Jörg Neumann

End-to-End-Workshop

Eine komplette .NET-Desktopanwendung mit WPF

Teil 2: Die Clientseite

Jörg Neumann

Microsoft Most Valuable Professional

- Principal Consultant bei Acando
- Associate bei Thinktecture
- MVP im Bereich „Client App Dev“
- Beratung, Training, Coaching
- Buchautor, Speaker
- Mail: Joerg.Neumann@Acando.de
- Blog: www.HeadWriteLine.BlogSpot.com

Agenda

- Analyse der Anforderungen
- Entwurf einer Architektur
- Techniken & Patterns
- Vorstellung der Lösung

Analyse

- Analyse der fachlichen Anforderungen
  - Um was geht es fachlich?
  - Wie sehen die User Workflows aus?
- Definition der technischen Anforderungen
  - Welchen Lebenszyklus hat die Anwendung?
  - Auf welchen Plattformen soll sie ausgeführt werden?

Story Boarding

- Story Boarding hilft in der Entwurfsphase
  - Visualisierung der fachlichen Möglichkeiten
  - Simulation der User Workflows
  - Planung der benötigten Assets

Conference Tool

Sprecher registrieren

Session einreichen

Session bewerten

Zeitplaner

Sprecher registrieren

Neu | Entfernen

Name: Max Mustermann

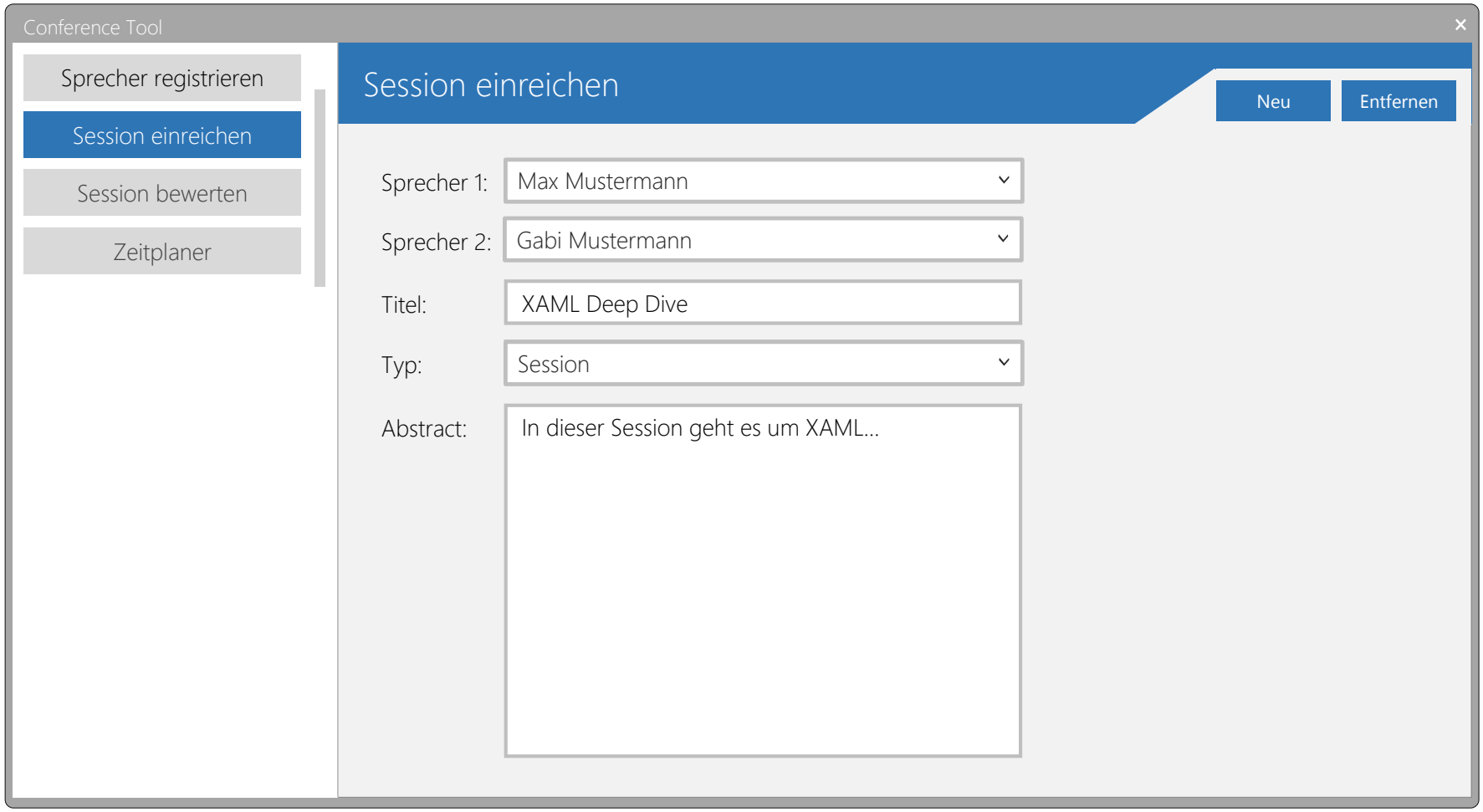
Firma: Consulting GmbH

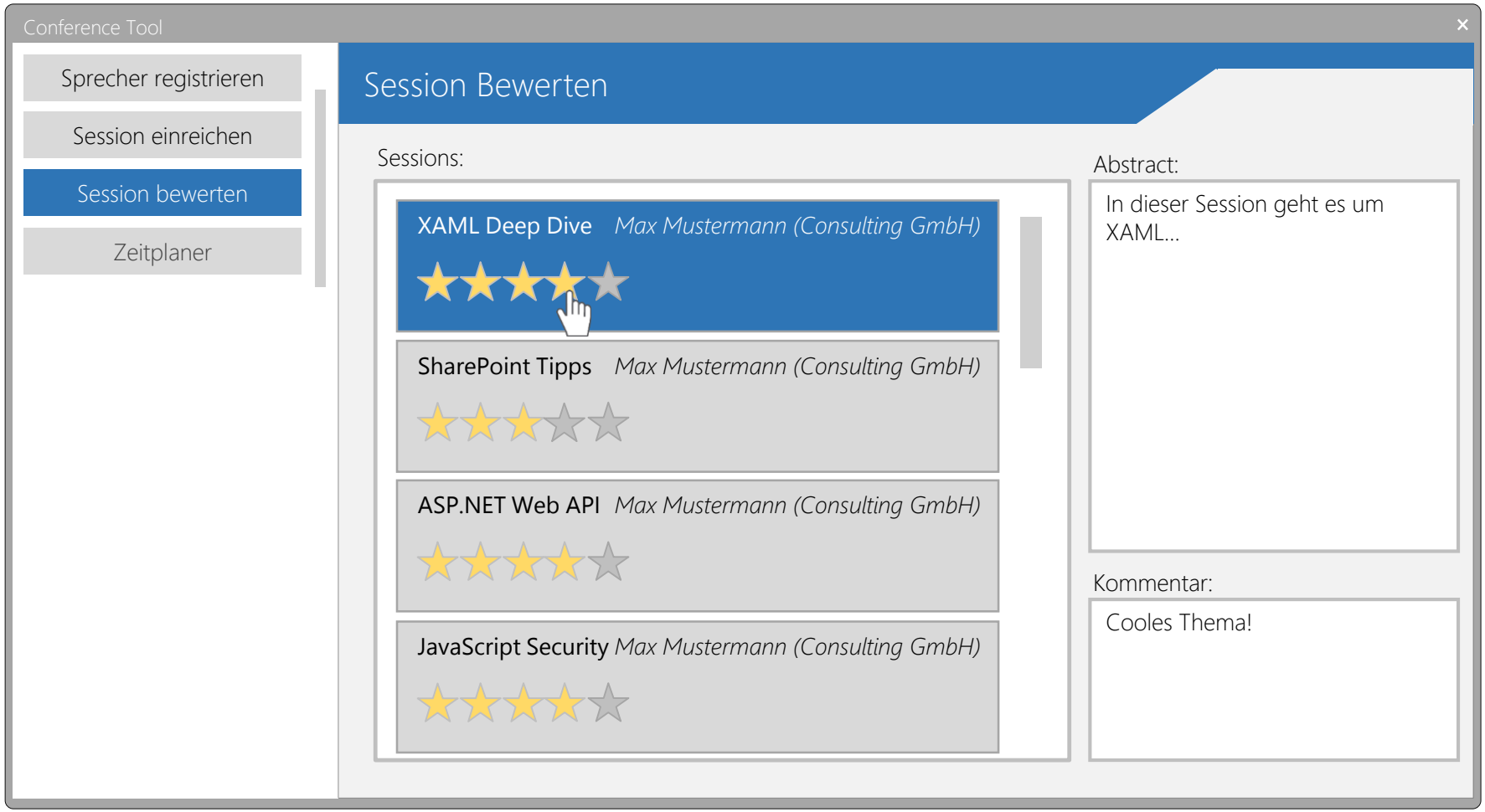
Mail: Max.Mustermann@Consulting.de

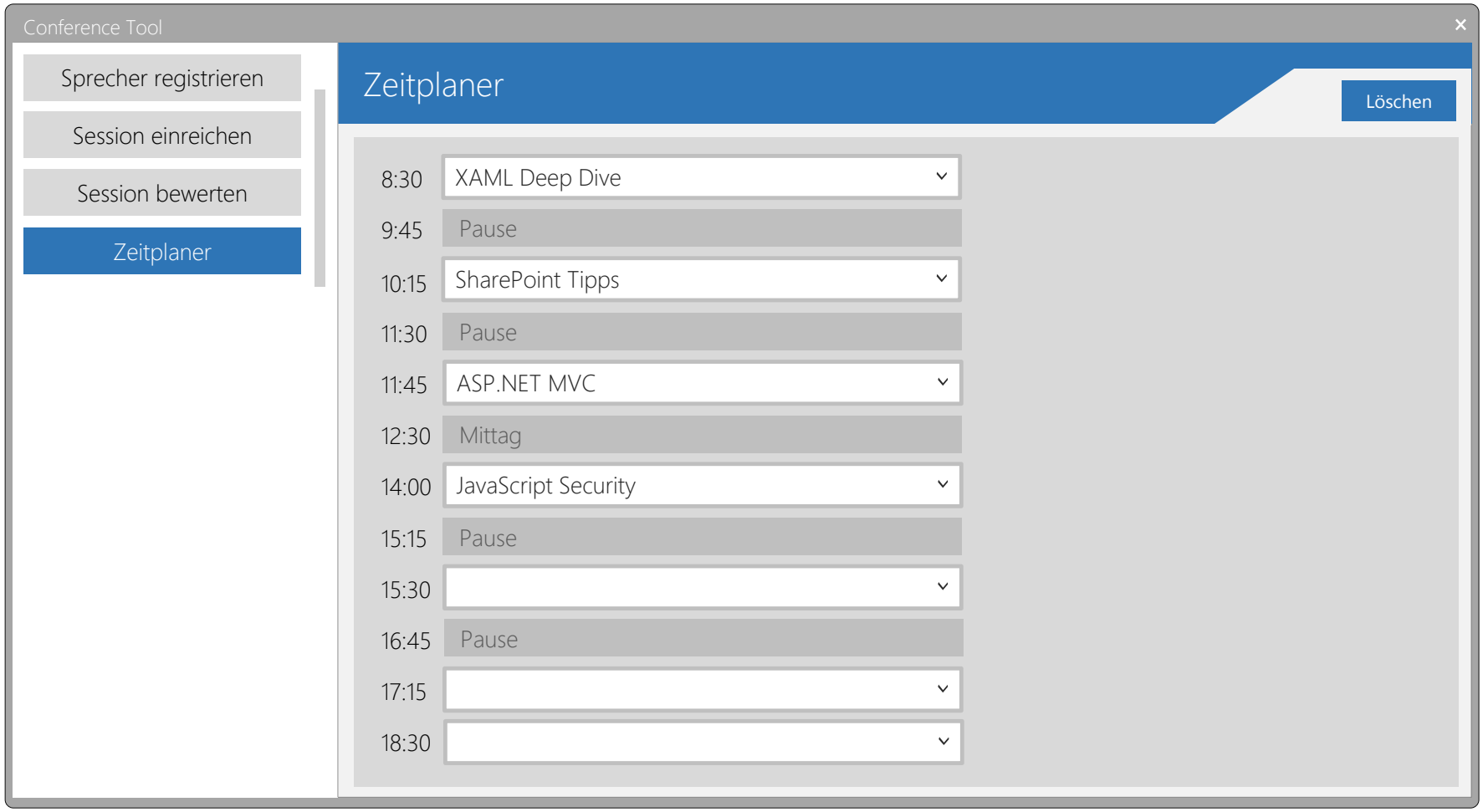
Telefon: 040/555-123

Bio: Max Mustermann ist Gründer der Firma Consulting GmbH. Er beschäftigt sich seit Jahren mit der Konzeption komplexer System, ...

Ändern | Entfernen


Conference Tool
Sprecher registrieren
Session einreichen
Session bewerten
Zeitplaner
Session einreichen
Neu
Entfernen
Sprecher 1:
Max Mustermann
Sprecher 2:
Gabi Mustermann
Titel:
XAML Deep Dive
Typ:
Session
Abstract:
In dieser Session geht es um XAML...


Conference Tool
Sprecher registrieren
Session einreichen
Session bewerten
Zeitplaner
Session Bewerten
Sessions:
XAML Deep Dive Max Mustermann (Consulting GmbH)
SharePoint Tipps Max Mustermann (Consulting GmbH)
ASP.NET Web API Max Mustermann (Consulting GmbH)
JavaScript Security Max Mustermann (Consulting GmbH)
Abstract:
In dieser Session geht es um XAML...
Kommentar:
Cooles Thema!


