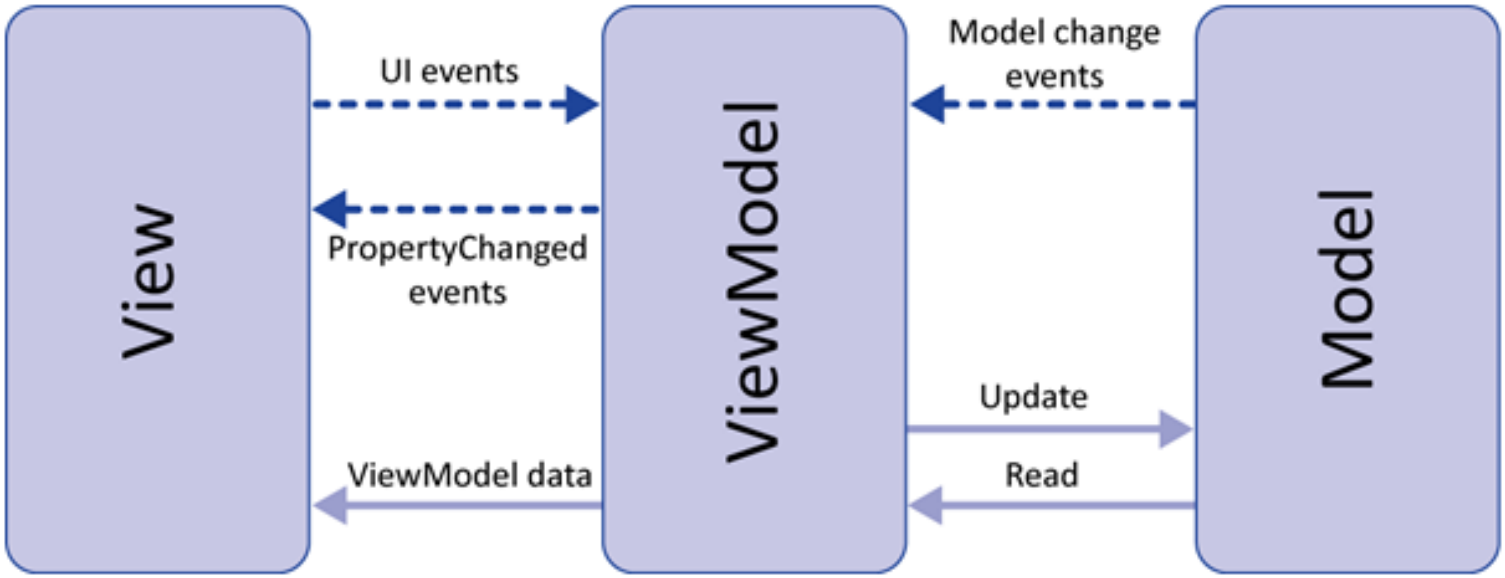
Medii vizuale de programare
Curs 8.1

Conf. dr.ing. GENGE Béla

Universitatea "Petru Maior", Departamentul de Informatica
Tîrgu Mureș, Romania
bela.genge@ing.upm.ro

