



Thomas Claudius Huber | Trivadis AG

# Das Model-View-ViewModel- Pattern

Client-Architektur für WPF und Silverlight

# Über Thomas C.H.

- .NET Senior Consultant
  - Trivadis AG Basel
  - Fokus: .NET, WPF, Silverlight, WinForms, XML, PL/SQL

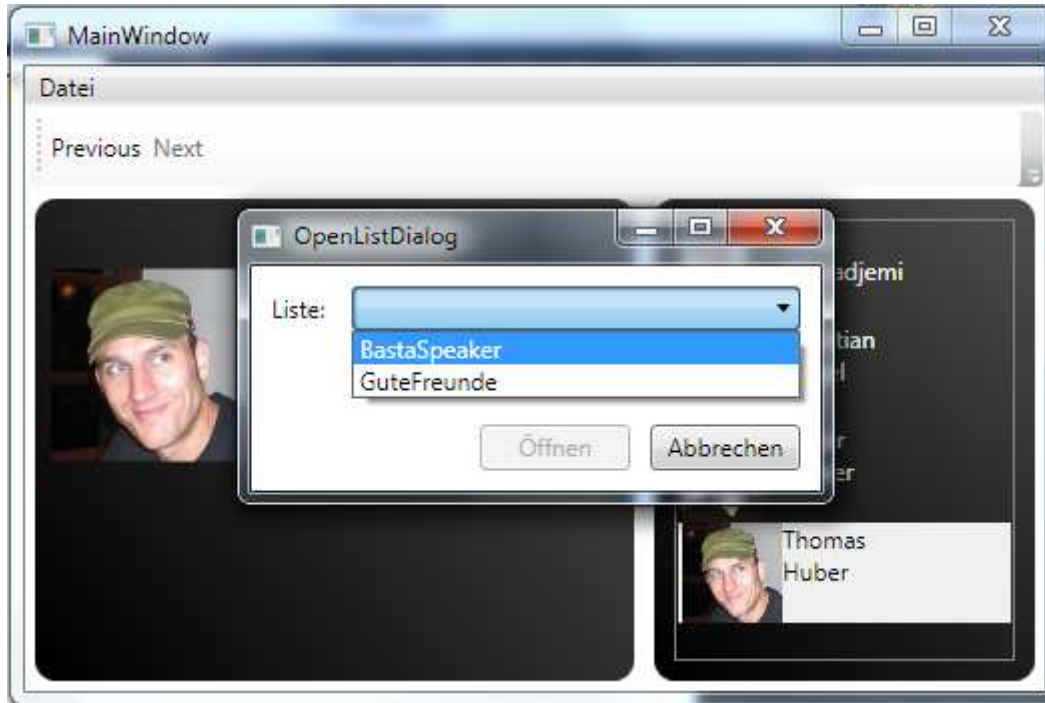


- Autor des “umfassenden Handbuchs” zur Windows Presentation Foundation



- [www.thomasclaudiushuber.com](http://www.thomasclaudiushuber.com)

# Die Beispielanwendung



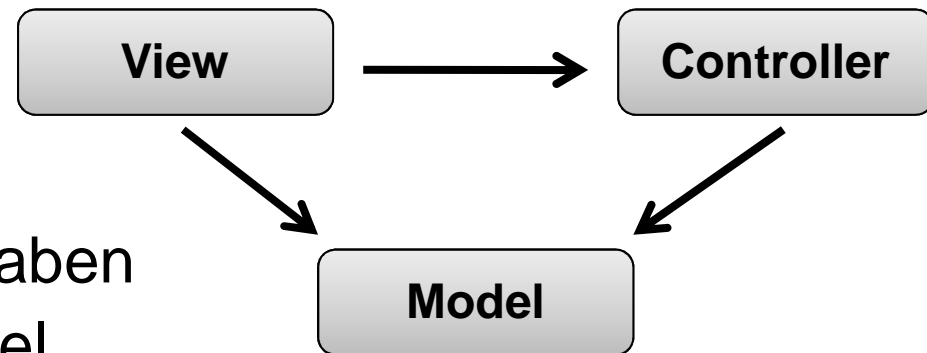
**Demo**

# Agenda

- MVC und MVVM
- Technische Grundlagen
  - Data-Binding
  - Commands
- MVVM.BeispielApp mit fortgeschrittene Szenarien
  - ViewModel-Übersicht
  - Feedback
  - DataProvider
  - UnitTests

