GUI-Entwicklung 2: Windows Presentation Foundation

Proseminar *Objektorientiertes Programmieren mit .NET und C#*

Sandra Müller

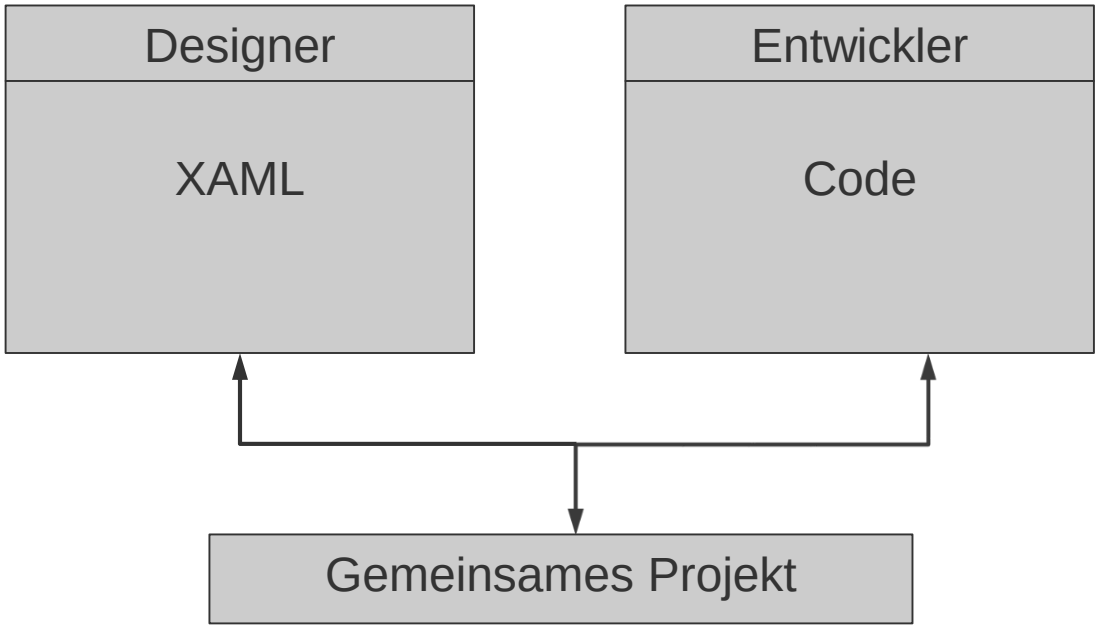
Institut für Informatik
Software & Systems Engineering

Agenda

1. Die WPF
2. Einführung in XAML
3. Wichtige Bestandteile der WPF
    1. Routed Events
    2. Data Binding
    3. Ressourcen
4. Fazit

1. Die WPF

- Einführung mit .NET 3.0
- Neues Werkzeug zum Erstellen Grafischer Benutzeroberflächen
- Neues Konzept: Trennung von Grafik und Logik



