MVVM

Johannes Krenig

Ziel des Vortrags

- Kleine Einführung in MVVM
- Vorstellung von WPF
- Verständnis für das Prinzip hinter MVVM

- Keine Programmierstunde
- Keine technischen Details

Gliederung

1. Geschichte des MVVM
2. Ziele des MVVM
3. Die Schichten des MVVM
4. Problemstellung
5. Lösungsversuch
6. Fazit
7. Übung

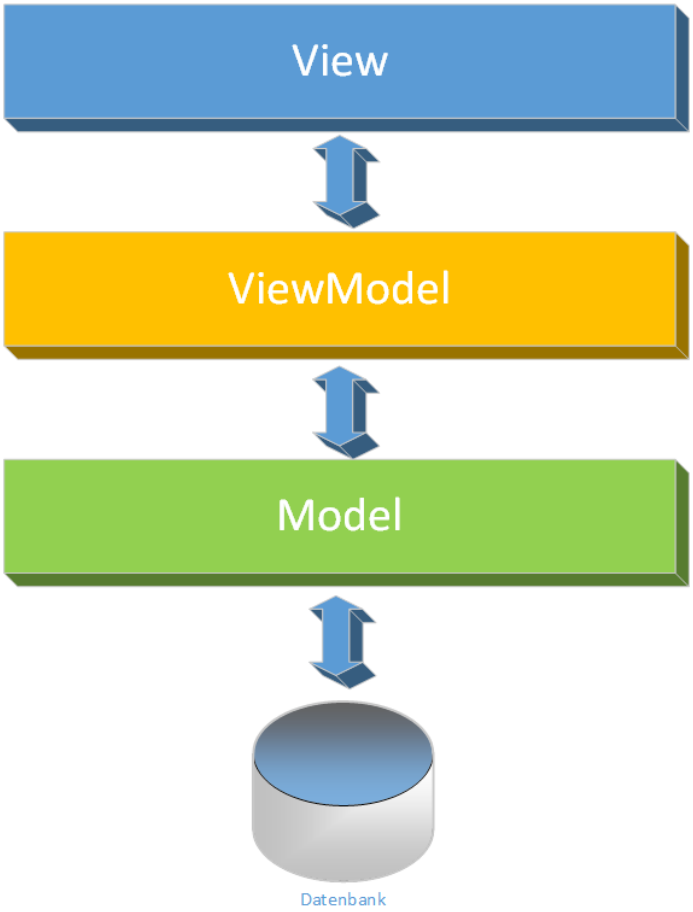
Geschichte des MVVM

- John Gossmann 2003 auf Blog beschrieben
- Entwicklung im Zuge mit WPF und Silverlight
- Weiterentwicklung von MVP-Entwurfsmuster

Ziele des MVVM

- Oberflächendesign soll unabhängig von der darunter liegenden Logik sein
- Gute Testeigenschaften der Logik