# Das Model-View-Controller-Pattern (MVC)

- Model
  - Das Datenmodell
- View
  - Die Benutzeroberfläche
- Controller
  - Behandelt Benutzereingaben und modifiziert das Model



- MVC wurde bereits 1979 erstmals beschrieben
  - Im Zusammenhang mit Smalltalk

# Das Model-View-Controller-Pattern (MVC)

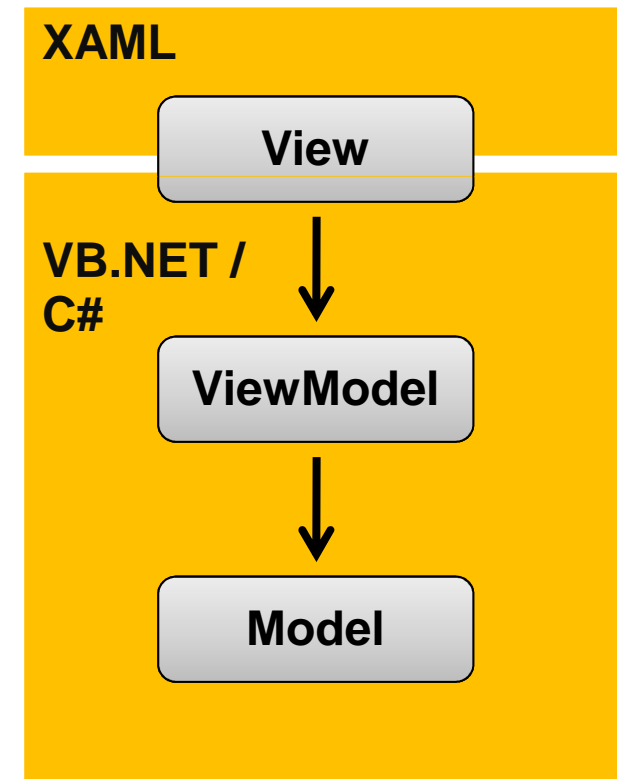
- Pattern-Experten sind sich über die exakte Rolle des Controllers nicht ganz einig
- Moderne UIs erlauben keine strikte Trennung zwischen *View* und *Controller*
  - Die meisten Controls übernehmen Anzeige und Modifikation von Daten
  - Typischer Diskussionsstoff sind beispielsweise bei Web-Anwendungen Java-Script-Validierungen, die ja in der View liegen.
- Heute gibt es modernere Varianten von MVC:
  - Model-View-Presenter (MVP)
  - Model-View-ViewModel (MVVM)

# Das MVVM-Pattern

- Wurde erstmals 2005 von John Gossman beschrieben
  - Gossman ist einer der Architekten von Expression Blend
  - <http://blogs.msdn.com/johngossman>
- Hat sich mittlerweile zum „State-of-the-Art“-Pattern für WPF- und Silverlight-Anwendungen etabliert
- Nutzt zentrale Features von WPF und Silverlight
  - Data-Binding
  - Commands (In Silverlight erst in der vierten Version enthalten)