Agenda

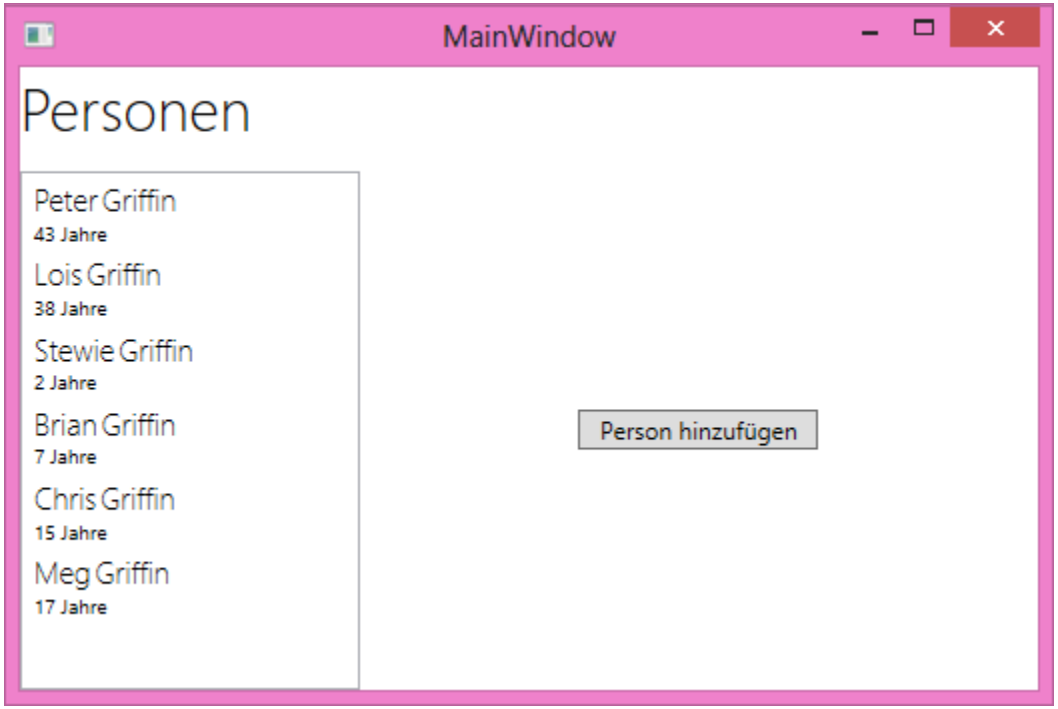
1. Die WPF
2. Einführung in XAML
3. Wichtige Bestandteile der WPF
    1. Routed Events
    2. Data Binding
    3. Ressourcen
4. Fazit

2. Einführung in XAML

- e**X**tensible **A**pplication **M**arkup **L**anguage
- Erweiterung von XML
- Verwendung zum Gestalten von User Interfaces
- Deklarative, hierarchisch aufgebaute Sprache
- Definition von Elementen mithilfe sog. Tags

2. Einführung in XAML

```
<Window>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition Width="2*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition Height="5*"/>
        </Grid.RowDefinitions>
```




```
        <TextBlock Grid.ColumnSpan="2" Text="Personen" FontFamily="Segoe UI Light" FontSize="30"/>
        <Button Grid.Column="1" Grid.Row="1" Width="120" Height="20" Content="Person hinzufügen"/>
        <ListView x:Name ="PeopleList" Grid.Column="0" Grid.Row="1"/>
    </Grid>
</Window>
```