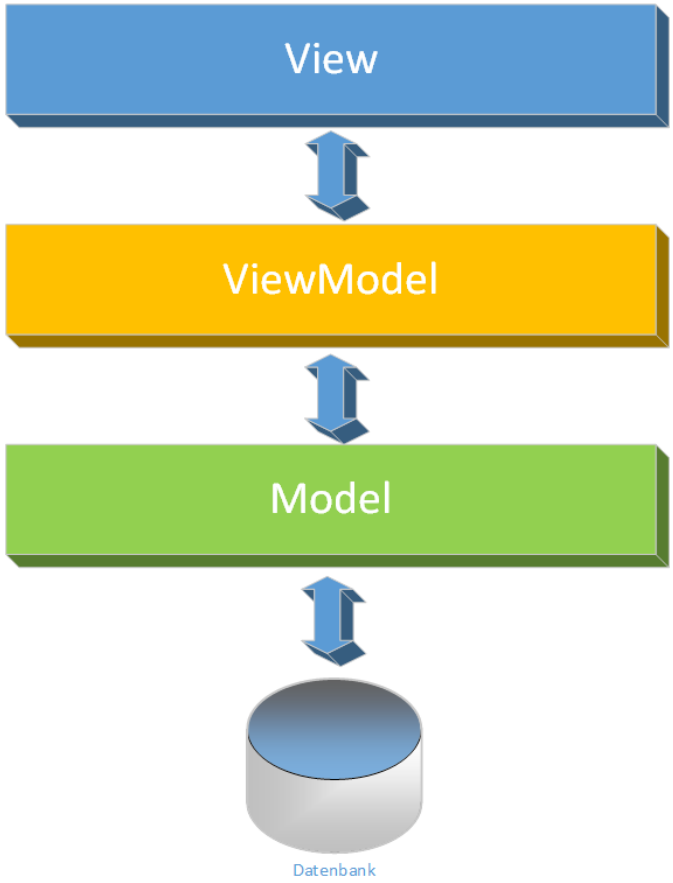
Die Schichten des MVVM



View

View ist wie im MVC und MVP für Bereitstellung des Benutzeroberfläche zuständig

Über Binding mit der ViewModel verbunden



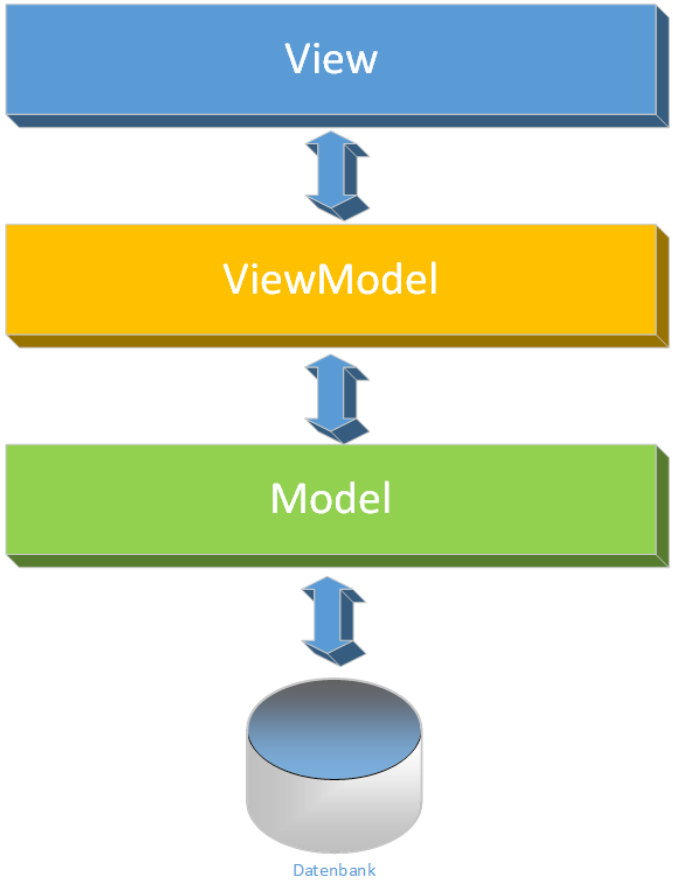
ViewModel

ViewModel beinhaltet die UI-Logik

Bindeglied zwischen View und Model

Ruft Methoden des Model auf

Stellt der View Properties zur Verfügung

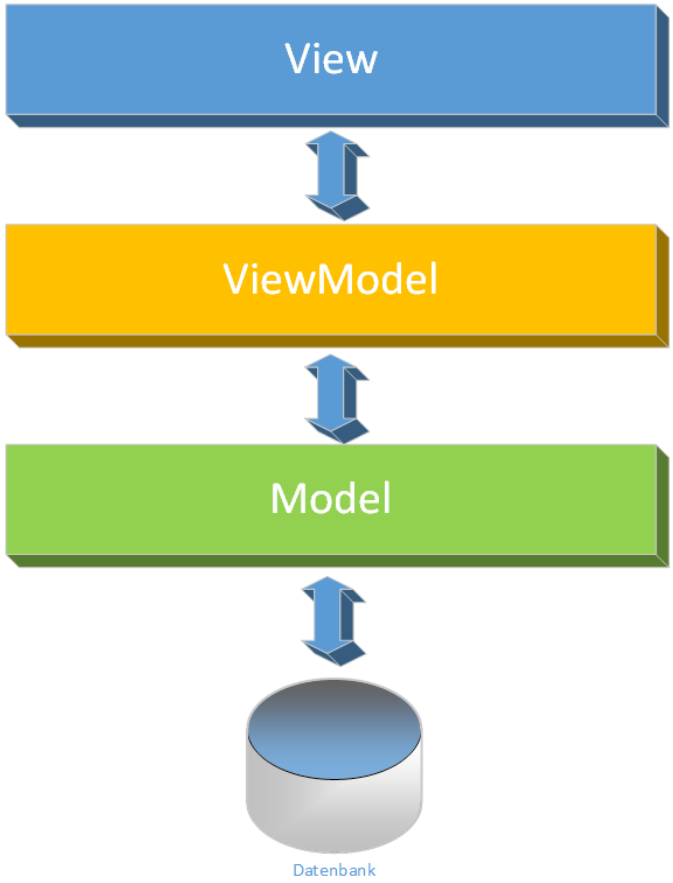


Model

Model ist wie in MVC und MVP die Datenzugriffsschicht

Validiert Benutzereingabe

Benachrichtigt ViewModel über Datenänderung



Problemstellung

Welcher Programmierer macht schon gerne Oberflächendesign?