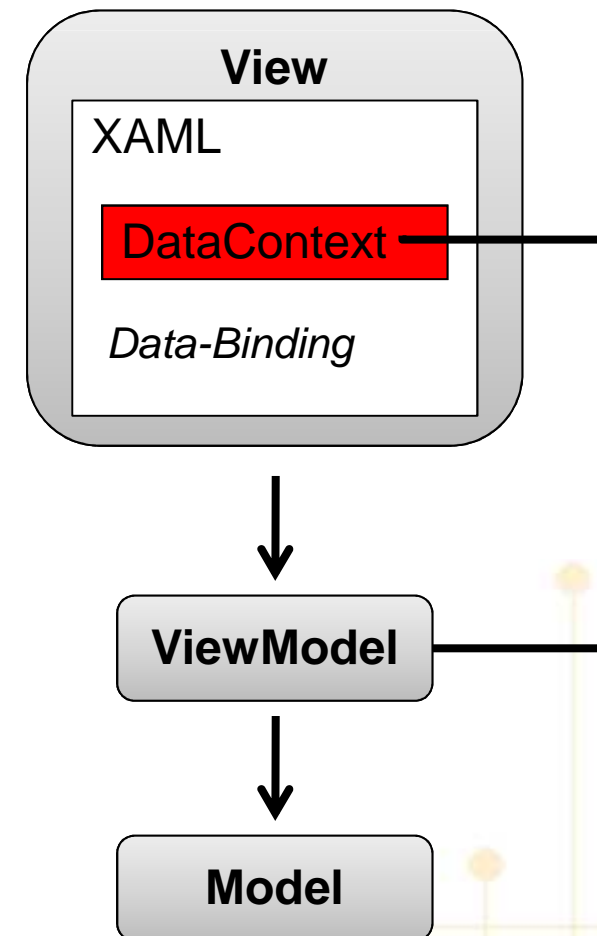
# Das MVVM-Pattern

- View (wie bei MVC)
  - In WPF/Silverlight in XAML mit Codebehind definiert
- ViewModel
  - Datenmodell spezifisch für das UI
  - Kapselt das Model
  - Enthält UI-spezifische Daten
    - Ist UI im Edit oder ReadOnly-Modus, ist ein Button deaktiviert, etc.
- Model (wie bei MVC)
  - DataSets, XML, Custom Objects, ...
  - Oft muss der UI-Entwickler nehmen, was von der Datenschicht kommt



# Das MVVM-Pattern

- ViewModel stellt über Properties Daten und Commands bereit
- Die View hat das ViewModel im DataContext, womit sich jedes Element in der View an Properties des ViewModels binden lässt



# Das MVVM-Pattern – die Stärken

- Arbeitsaufteilung zwischen Entwickler und Designer wird durch das Pattern stark verbessert
- Ein großer Teil der UI-Logik kann in Unit-Tests eingebunden werden
- UI und UI-Logik sind strikter getrennt, wodurch die Anwendung modularer aufgebaut ist und wartbarer wird.

# Agenda

- MVC und MVVM
- Technische Grundlagen
  - Data-Binding
  - Commands
- MVVM.BeispielApp mit fortgeschrittene Szenarien
  - ViewModel-Übersicht
  - Feedback
  - DataProvider
  - UnitTests

# Data-Binding

- In XAML via Binding-Markup-Extension

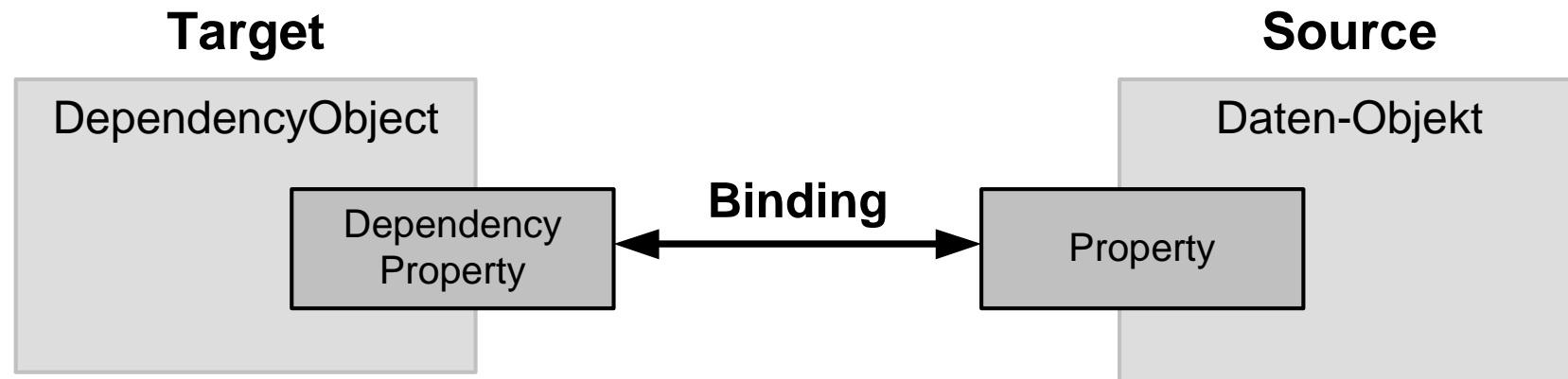
```
<TextBox Text="{Binding ElementName=sliSource,Path=Value}"/>  
<Slider x:Name="sliSource" Minimum="0" Maximum="100" Value="0"/>
```

- In VB.Net via SetBinding-Methode

```
Dim b As New Binding()  
b.ElementName = sliSource.Name  
b.Path = New PropertyPath("Value")  
txtTarget.SetBinding(TextBox.TextProperty, b)
```

# Data-Binding – Typ von Ziel und Quelle

- Das Ziel (Target) eines Data-Bindings muss eine Dependency-Property sein
- Die Quelle (Source) kann eine normale .NET Property, ein Objekt oder eine Dependency-Property sein