Conference Tool
Sprecher registrieren
Session einreichen
Session bewerten
Zeitplaner
Zeitplaner
Löschen
8:30
XAML Deep Dive
9:45
Pause
10:15
SharePoint Tipps
11:30
Pause
11:45
ASP.NET MVC
12:30
Mittag
14:00
JavaScript Security
15:15
Pause
15:30
16:45
Pause
17:15
18:30

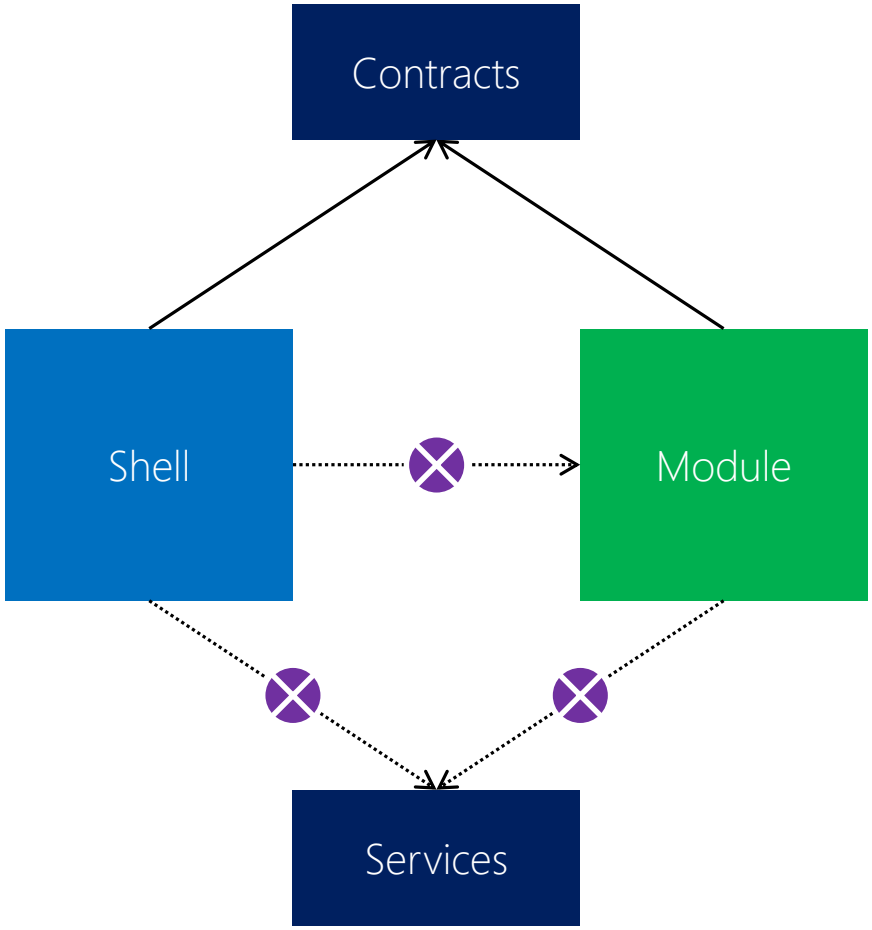
Technische Herausforderungen

- Anwendungen haben meist einen langen Lebenszyklus
  - Dies muss in der Architektur explizit berücksichtigt werden!
- Continuous Integration
  - Ständige Wartung und Weiterentwicklung gewährleisten
  - Parallele Implementierung durch mehrere Entwickler
- Testbarkeit
  - Die Anwendung ist zu jeder Zeit in einem konsistenten Zustand
  - Erfordert eine gute Planung im Architekturprozess

Atomare Einheiten

- Alle Teile der Anwendung verhalten sich autonom
  - Keine direkten Abhängigkeiten zwischen Shell und Modulen
- Nutzung von gemeinsamen Ressourcen
  - Bereitstellung von gemeinsamer Funktionalität in Pools
  - Zugriff über Interfaces
  - Späte Bindung zur Laufzeit
- Vorteile
  - Änderungen an einem Modul wirken sich nicht auf andere Teile aus
  - Funktionalität kann zur Testzeit ausgetauscht werden
  - Alle Teile können getrennt voneinander versioniert werden