=>Die Benutzeroberfläche soll unabhängig von der Programmierung durch Designer statt finden…

Lösungsversuch:

MVVM, umgesetzt mit WPF

Mittel:

- Binding
- Command-Konzept
- XAML

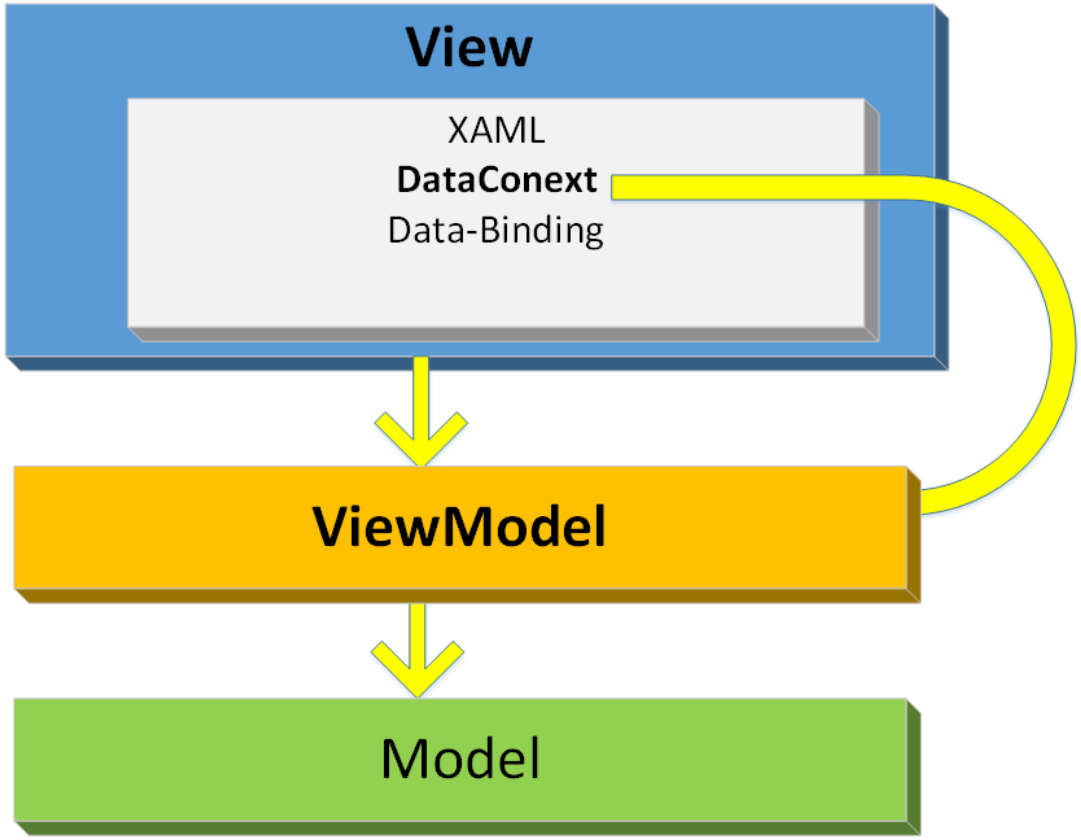
WPF

- Windows Presentation Foundation
- Framework zur GUI-Erstellung von Microsoft
- Teil des .NET-Frameworks von Microsoft
- Nutzung von DirectX
- Hochwertige grafische Ergebnisse