# Data-Binding - Benachrichtigungen

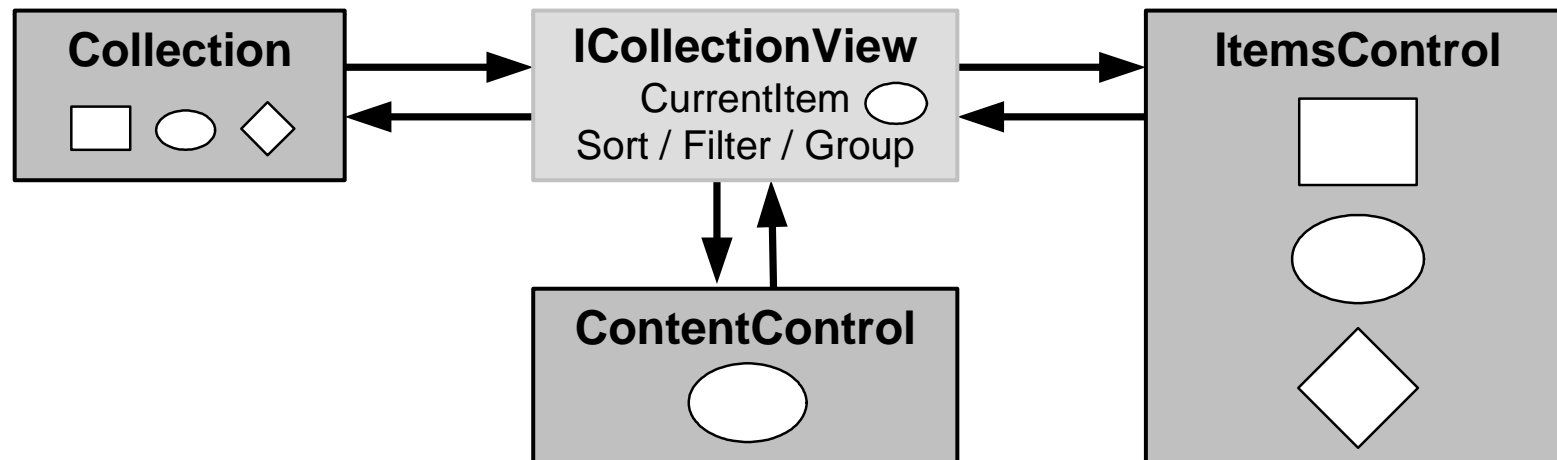
- Ist die Quelle eine Dependency-Property, ist alles ok.
- Ist die Quelle eine normale .NET Property
  - muss die Klasse **INotifyPropertyChanged** implementieren
- Ist die Quelle eine Collection
  - sollte die Collection **INotifyCollectionChanged** implementieren
  - Mit *ObservableCollection(Of T)* gibt es bereits eine Collection, welche dieses Interface implementiert.

# Data-Binding – der Ursprung der Quelle

- Zum expliziten Setzen der Quelle eines Data-Bindings bietet die Binding-Instanz drei Möglichkeiten an:
  - Source-Property
  - ElementName-Property
  - RelativeSource-Property
- Ist keine der drei Properties gesetzt, hält die Binding-Instanz nach dem DataContext Ausschau:
  - FrameworkElement definiert eine DataContext-Property. Der Wert dieser Property wird über den Element Tree vererbt.
  - Binding-Instanz ohne explizite Quelle nutzen den Wert des DataContexts als Quelle.

# Data-Binding und Collections

- WPF nutzt für das Currency-Management CollectionViews



- Wird nicht explizit eine CollectionView instanziiert, erstellt die WPF eine Default-CollectionView
  - Diese gibt's über die statische Methode **CollectionViewSource.GetDefaultView(...)**

# Data-Binding und MVVM

- ViewModel wird in den DataContext der View gesetzt
  - Elemente in der View lassen sich an die Properties des ViewModels binden
- ViewModel implementiert INotifyPropertyChanged
  - Die Properties lösen bei Änderungen das für das Data-Binding wichtige PropertyChanged-Event aus.
  - Oft wird eine Basisklasse implementiert, die das Interface implementiert und eine Hilfs-Methode zum Auslösen des PropertyChanged-Events enthält.