Architektur



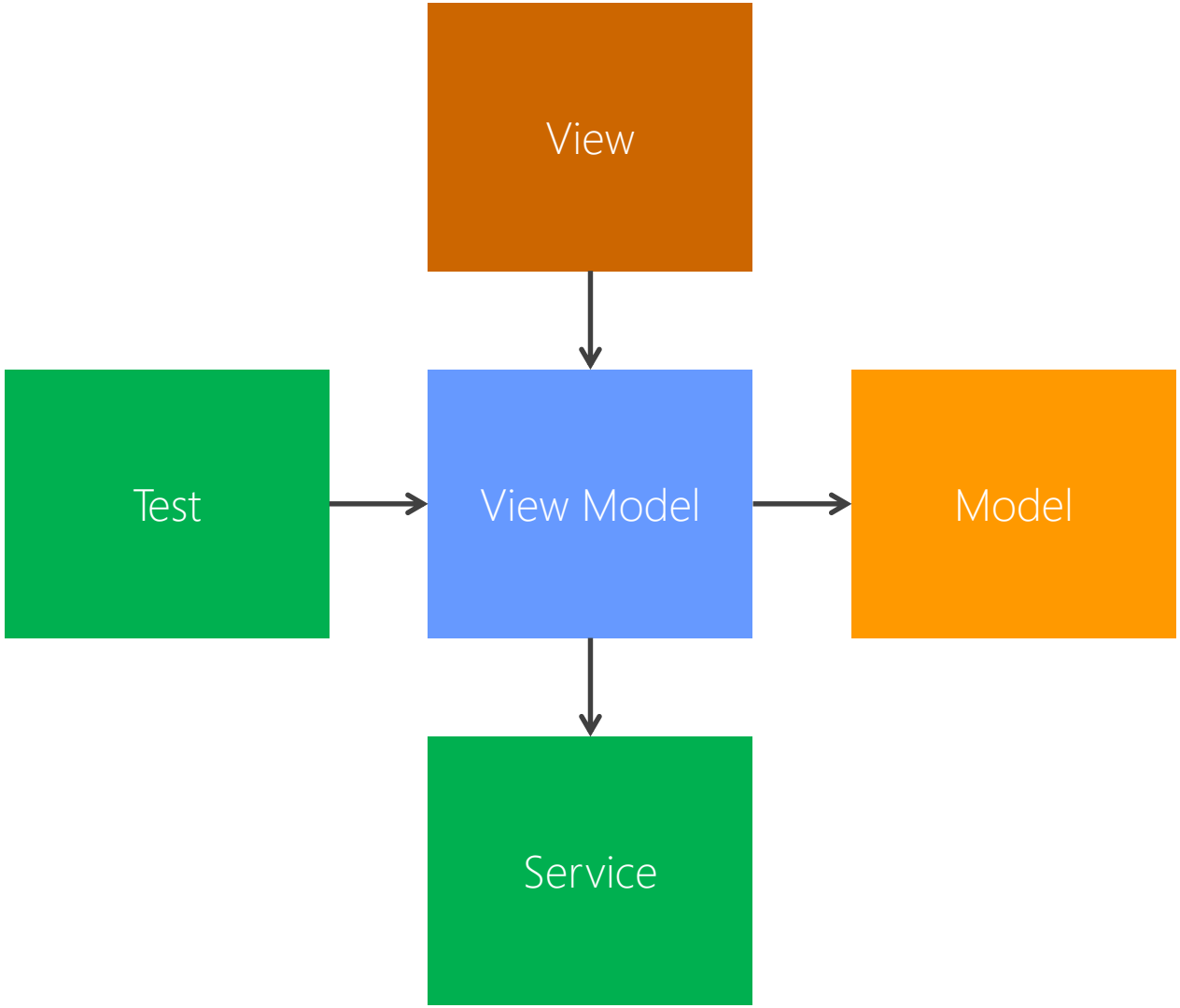
Techniken & Patterns

- Testbarkeit
  - Modul-View-ViewModel
  - Dependency Injection
- Modularisierung
  - Module Management
  - Service Pooling
  - Message Aggregation
- UI-Separation
  - Resource Management
  - State Management
  - Behaviors

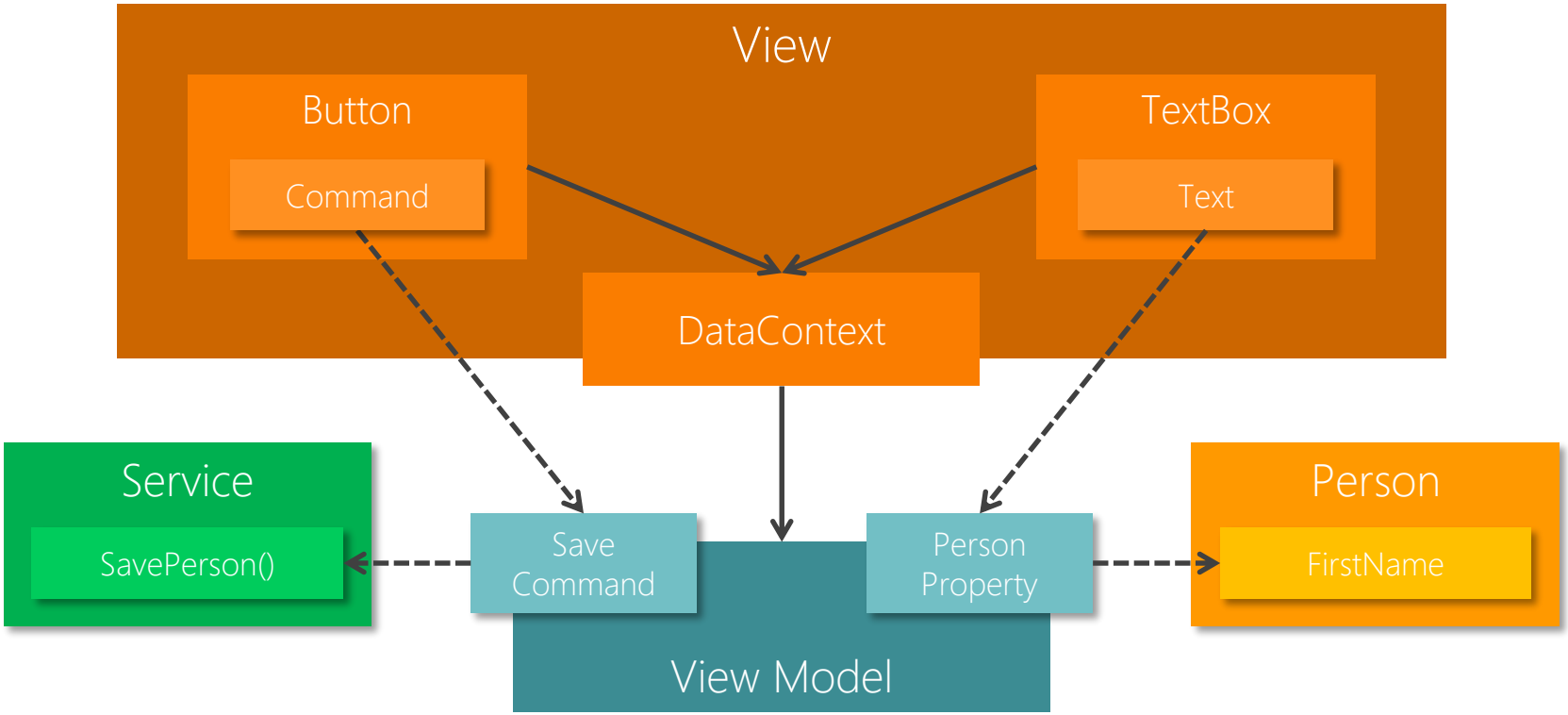
Das MVVM-Pattern

- Model-View-ViewModel
- Trennung von UI und Logik
- Ermöglicht bessere Testbarkeit
- Bessere Portierbarkeit
- Basiert auf Data Binding und Commands

MVVM Architecture



Funktionsweise



Voraussetzungen

- Synchronisieren von UI & Model
  - INotifyPropertyChanged-Implementierung
  - Basisklasse für Model und ViewModel
- Commands kommunizieren
  - ICommand-Implementierung

Demo

MVVM

Implementierung

- Implizite ViewModel-Binding
  - Die View wird instanziiert
  - ViewModel wird über XAML instanziiert und gebunden
- Explizite ViewModel-Binding
  - Das ViewModel wird instanziiert
  - View wird vom ViewModel instanziiert und gebunden