Binding

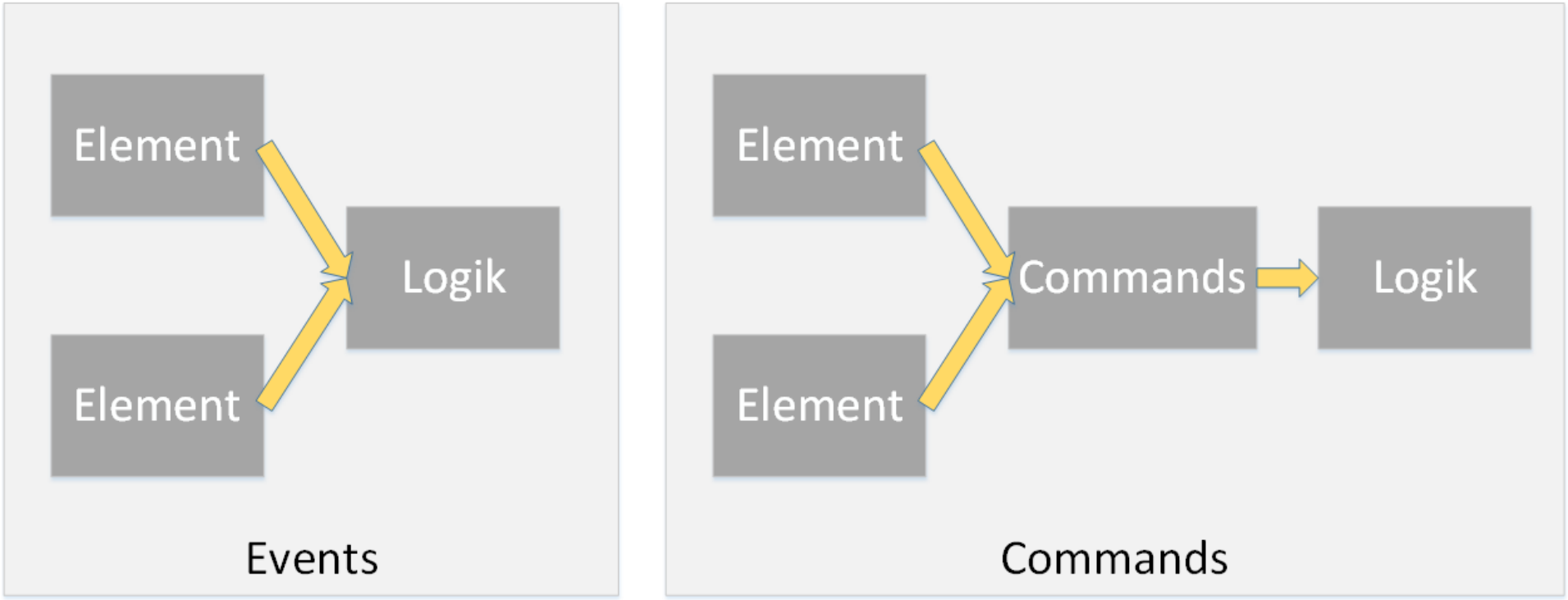
Die View bindet sich an Eigenschaften des ViewModels

Automatische gegenseitige Aktualisierung von View und ViewModel



Command-Konzept

Commands entkoppeln zusätzlich View von der UI-Logik



XAML

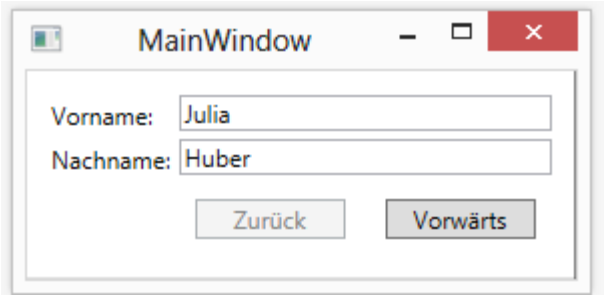
Deklarationssprache zur Erstellung von Benutzeroberflächen in WPF