**Demo**

# Commands

- Ein Command ist eine Art losgelöstes Event
  - Es trennt die Semantik von der eigentlichen Logik
- In WPF/Silverlight ist ein Command ein Objekt, das die Schnittstelle *ICommand* implementiert

```
Public Interface ICommand
    Function CanExecute(ByVal parameter As Object) As Boolean
    Sub Execute(ByVal parameter As Object)
    Event CanExecuteChanged As EventHandler
End Interface
```

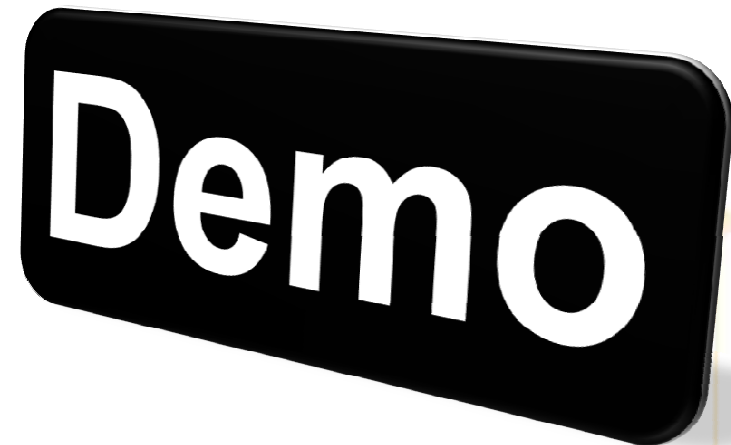
- Ein Command wird ausgeführt von Objekten, die eine *Command*-Property besitzen
  - In Silverlight 4: Button, Hyperlink
  - In WPF: Button, MenuItem

# Commands

- WPF besitzt komplexere Command-Infrastruktur
  - Die genutzten RoutedCommands gibt's nur hier, nicht jedoch in Silverlight
- Silverlight besitzt lediglich einfache Command-Unterstützung
- Es gibt weitere Command-Implementierungen
  - PRISM (Composite WPF & Silverlight) enthält diverse Commands
  - Eigene Commands lassen sich einfach implementieren.

# Commands und MVVM

- Damit die Logik aus der Command-Klasse in das ViewModel kommt, wird oft ein Command mit Delegates verwendet
  - Action(Of Object) für Execute
  - Func(Of Object, Boolean) für CanExecute