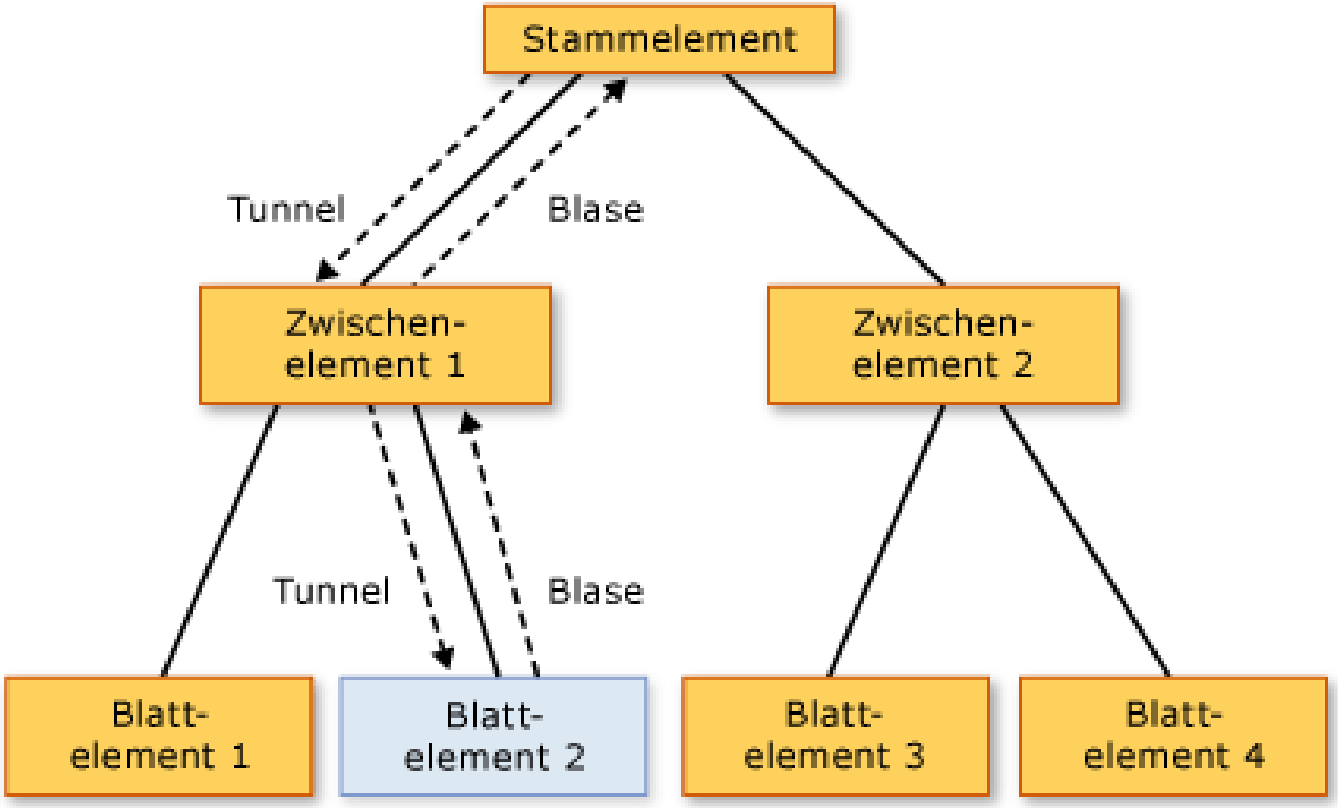
Agenda

1. Die WPF
2. Einführung in XAML
3. Wichtige Bestandteile der WPF
    1. Routed Events
    2. Data Binding
    3. Ressourcen
4. Fazit

3.1 Routed Events

- Eventklasse der WPF
- Bringt Vorteil gegenüber den CLR-Events: durchreichen der Events durch die Elementhierarchie
- Verschiedene Routingstrategien:
  - Bubbling
  - Tunneling
  - Direct
- Reihenfolge der Abarbeitung: Tunneling ⇨ Bubbling

3.1 Routed Events



3.1 Routed Events – Erstellung der Events

- Initialisierung in XAML bspw. durch Click-Event

```
<Button Grid.Column="1" Grid.Row="1" Width="120" Height="20" Content="Person hinzufügen"
      Click="Button_Click_AddPerson" />
```

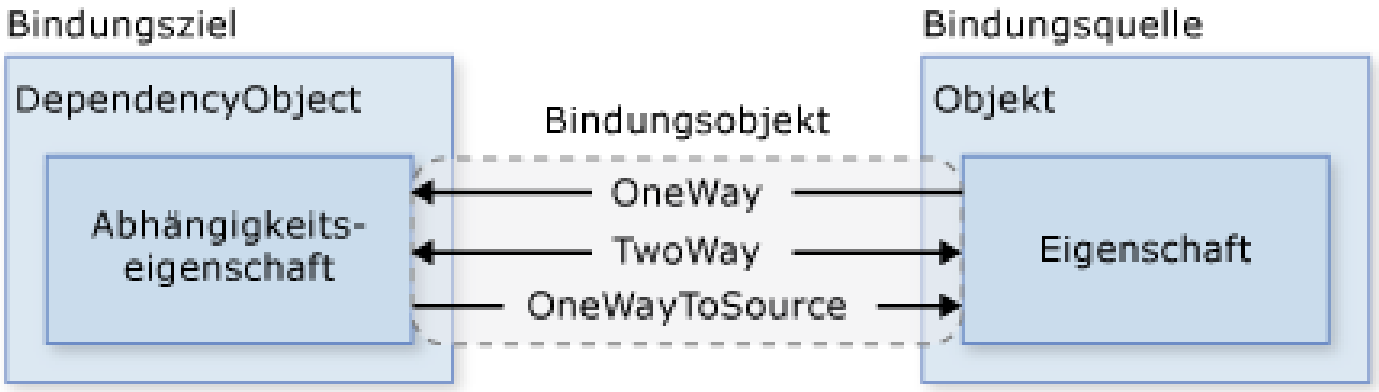
- Analog zu CLR-Events: Handler hinzufügen

```
private void Button_Click_AddPerson(object sender, RoutedEventArgs e)
        {
            //DO SOMETHING
        }
```

