RelayCommand

- ▶ You will be defining tons of commands
 - Create reusable helper commands as starting points
- ▶ **RelayCommand** (a.k.a. **DelegateCommand**)

```
public class RelayCommand : ICommand
{
    private readonly Predicate<object> _canExecute;
    private readonly Action<object> _executeAction;
    ...
    public RelayCommand( Action<object> execute,
                        Predicate<object> canExecute = null )
    {
        ...
    }
}
```



ViewModelBase

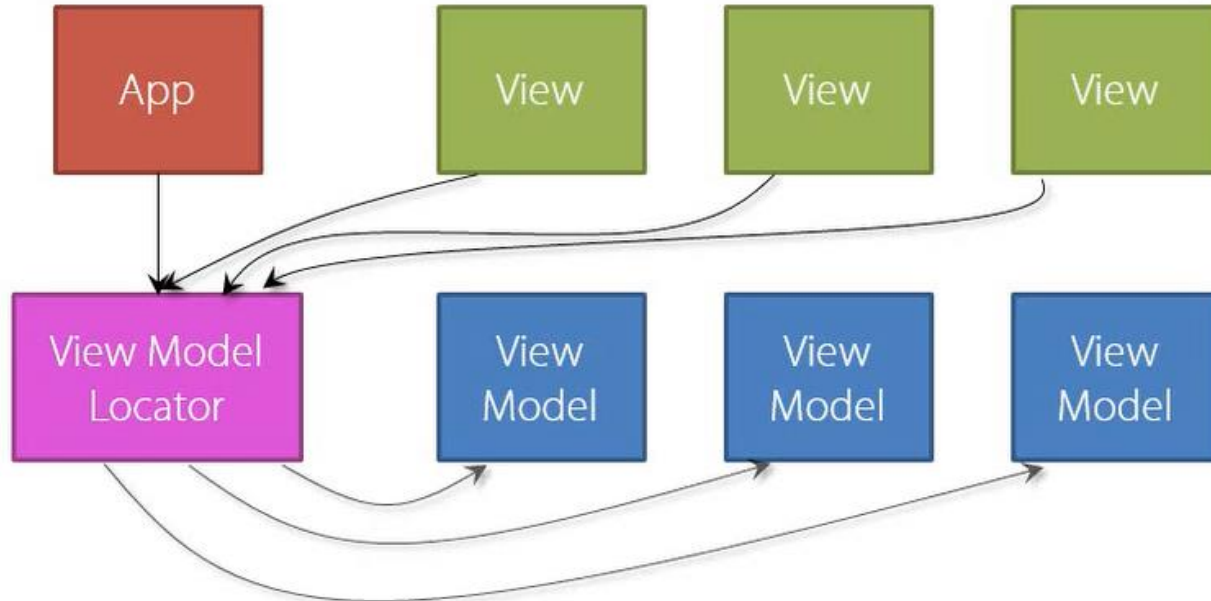
- ▶ All view models will always implement
 - **INotifyPropertyChanged**
 - checking and debug features
- ▶ Share this code from a common base class

```
public class ViewModelBase : INotifyPropertyChanged
{
    ...
}
```

- ▶ Sometimes people similarly define a **ModelBase**
 - A matter of preference



ViewModel Locator Pattern



- ▶ ViewModelLocator instance usually defined in **App.xaml**
- ▶ *Alternative: ViewModel First pattern*



Composing Views and ViewModels

- ▶ “Parallel” and compositional View and ViewModel hierarchies
- ▶ Views can be **UserControl** instances
- ▶ Create
 - ViewModel from Model
 - Data templates for view models
- ▶ ViewModel expose Model objects
 - Directly? ~“Pragmatics”
 - Through individual properties? ~“Purists”



Summary

- ▶ Introduction to MVVM
- ▶ Creating an MVVM Application
- ▶ Common Techniques



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

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

Hasselvangel 243

8355 Solbjerg

Denmark