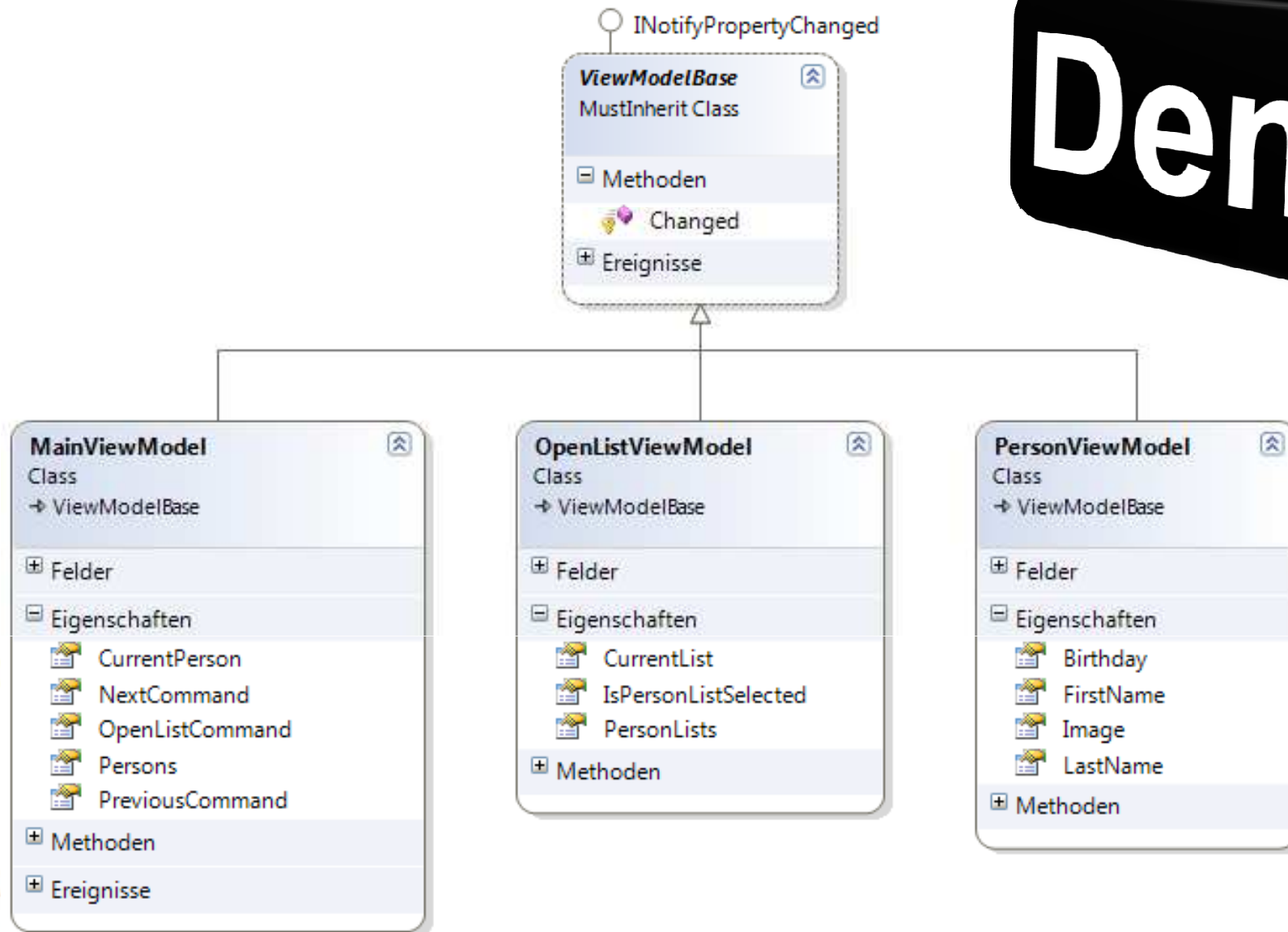
**Demo**

# Agenda

- MVC und MVVM
- Technische Grundlagen
  - Data-Binding
  - Commands
- MVVM.BeispielApp mit fortgeschrittene Szenarien
  - ViewModel-Übersicht
  - Feedback
  - DataProvider
  - UnitTests

# BeispielApp: ViewModel-Übersicht

Demo



# BeispielApp: Feedback

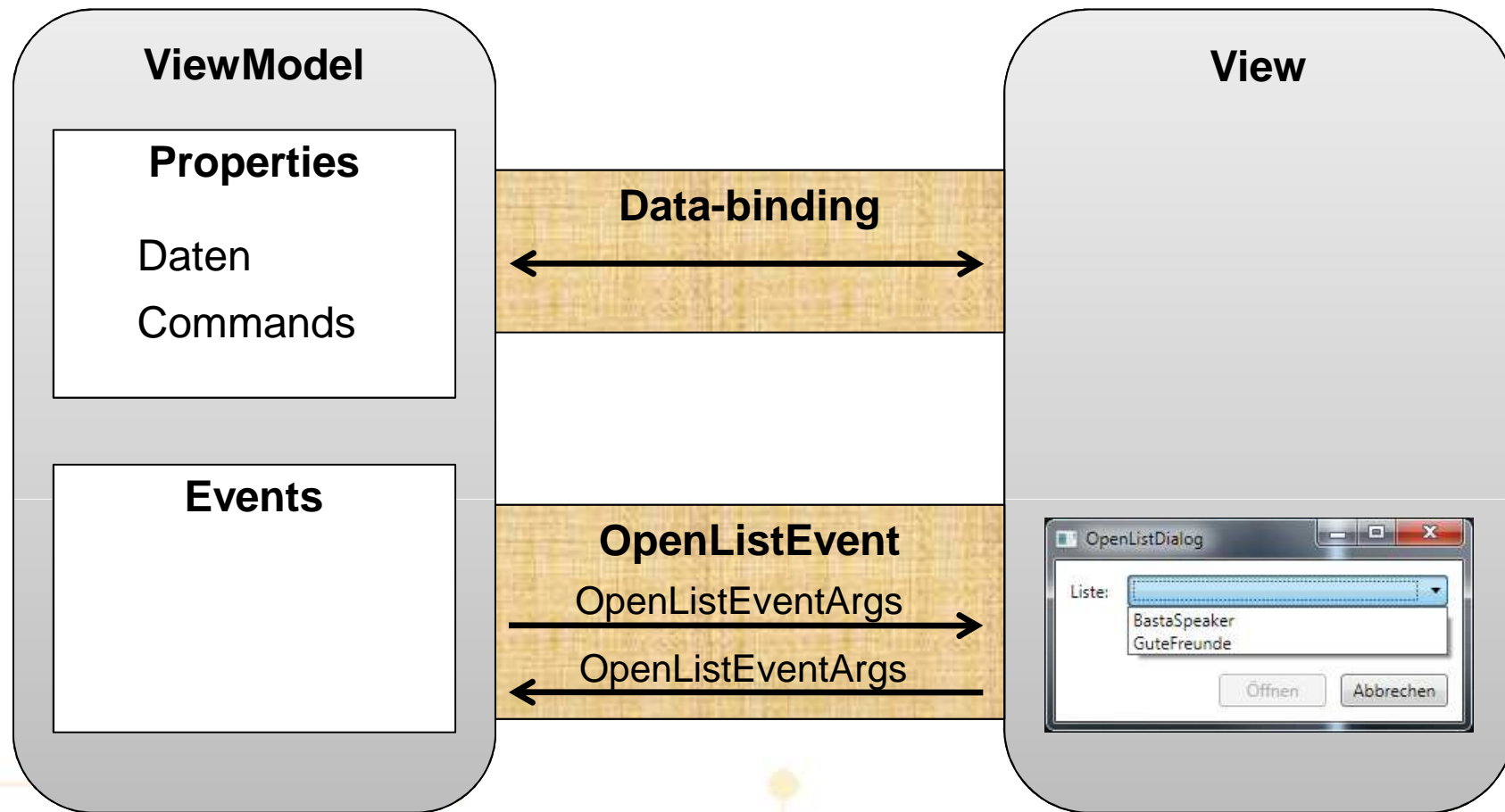
- An einer Stelle im ViewModel wird Feedback benötigt. Dem Benutzer muss ein Dialog angezeigt werden.
- Problem: ViewModel darf kein UI enthalten
  - Nur so bleibt das ViewModel für Unit-Tests geeignet
- Lösung???