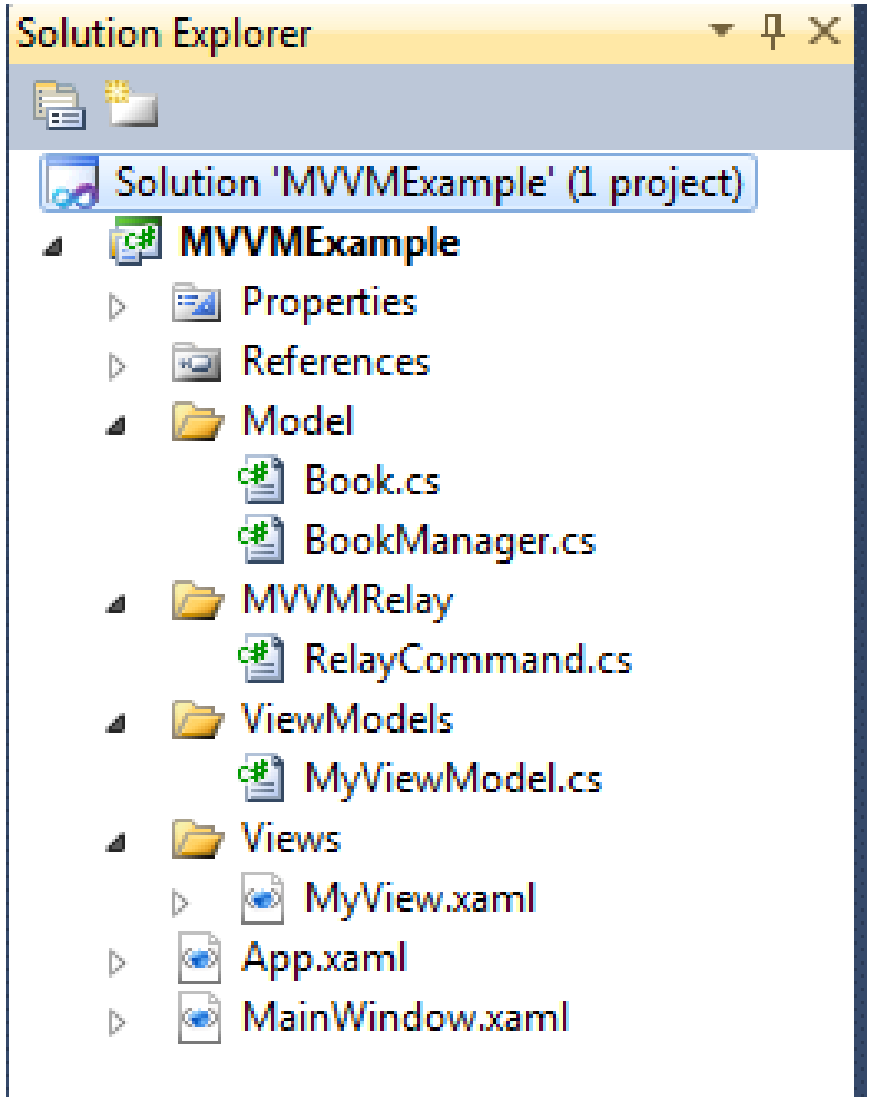
MVVM

- MVVM - Model View ViewModel.
- Design Pattern derivat din MVC: ce adaugă un *view model* pentru legarea părții de *Model* cu *View*.
- A fost creat pentru implementarea "data binding" în WPF și pentru o separare mai bună a *Model* de *View*.
- Față de MVC, MVVM elimină codul care face legătura dintre *View* și *Model*, ceea ce permite concentrarea pe design-ul UI.



Organizarea proiectului

- Proiect WPF cu numele: MVVMExample.
- 4 foldere:
  - Model: clasele ce lucrează asupra datelor.
  - Views: controalele de GUI (View-uri).
  - ViewModels: clasele *view model*.
  - MVVMRelay: clasa care rutează comenzile între obiecte.



MainWindow.xaml

Design ↑↓ XAML

```
<Window x:Class="WpfApplication2.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:Views="clr-namespace:WpfApplication2.Views"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Views:MyView/>
    </Grid>
</Window>
```

- OBS: No code behind!

View

- Configurare View-Model.
- OBS: No code behind!

Design ↑↓ XAML

```
<UserControl x:Class="WpfApplication2.Views.MyView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presenta
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibi
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:ViewModels="clr-namespace:WpfApplication2.ViewModels"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">

    <UserControl.DataContext>
        <ViewModels:MyViewModel/>
    </UserControl.DataContext>
    <Grid Name="BooksDetailView">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="20" />
```

View

- Definire legături către proprietăți din clasa ModelView.

XAML

```
<TextBox Name="TbxTitle"
      Grid.Row="0" Grid.Column="1"
      HorizontalAlignment="Left"
     Text="{Binding Path=Title}">
</TextBox>

<TextBox Name="TbxAuthors"
      Grid.Row="1" Grid.Column="1"
      HorizontalAlignment="Left"
      Text="{Binding Path=Authors}"/>

<Grid Name="GridBtns"
       Grid.Row="4" Grid.ColumnSpan="2"
       HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
    <Grid.ColumnDefinitions>
```

View

- Definire comenzi pentru butoane (metodele *Execute* și *CanExecute*).

XAML

```
<Button Name="BtnAdd" Content="AddBook"
    Grid.Column="0"
    HorizontalAlignment="Center"
    Command="{Binding AddBookCommand}"
    />
<Button Name="BtnSearch" Content="SearchBook"
    Grid.Column="1"
    HorizontalAlignment="Center"
    Command="{Binding SearchBookCommand}"
    />
<Button Name="BtnDelete" Content="DeleteBook"
    Grid.Column="2"
    HorizontalAlignment="Center"
    Command="{Binding DeleteBookCommand}"
```