3.2 Data Binding

- Darstellung von Daten in der Benutzeroberfläche
- Automatische Aktualisierung der Daten
- Arten der Bindung:
  - Default
  - OneTime
  - OneWay
  - OneWayToSource
  - TwoWay

3.2 Data Binding - Bindungsmodi



3.2 Data Binding - Implementierung

- Bindung mithilfe von C#-Code:

```
private void AddBinding()
        {
            Binding binding = new Binding();
            binding.Path = new PropertyPath("NewPrename");
            binding.Mode = BindingMode.OneWayToSource;
            TextBoxPrename.SetBinding(TextBox.TextProperty, binding);
        }
```

3.2 Data Binding - Implementierung

- Bindung mithilfe von XAML-Code:

```
<TextBox>
      <TextBox.Text>
            <Binding Path="NewSurname" Mode="OneWayToSource"/>
      </TextBox.Text>
</TextBox>
```

- Bindung mithilfe der XAML Markup Extension:

```
<TextBox Text="{Binding Path=NewAge, Mode=TwoWay}"/>
```

3.2 Data Binding – Aktualisierung der Daten

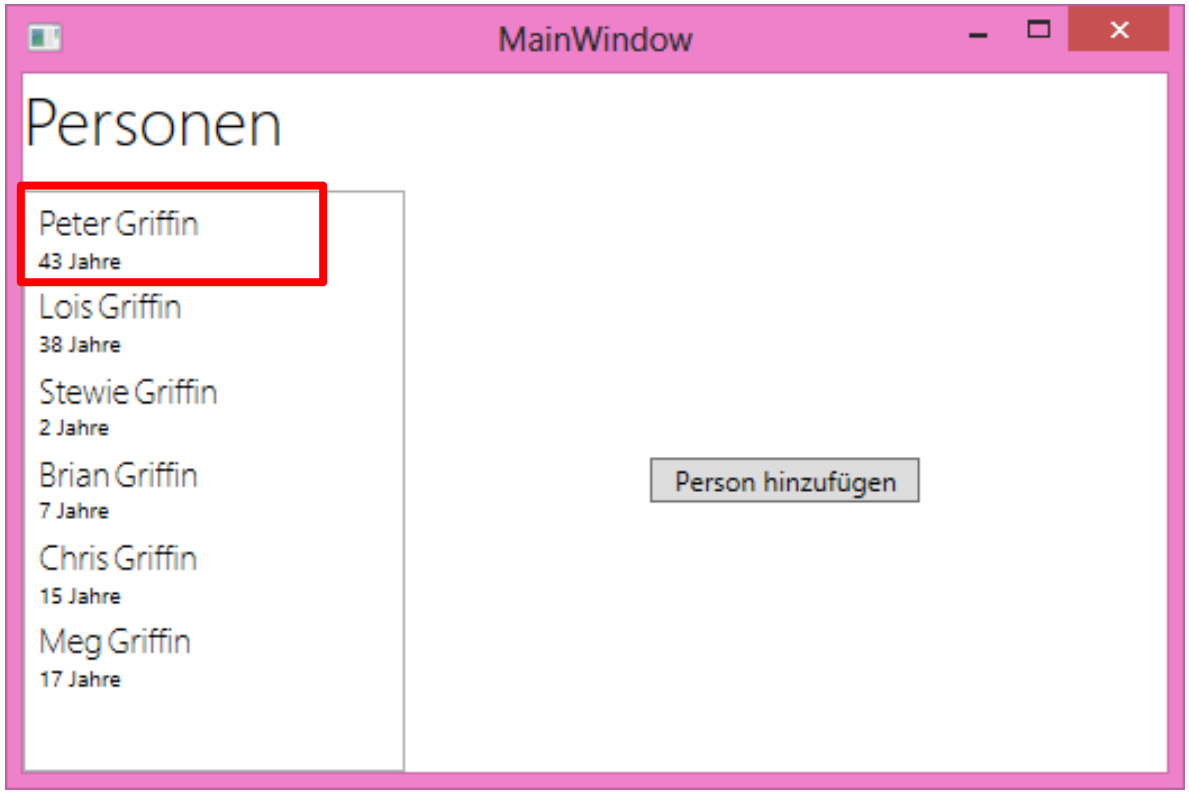
- Notwendig:
  - DependencyProperties
  - Implementierung von INotifyPropertyChanged
  - Implementierung des Events PropertyChanged

```
public event PropertyChangedEventHandler PropertyChanged;
```