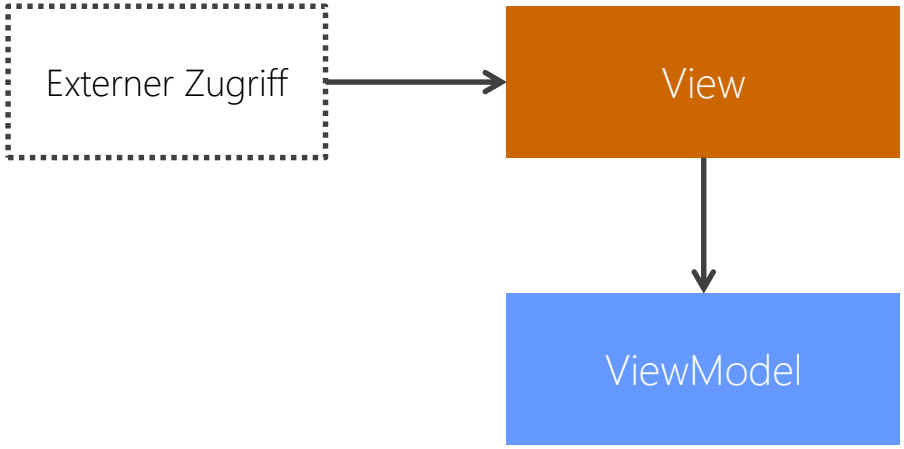
Implizite ViewModel-Bindung

```
<Window x:Class="MyApp.Views.MainView"
  xmlns:vm="clr-namespace:MyApp.ViewModels" … >

  <Window.Resources>
    <vm:MainViewModel x:Key="viewmodel" />
  </Window.Resources>

  <Grid DataContext="{StaticResource viewmodel}">
    …
  </Grid>
</Window>
```

```
public class PersonViewModel : ViewModelBase
{
  …
}
```

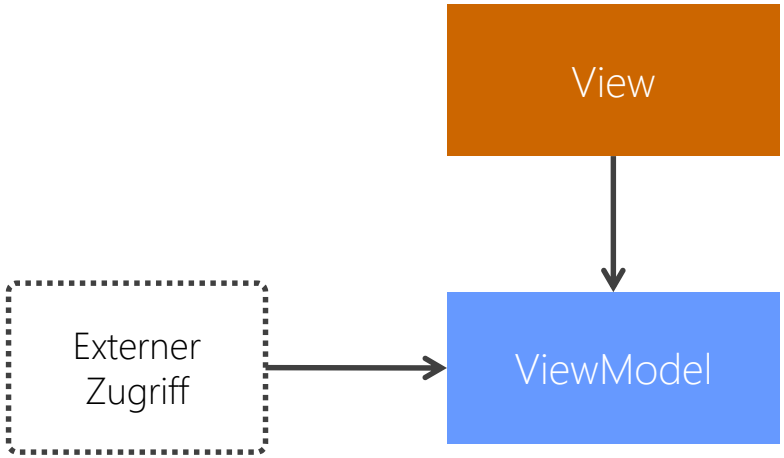


Explizite ViewModel-Bindung

```
public class PersonViewModel : ViewModelBase
{
  private PersonView _view;
  public void Initialize()
  {
    if (_view == null)
    {
      _view = new PersonView();
      _view.DataContext = this;
    }
    …
```

```
<UserControl …
  x:Class="MyApp.Views.PersonView">

  <Grid>
    …
  </Grid>
</UserControl>
```



Anforderungen an Datenklassen

- Synchronisation
  - INotifyPropertyChanged
- Transaktionssteuerung / Undo
  - IEditableObject
- Validierung
  - IDataErrorInfo, INotifyDataErrorInfo

Anforderungen an Datenlisten

- Synchronisieren von UI & Model
  - INotifyCollectionChanged
  - ObservableCollection<T>
- Navigieren, Sortieren, Filtern
  - ICollectionView
  - ListCollectionView
- Editieren
  - IEditableCollectionView
  - ListCollectionView

Demo

Umgang mit Listen

Zustände der Oberfläche steuern

- Zustände werden über XAML definiert
  - Zustände werden über VisualStateManager definiert
  - Ein State enthält ein Storyboard
  - Eigenschaften werden über Animationen verändert
- Umschalten von States
  - Steuerung über eine State-Eigenschaft das ViewModels
- Herausforderung
  - VisualStateManager benötigt ein UI-Element für die Umschaltung

Demo

State Management

UI-Interaktionslogik

- Verhalten von Controls beeinflussen
  - Auswahl eines Bildes über einen Dateidialog
  - Focus-Steuerung
  - Tastatur- und Mausereignisse
  - …

UI-Interaktionslogik

- Zuweisung
  - Kann nicht im ViewModel erfolgen
  - Sollte per Markup zugewiesen werden
- Behaviors
  - Attached Properties
  - Expression Blend Behaviors

Demo

UI-Interaction Logic

Modularisierung

- Modularisierung
  - Anwendung besteht aus einer generischen Shell
  - Fachliche Module werden als atomare Einheiten entwickelt
  - Dynamisches Nachladen der Module und Services
- Das Managed Extensibility Framework
  - Teil von .NET 4.0/4.5 & Silverlight 4/5
  - Basiert auf Attributen
  - Kann flexibel erweitert werden

MEF

- Das Managed Extensibility Framework
  - Teil von .NET 4.0/4.5 & Silverlight 4/5
  - System.ComponentModel.Composition
  - Dynamisches Laden und Instaziieren
  - Basiert auf Attributen
  - Kann flexibel erweitert werden

Begriffsdefinition

- Export
  - Exportierte Typen (Plug-Ins)
- Imports
  - Verweis auf exportierte Typen
- Parts
  - Imports und Exports
- Composition
  - Verbinden von Parts

Imports und Exports

- Exports werden geladen, instanziiert und mit Import verbunden
- Export definieren

```
[Export]
public class MyModule
{
  …
}
```

- Import definieren

```
public class Program
{
    [Import]
    public MyModule Module { get; set; }
}
```

Interface-basierte Exports

- Interface definieren

```
public interface IModule
{
  void Initialize();
  FrameworkElement GetView();
}
```

- Exports definieren

```
[Export(typeof(IModule))]
public class Module1 : IModule
{
  …
}
```

```
[Export(typeof(IModule))]
public class Module2 : IModule
{
  …
}
```

- Import definieren

```
public class Program
{
  [ImportMany]
  public IEnumerable<IModule> Modules { get; set; }
}
```

Composition

- Das Zusammenführen von Import und Exports zur Laufzeit
- Wird von CompositionContainer vollzogen
  - Option 1: Import/Export-Attribute
  - Option 2: Programmatische Composition

Catalogs

- Zum Auffinden von Parts werden Catalogs verwendet
  - Ermitteln alle Exports in einem bestimmten bereich
  - Stellen Metadaten für die Instanziierung bereit
- Standardkataloge
  - AssemblyCatalog
  - DirectoryCatalog
  - TypeCatalog
  - AggregatingCatalog

Demo

MEF

Lazy Loading

- Instanz wird erst erzeugt wenn ein Zugriff erfolgt
- Verwendung der Klasse **Lazy<T>**

```
public class Program
{
  [ImportMany]
  public IEnumerable<Lazy<IModule>> Modules { get; set; }
}
```

- Zugriff:

```
IModule module = this.Modules.First().Value;
```

Part Lifetime

- Standardmäßig cached der CompositionContainer die Instanzen
  - Keine automatische Garbage Collection
  - Die Lebensdauer von Parts kann explizit gesteuert werden