# BeispielApp: Feedback

- Lösung: Das ViewModel stellt ein Event bereit.
  - Die View (UI) definiert einen Eventhandler für das Event (in der Codebehind-Datei)
  - In einem Unit-Test lässt sich so auch ein Eventhandler für das Event definieren, der die Daten direkt übergibt.
- Die Daten werden über spezielle EventArgs übergeben
  - View zeigt im Eventhandler einen Dialog an. Nach Bestätigen werden die Daten an die EventArgs übergeben. Die EventArgs werden im ViewModel weiterverarbeitet.
- ViewModel kann eine Exception werfen, wenn das Event nicht abonniert wird

# BeispielApp: Feedback

- Das ViewModel stellt Properties für das Data-Binding und Events für das Feedback bereit



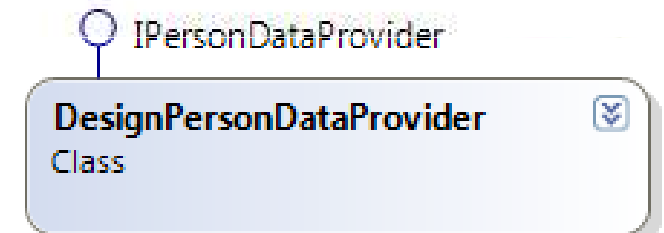
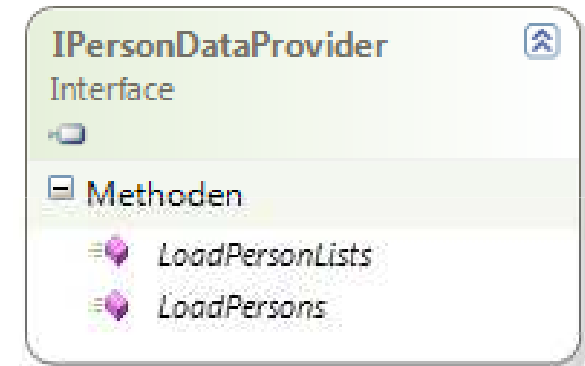
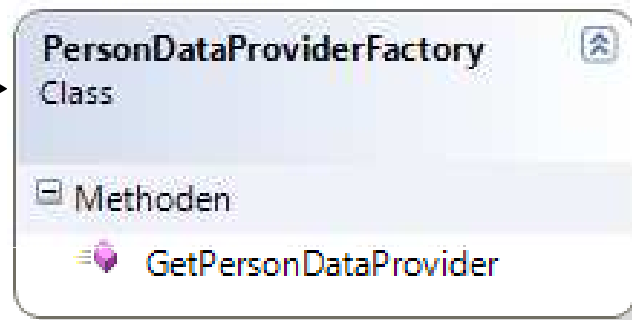
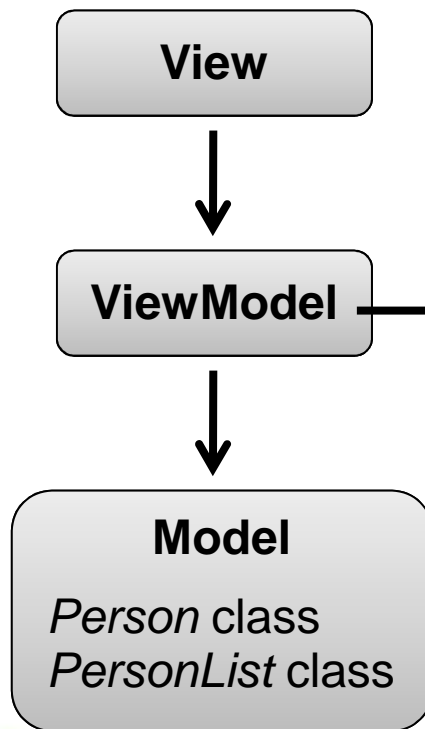
# BeispielApp:Feedback