- Hilfreich:
  - Implementierung einer Methode OnPropertyChanged zum Auslösen von PropertyChanged

```
public void OnPropertyChanged(string name)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(name));
}
```

3.2 Data Binding



3.2 Data Binding – DataTemplates

- Erstellen einer Vorlage für das Aussehen von Elementen/Elementgruppen
- Benötigt für die Darstellung und Bindung von Elementen in bspw. Einer ListView
- Definition in Ressourcen

```
<Window.Resources>
        <DataTemplate x:Key="ListItemDataTemplate">
                 <StackPanel Grid.Row="0" Orientation="Horizontal">
                    <TextBlock Text = "{Binding Prename}"/>
                    <TextBlock Text = "{Binding Surname}"/>
                </StackPanel>
        </DataTemplate>
</Window.Resources>
```

3.2 Data Binding – ValueConverter

- Bisher: Nur Bindung von strings möglich
- ValueConverter: Werte können automatisch in vom Code benutzbares Format konvertiert werden
- Voraussetzung: Interface IValueConverter muss implementiert werden
- Interface fordert Convert und ConvertBack Methode
- Convert: Konvertierung Code zu UI
- ConvertBack: UI –Eingaben zu Code

```
public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
        {
            return value.ToString() + " Jahre";
        }
```

3.3 Ressourcen

- Benötigt zum konsistenten gestalten von Apps
- Ablegen von Styles für verschiedene Elemente im Ressourcen-Attribut einer XAML Klasse, bspw

```
<Application.Resources>

</Application.Resources>
```

- Wichtig: jeder Ressource muss ein individueller Schlüssel über das Attribut `x:Key` zugeteilt werden

3.3 Ressourcen - Ressourcenwörterbücher

- Umfangreiche Ressourcen: Auslagern in XAML-Klasse ResourceDictionary
- Definieren der Resourcen dort
- Einbindung der Resourcen in gesamte Application oder in einzelne Elemente über Resources-Tag

Agenda

1. Die WPF
2. Einführung in XAML
3. Wichtige Bestandteile der WPF
    1. Routed Events
    2. Data Binding
    3. Ressourcen
4. Fazit

4. Fazit

- WPF bietet viele praktische Tools um die Benutzeroberfläche zu gestalten
- Löst das Skalierungsproblem von WinForms
  → relative Positionierung
- Vereinfachte Handhabung von Eventhandling durch Routed Events
  - Routingstrategien
  - Abfangen der Events an mehreren Stellen möglich
- Flexible Datenbindung
  - Keine Beschränkung auf Controls
  - Alle DependencyProperties sind bindungsfähig