- **PartCreationPolicy**-Attribut im Export

```
[ImportMany(
  RequiredCreationPolicy = CreationPolicy.NonShared)]
  public IEnumerable<Lazy<IModule>> Modules { get; set; }
```

- **RequiredCreationPolicy**-Parameter im Import-Attribut

```
[Export(typeof(IModule))]
[PartCreationPolicy(CreationPolicy.NonShared)]
public class Module1
{ … }
```

Demo

Lazy Loading

Metadaten

- Exports können Metadaten bereitstellen
- Definition erfolgt über das **ExportMetadata**-Attribut

```
[Export(typeof(IModule))]
[ExportMetadata("DisplayName", "Buchungen")]
[ExportMetadata("DisplayIndex", 0)]
public class Module1 { … }
```

- Metadaten importieren

```
[ImportMany]
public IEnumerable<Lazy<Imodule, IDictionary<string, object>> Modules { get; set; }
```

- Zugriff

```
string displayName = this.Modules.First().Metadata["DisplayName"];
```

Metadaten

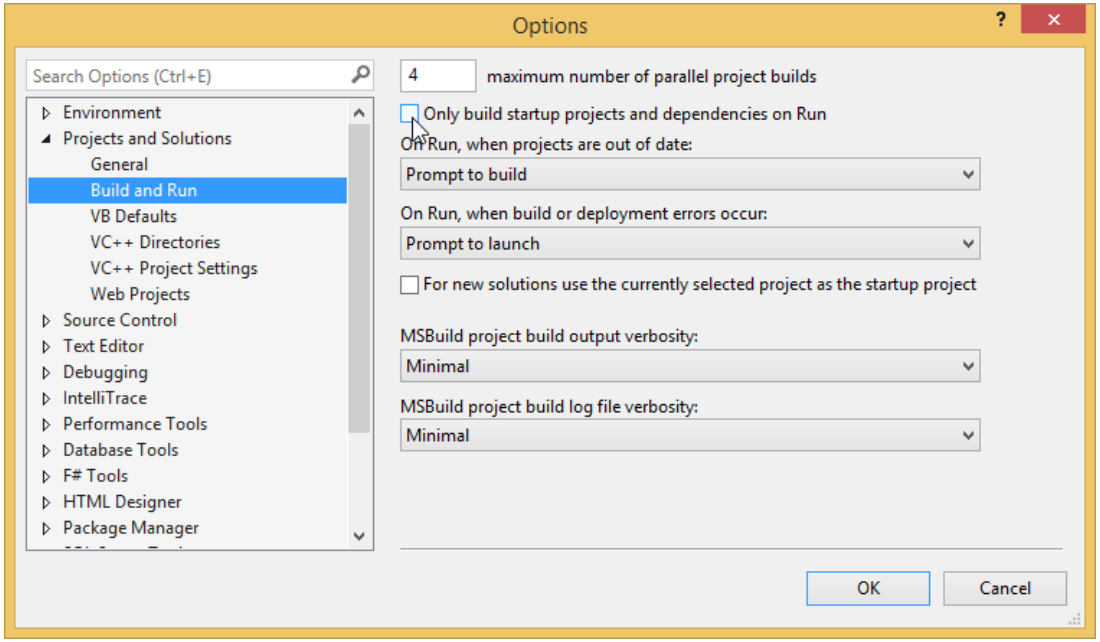
- Nachteile
  - Untypisierter Zugriff
  - Keine Garantie, dass Metadaten vom Export bereitgestellt werden
- Alternative: Benutzerdefiniertes Metadaten-Attribut erstellen
  - Erstellung eines Interface zur Beschreibung der Metadaten
  - Ableitung von **ExportAttribute**-Klasse
  - Bereitstellung der Metadaten über den Konstruktor

Demo

MEF Metadata

Debugging von MEF-Projekten

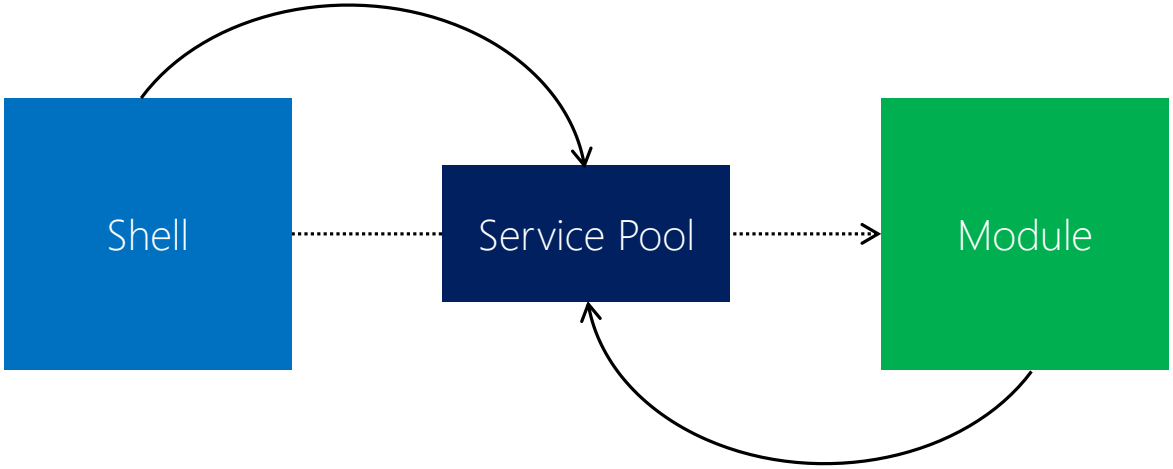
- Standardmäßig wird nur das Startprojekt kompiliert
  - Es kann vorkommen, dass zur Laufzeit keine Module vorhanden sind
- Visual Studio-Build-Option abschalten
  - „Only build startup projects and dependencies On Run“



Dynamische Service-Bindung

- Shell und Module nutzen gemeinsame Services
  - Fachliche Services / Service Proxies
  - Technische Services der Shell
- Erstellung sollte über Dependency Injection erfolgen
  - Keine direkte Instanziierung
  - Abstraktion durch Interfaces
  - Nutzung von Service-Mocks zur Testzeit
  - Module können Services unterschiedlicher Versionen nutzen
- Lösungsansatz
  - Shell ermittelt zur Laufzeit vorhandene Services und stellt diese den Modulen über einen Pool zu Verfügung