View

- Definire ListView și GridView pentru vizualizarea și legătura cu ModelView pentru vizualizarea conținutului.

```
</Grid>
<ListView Name="LstBooks"
          Grid.Column="3"
          ItemsSource="{Binding Path = Books}"
          SelectedItem="{Binding Path = SelectedBook}">
    <ListView.View>
        <GridView>
            <GridViewColumn Header="Title"
                            Width="50"
                            DisplayMemberBinding="{Binding Path=Title}" />
            <GridViewColumn Header="Authors"
                            Width="100"
                            DisplayMemberBinding="{Binding Path=Authors}"

        </GridView>
    </ListView.View>
```

ViewModel

- Clasa definește:
  - Un obiect "domObject" ce reprezintă "Modelul" obiectului (cărții) cu care se lucrează (adăugare/selecție).
  - Un obiect "bookManager" ce administrează lista de cărți adăugate.
  - Un obiect de tipul *ObservableCollection* ce conține lista de obiecte monitorizate de diverse obiecte GUI (i.e., lista din View).
  - Definiția comenzilor (adăugare, ștergere, căutare).

```
class MyViewModel
{
    #region Private Variables
    private readonly Book domObject;
    private readonly BookManager bookManager;
    private readonly ObservableCollection<Book> _books;
    private readonly ICommand _addBook;
    private readonly ICommand _deleteBook;
    private readonly ICommand _searchBook;
    #endregion

    public MyViewModel()
    {
        _books = new ObservableCollection<Book>();
        _addBook = new MVVMRelay.RelayCommand(Add, CanAdd);
        domObject = new Book();
        bookManager = new BookManager();
```

ViewModel

- Clasa definește (cont.):
  - Proprietăți pentru câmpurile unei cărți și apelul metodei *OnPropertyChanged* la modificarea proprietăților.

```
#region Properties
public string Title
{
    get { return domObject.Title; }
    set
    {
        domObject.Title = value;
        OnPropertyChanged(Title);
    }
}
public string Authors
{
    get { return domObject.Authors; }
    set
    {
        domObject.Authors = value;
        OnPropertyChanged(Authors);
    }
}
```

ViewModel

- Clasa definește (cont.):
  - Proprietăți pentru accesarea listei de cărți.

```
/// <summary>
/// Gets the List of books. Used to maintain the Book List.
/// </summary>
public ObservableCollection<Book> Books { get { return _books; } }

/// <summary>
/// Sets the Selected Book. Used to identify the selected book from the list.
/// </summary>
public Book SelectedBook
{
    set
    {
        Title = value.Title;
        Authors = value.Authors;
    }
}
#endregion
```

ViewModel

- Clasa definește (cont.):
  - Comenzile *Execute* și *CanExecute*.

```
public ICommand AddBookCommand{ get { return _addBook; } }
public ICommand DeleteBookCommand{ get { return _deleteBook; } }
public ICommand SearchBookCommand { get { return _searchBook; } }
#endregion

#region AddBookCommand
public bool CanAdd(object obj)
{
    // Enable button only if we can add
    if (Title != string.Empty && Authors != string.Empty)
        return true;
    return false;
}
public void Add(object obj)
{
    var book = new Book { Title = Title, Authors = Authors };
    if (bookManager.addBook(book))
    {
        // Add it to the list of observable objects
        _books.Add(book);
        // Reset book
        Title = string.Empty;
        Authors = string.Empty;
        MessageBox.Show("Book successfully added!");
    }
    else
```

ViewModel

- Clasa definește (cont.):
  - Metodele pentru generarea de evenimente la schimbarea valorii modelului.

```
#region INotifyPropertyChanged Members

/// <summary>
/// Aici sunt evenimentele la care  controalele din view for face subscriptie.
/// Acestea se vor improspata in momentul schimbarii de stare a unei proprietati.
/// </summary>
public event PropertyChangedEventHandler PropertyChanged;

/// <summary>
/// Cand proprietatea se schimb apelam aceasta metoda pentru a porni evenimentul de PRopertyChanged
/// </summary>
/// <param name="propertyName"></param>
public void OnPropertyChanged(string propertyName)
{
    //Pornim evenimentul de PropertyChaned in caz ca cineva v-a face subscriptie la el.
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
#endregion
```

Model

- Definește două clase:
  - Book.
  - BookManager.

```
namespace WpfApplication2.Model
{
    class Book
    {

        public string Title { get; set; }
        public string Authors { get; set; }

        public Book() {
            Title = "";
            Authors = "";
        }

    }
}
```

```
namespace WpfApplication2.Model
{
    class BookManager
    {
        private readonly List<Book> _books;

        public BookManager()
        {
            _books = new List<Book>();
        }

        public bool addBook(Book book)
        {
            // TODO: Search for duplicates!
            _books.Add(book);
            return true;
        }
```