XAML hat sich ebenso wie HTML dazu bewährt

XAML ist von XML abgeleitet

Oberflächendesigner muss nur XAML können statt Programmierkenntnisse

XAML

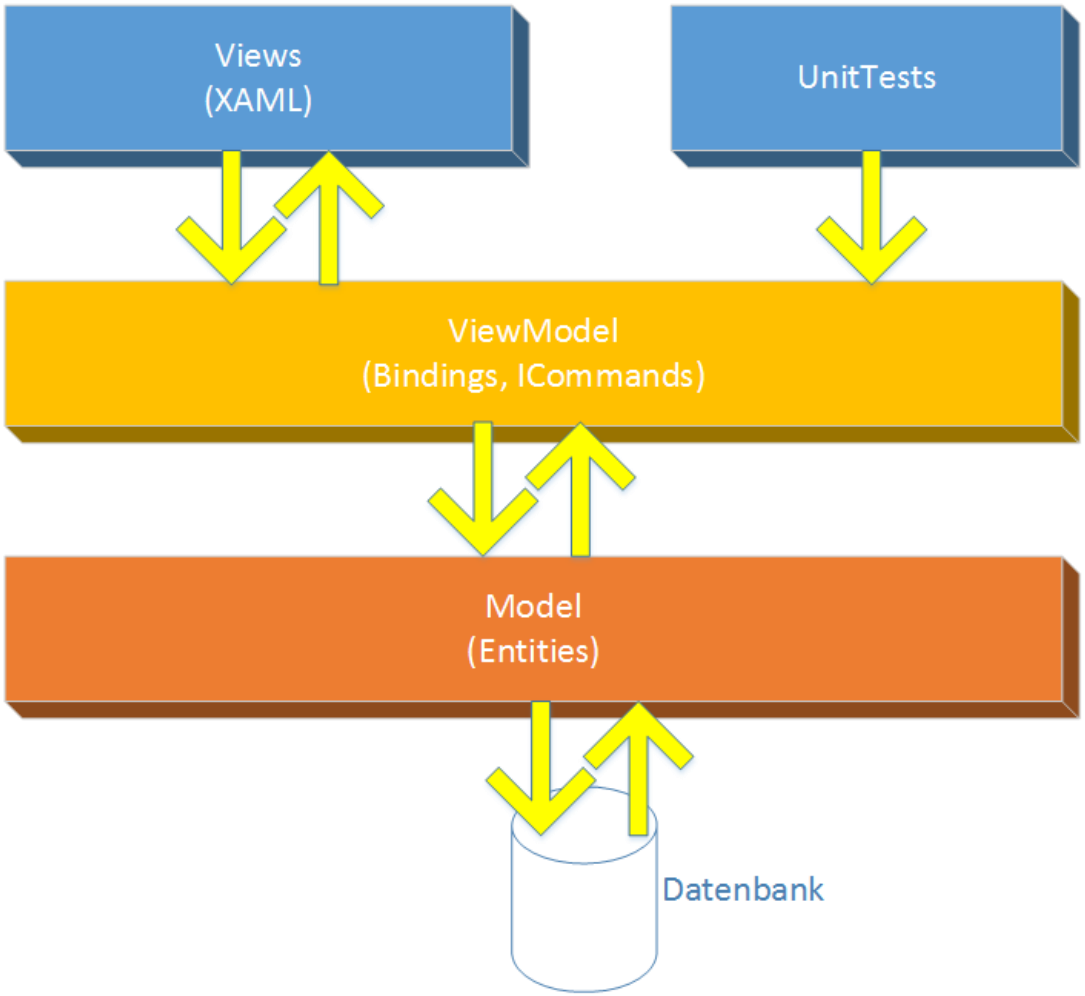



```
<Grid DataContext="{StaticResource mainViewModel}" Margin="10">
      <Grid.RowDefinitions>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="Auto"/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="Auto"/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>

      <TextBlock Text="Vorname:" VerticalAlignment="Bottom" Margin="2"/>
      <TextBox Text="{Binding Friends/FirstName}" Grid.Column="1" Margin="2"/>
      <TextBlock Text="Nachname:" Grid.Row="1" VerticalAlignment="Bottom" Margin="2"/>
      <TextBox Text="{Binding Friends/LastName}" Grid.Row="1" Grid.Column="1" Margin="2"/>

      <StackPanel Orientation="Horizontal" Grid.Row="3" Grid.Column="1">
         <Button Content="prev" Command="{Binding PreviousCommand}" Width="75" Margin="10"/>
         <Button Content="next" Command="{Binding NextCommand}" Width="75" Margin="10"/>
      </StackPanel>
   </Grid>
```

MVVM und WPF


Views
(XAML)
UnitTests
ViewModel
(Bindings, ICommands)
Model
(Entities)
Datenbank

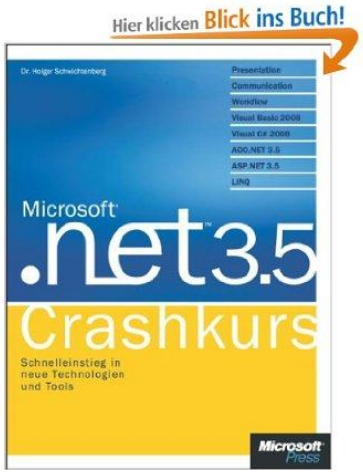
Fazit

+ Trennung von Oberflächendesign und Logik
+ Gute Testbarkeit der UI-Logik
+ Modularität
+ XAML
+ WPF

–Overhead
– bei kleinen Anwendungen
– kein Datenkontext möglich

Literatur

Thomas Huber. Windows Presentation Foundation 4.5 - Das umfassende Handbuch
Galileo Computing



Holger Schwichtenberg
Microsoft. NET 3.5 Crashkurs
Microsoft Press

Noch Fragen?