Architektur



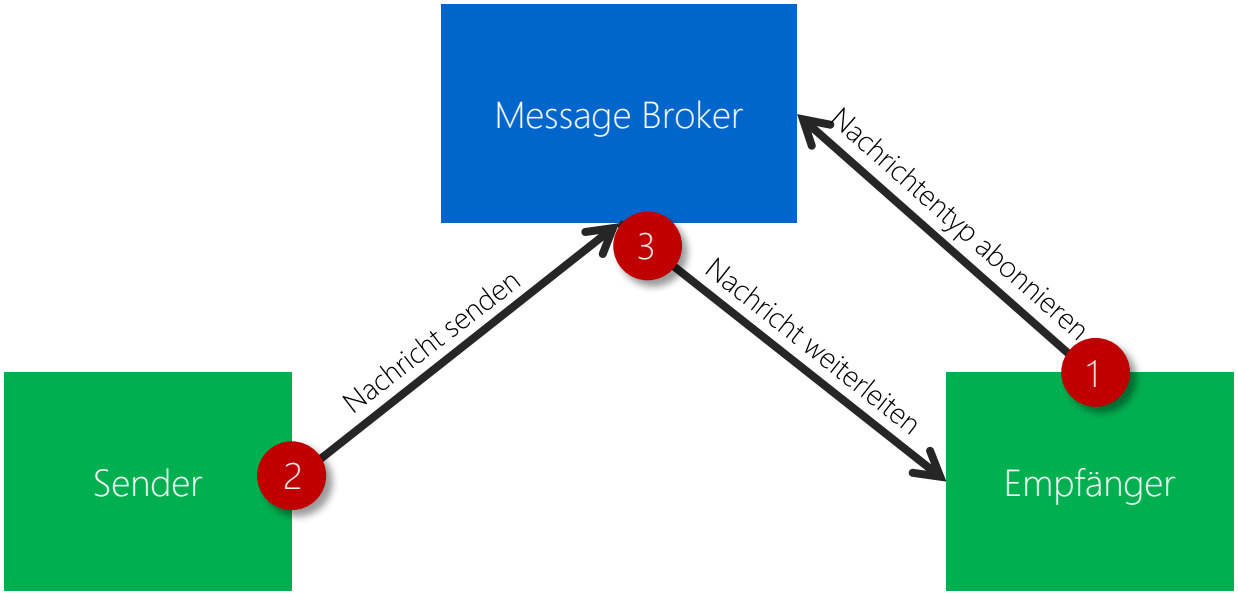
Demo

Service Pooling

Message Aggregation

- Shell und Module müssen miteinander kommunizieren
  - z.B. Anzeige von Statusmeldungen
- Messaging nach dem Publisher-/Subscriber-Prinzip
  - Ein Teil abonniert einen Nachrichtentyp
  - Ein anderer Teil sendet eine Nachricht
  - Das Messaging-System vermitteln
  - Zur Kommunikation werden Message-Typen verwendet

Funktionsweise



Demo

Messaging

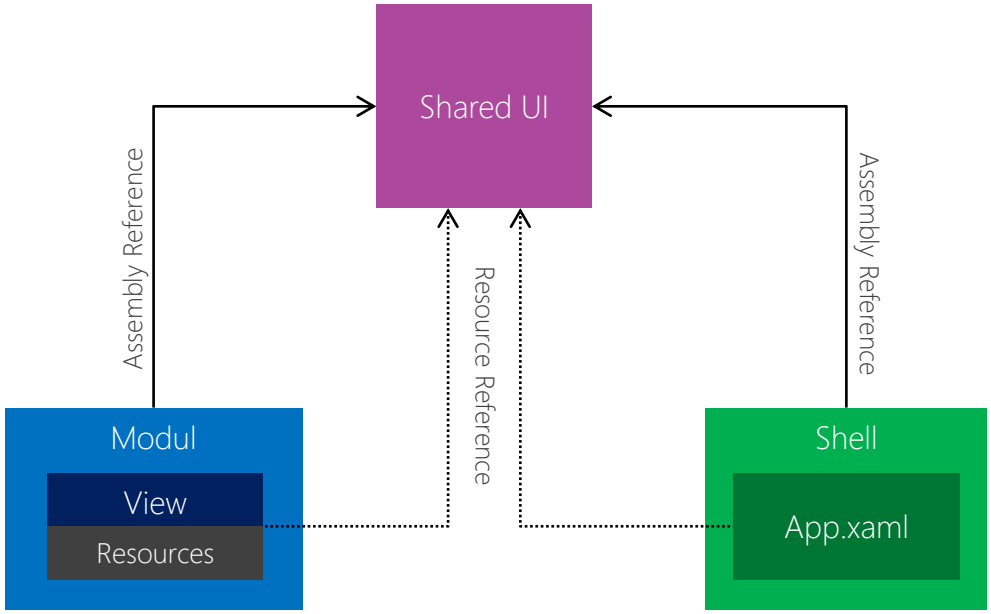
Umgang mit UI-Ressourcen

- UI-Ressourcen sollten zentral verwaltet werden
  - Werden von Shell und Modulen verwendet
  - Ermöglicht ein einheitliches Look & Feel
  - Erleichtert die Rollentrennung zwischen Entwickler und Designer
- Ressource-Typen
  - Farben, Fonts, Images, Größen
  - Styles & Templates
  - Custom Controls, Value Converter
- Fachliche Ressourcen werden auf Modulebene definiert
  - Data Templates, Data Template Selectors, User Controls
- Organisation
  - Definition der Assets in separaten Resource Dictionaries

Lösungsansatz

- Ressourcen werden in separater Assembly hinterlegt
  - Shell und Module referenzieren diese Assembly
- XAML-Ressourcen müssen eingebunden werden
  - Über Merged Resource Dictionaries
- Shell
  - app.xaml
- Module
  - Resource Dictionary der View