**Demo**

# BeispielApp: DataProvider

- Für den Designer ist es hilfreich, wenn zur Designzeit Beispieldaten zur Verfügung stehen
  - Das Aussehen von Listen ist somit in Expression Blend direkt zur Designzeit sichtbar
- Diese Funktionalität wird über eine IDataProvider-Schnittstelle erreicht:
  - Zur Designzeit wird ein Design-IDataProvider geladen
  - Zur Laufzeit ein IDataProvider der die Daten von der Datenbank holt

# BeispielApp: DataProvider



# BeispielApp: DataProvider

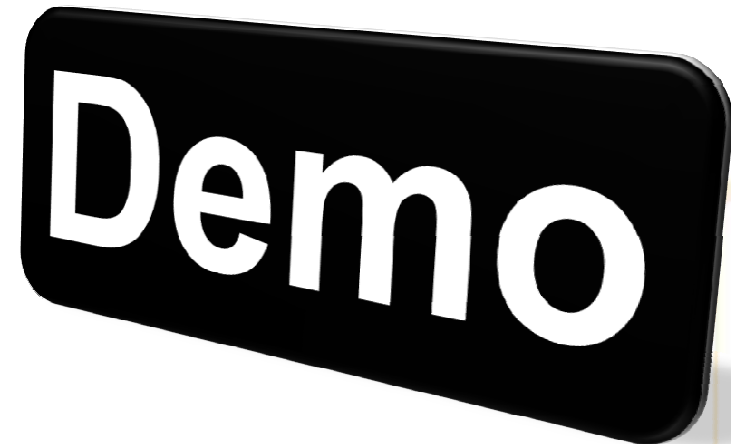
- Design in Expression Blend dank Designzeit-DataProvider besser möglich

**Demo**

# Unit Tests gegen ViewModel

```
[TestMethod]
// Test navigate Forward
public void TestIsListSelected()
{
    OpenListViewModel vm = new OpenListViewModel(GetPersonLists());
    vm.CurrentList = vm.PersonLists[0];
    Assert.IsTrue(vm.IsPersonListSelected);
}

private List<PersonList> GetPersonLists()
{
    return new List<PersonList>{
        new PersonList(),
        new PersonList()
    };
}
```



# Agenda

- MVC und MVVM
- Technische Grundlagen
  - Data-Binding
  - Commands
- MVVM.BeispielApp mit fortgeschrittene Szenarien
  - ViewModel-Übersicht
  - Feedback
  - DataProvider
  - UnitTests

# Fazit MVVM

- Bessere Unterstützung für den Designer-/Entwickler-Workflow
- Die Logik liegt im vom UI unabhängigen ViewModel, was gefundenes Fressen für Unit-Tests ist
- Die Anwendung bekommt eine bessere Struktur und bleibt wartbar

# BeispielApp + Slides & Co.

- Beispielanwendung + Slides werden heute Abend in meinem Blog gepostet:  
***[www.thomasclaudiushuber.com/blog](http://www.thomasclaudiushuber.com/blog)***
- Fragen direkt an  
***[thomas@thomasclaudiushuber.com](mailto:thomas@thomasclaudiushuber.com)***



Besten Dank für Eure Aufmerksamkeit!

**Any Questions?!**