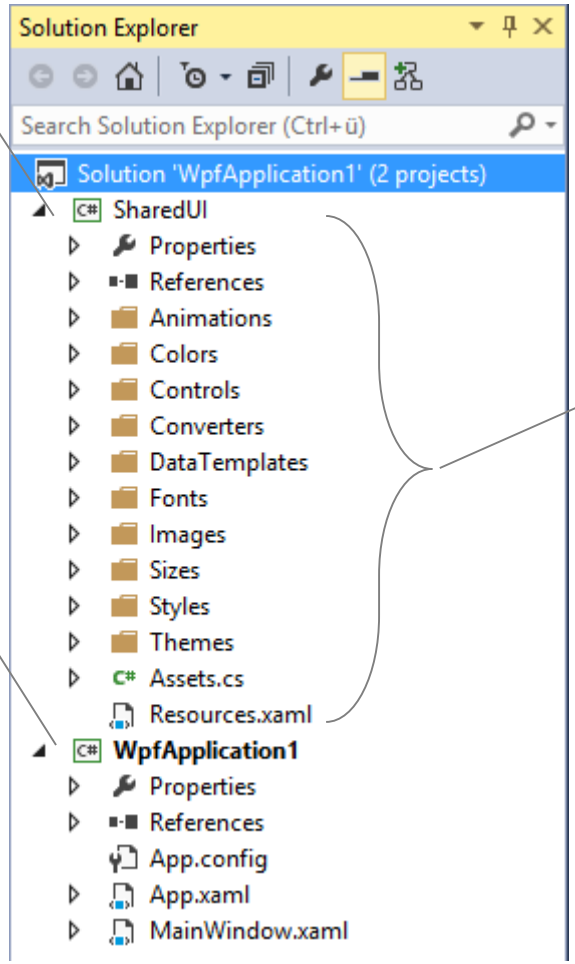
Umgang mit UI-Ressourcen



Aufbau des UI-Projekts

UI Ressourcen

App



Farbschema der App
Größen, Abstände, …
Styles & Templates
Bilder,
Konverter, …

Demo

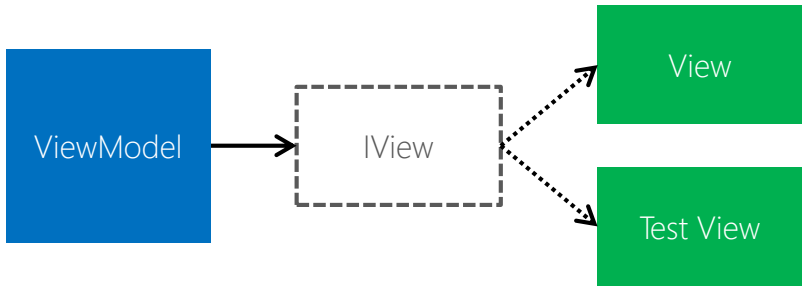
UI Separation

Tipps & Tricks

- Automatische View-Bindung
- Asynchrone Validierung

View-Bindung in MVVM

- Herausforderung
  - View-Implementierungen verschleiern
  - Kein UI-Zugriff in Unit Tests
  - Kein direkter Zugriff auf die View aus dem ViewModel
- Lösung
  - Factory-Klasse findet autom. die zugehörige View
  - Konventionsbasierte Ermittlung mit MEF 2.0



Demo

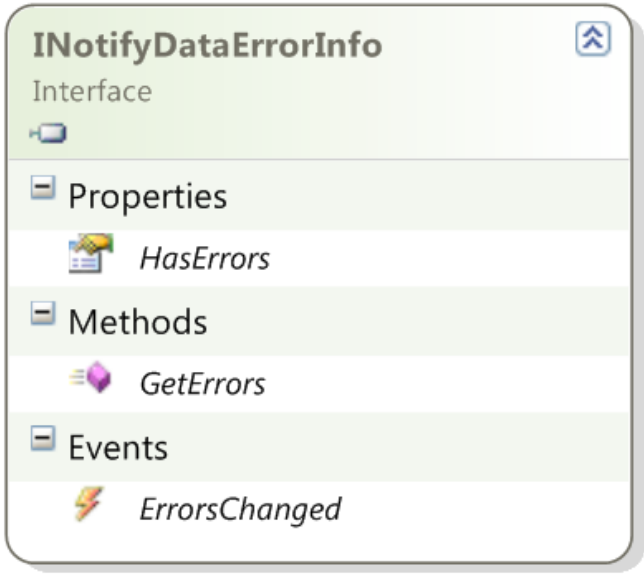
Automatische View-Bindung

Asynchrone Validierung

- INotifyDataErrorInfo-Interface
  - Nachfolger von IDataErrorInfo
  - Verfügbar in WPF & Silverlight
- Ermittelt eine Liste von Fehlern
- Ermöglicht asynchrone Validierung

INotifyDataErrorInfo

- HasErrors
  - Zeigt an, ob das Objekt Fehler enthält
- GetErrors()
  - Gibt eine Liste der Fehler für eine Eigenschaft zurück
- ErrorsChanged
  - Wird ausgelöst, wenn sich der Fehlerstatus ändert



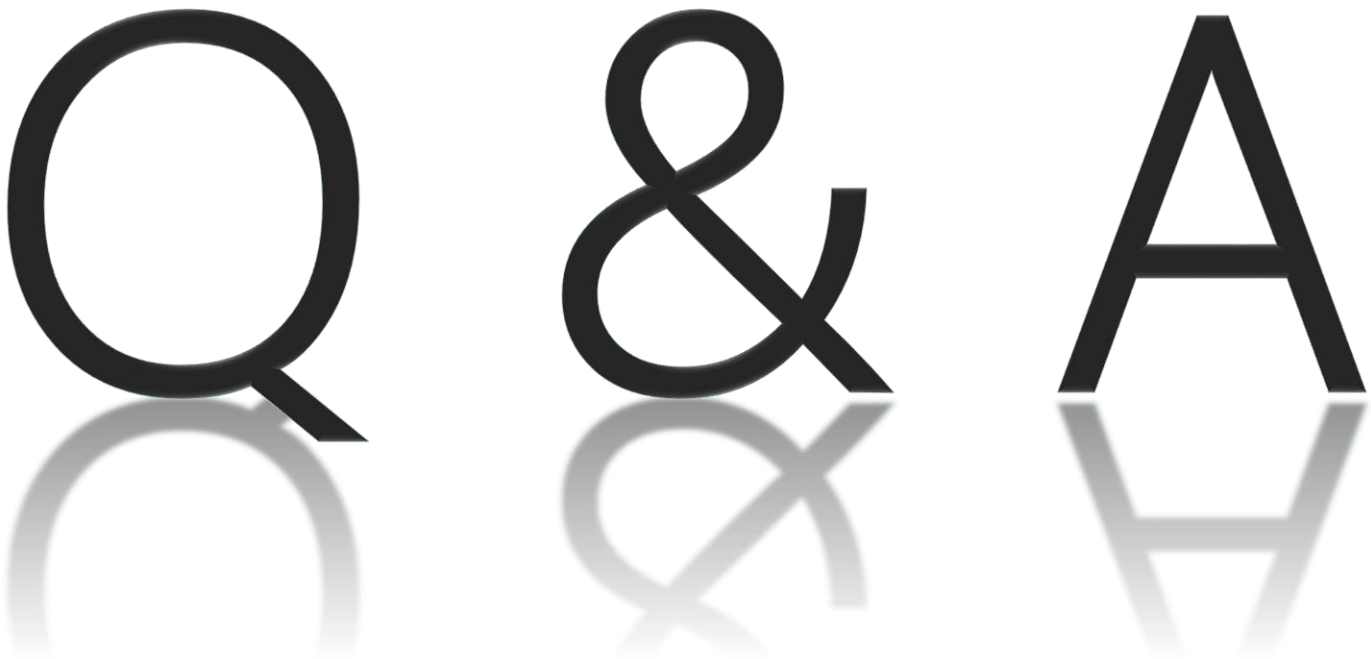
INotifyDataErrorInfo

- Datenklasse implementiert das Interface
- Validierung am Binding aktivieren
  - ValidatesOnNotifyDataErrors=True

```
<TextBox Text="{Binding Path=Name,
                        Mode=TwoWay,
                        ValidatesOnNotifyDataErrors=True}"/>
```

Demo

Eingabe-validierung


Q & A

Sie brauchen Unterstützung?

- Training, Coaching, Entwicklungsunterstützung
  - joerg.neumann@acando.de