

# dnead Zusammenfassung

Jan Fässler, Jonas Schwammburger

5. Semester (HS 2013)

# Inhaltsverzeichnis

<b>1 Overview</b>	<b>1</b>
1.1 .Net Development Frameworks . . . . .	1
1.1.1 CLR, CIL . . . . .	1
1.1.2 Strategie, Nutzen und Trends . . . . .	1
1.1.3 Managed und Unmanaged . . . . .	1
1.1.4 Sicherheit . . . . .	2
1.1.5 Attribute . . . . .	2
1.2 .NET Compact Framework . . . . .	2
1.3 SharePoint . . . . .	2
1.4 BizTalk . . . . .	3
1.5 Summary . . . . .	4
<b>2 Team Foundation Server</b>	<b>5</b>
2.1 Features . . . . .	5
2.1.1 Versionskontrolle . . . . .	5
2.1.2 TFS-Build . . . . .	5
2.1.3 Reports . . . . .	6
2.1.4 Benutzerverwaltung . . . . .	6
2.1.5 Serveraufbau . . . . .	6
<b>3 Windows Presentation Foundation</b>	<b>7</b>
3.1 Leistungsmerkmale . . . . .	7
3.1.1 Eigenschaften und Ereignisse . . . . .	7
3.1.2 Grafik . . . . .	7
3.1.3 Interoperabilität . . . . .	8
3.1.4 Medien und Dokumente . . . . .	8
3.1.5 Text und Typographie . . . . .	8
3.1.6 Benutzerschnittstelle . . . . .	8
3.2 XAML . . . . .	8
3.2.1 Declarative Programming . . . . .	8
3.2.2 XAML vs. Code . . . . .	9
3.2.3 Namespaces . . . . .	9
3.2.4 Property Element Syntax . . . . .	9
3.3 XAML compilation . . . . .	10
3.4 XAML Layout . . . . .	10
3.4.1 Panels . . . . .	10
3.4.2 Attached Properties . . . . .	10
3.4.3 Margin, Padding und Alignment . . . . .	11
3.4.4 Transformation . . . . .	11
3.5 Data Binding . . . . .	11
3.5.1 DataContext . . . . .	12
3.5.2 UpdateTrigger & Binding Direction . . . . .	12
3.5.3 INotifyPropertyChanged . . . . .	12
3.5.4 ObservableCollection . . . . .	13
3.6 Value Converter . . . . .	13
<b>4 Model-View-ViewModel</b>	<b>14</b>
4.1 Einleitung . . . . .	14
4.1.1 Pattern . . . . .	14
4.1.2 User Interaction . . . . .	14
4.1.3 Overview . . . . .	14
4.2 Commands . . . . .	15
<b>5 Entity Framework</b>	<b>16</b>
5.1 ADO.NET . . . . .	16
5.1.1 Architecture . . . . .	16

5.1.2	Data Provider . . . . .	16
5.2	Considerations . . . . .	16
5.2.1	Sample . . . . .	16
5.2.2	the base for ORMs . . . . .	17
5.3	Entity Framework . . . . .	17
5.3.1	Overview . . . . .	17
5.3.2	Architecture . . . . .	18
5.3.3	Entity Data Model . . . . .	19
5.3.4	Benefits & Considerations . . . . .	19
<b>6</b>	<b>Dependency Injection</b>	<b>20</b>
6.1	DI and IoC . . . . .	20
6.1.1	Overview . . . . .	20
6.1.2	Implications . . . . .	20
6.1.3	Advantages of IoC . . . . .	20
6.2	The DI Container Unity . . . . .	20
6.2.1	Options and Types . . . . .	20
6.2.2	Options . . . . .	21
6.2.3	Pros & Cons . . . . .	21
6.2.4	Services of IoC Container . . . . .	21
<b>7</b>	<b>ASP.NET MVC</b>	<b>22</b>
7.1	Controllers . . . . .	22
7.1.1	Actions . . . . .	22
7.1.2	Ajax . . . . .	22
7.1.3	View typed . . . . .	23
7.1.4	View untyped . . . . .	23
7.2	Views . . . . .	23
7.2.1	Code Blocks . . . . .	23
7.2.2	HtmlHelpers . . . . .	24
7.2.3	Layout Pages . . . . .	24
7.3	Routes . . . . .	24
7.4	Model . . . . .	24
<b>8</b>	<b>Codebeispiele</b>	<b>24</b>
8.1	WPF XAML . . . . .	24

# 1 Overview

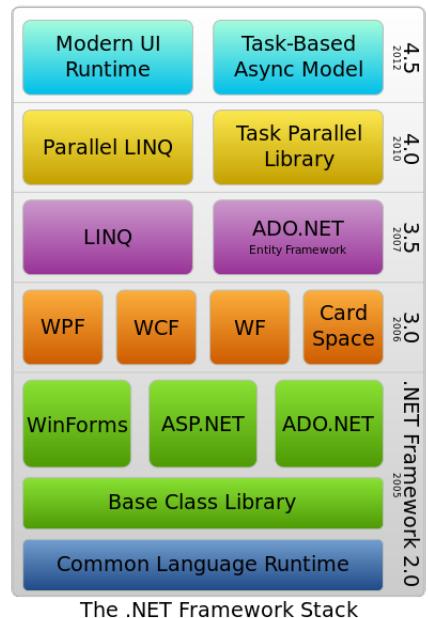
## 1.1 .Net Development Frameworks

Die .NET-Plattform ist die Umsetzung des Common-Language-Infrastructure-Standards (CLI) und stellt mit diesem eine Basis zur Entwicklung und Ausführung von Programmen dar, die mit unterschiedlichen Programmiersprachen auf verschiedenen Plattformen erstellt wurden. Hauptbestandteile sind die (objektorientierte) Laufzeitumgebung Common Language Runtime (CLR), die Base Class Library (BCL) sowie diverse Hilfsprogramme zum Beispiel zur Rechteverwaltung.

### 1.1.1 CLR, CIL

Die Common Language Runtime (CLR) ist die Laufzeitumgebung von .NET und stellt somit den Interpreter für den standardisierten Zwischencode, die Common Intermediate Language (CIL), dar. Die CIL hieß früher Microsoft Intermediate Language (MSIL), wurde aber im Rahmen der Standardisierung durch die Ecma International umbenannt. Für sie wurde ein sprachübergreifendes System von objektbasierten Datentypen definiert, so dass für alle Hochsprachen, die sich an den Common Language Infrastructure-Standard (CLI) halten, gültiger CIL-Bytecode erstellt werden kann.

.NET wurde von Anfang an dafür entwickelt, dass Programmierer in unterschiedlichen Programmiersprachen arbeiten können. Jede dieser Hochsprachen wird von .NET dann in die CIL übersetzt.



### 1.1.2 Strategie, Nutzen und Trends

Das Besondere an der CLR ist weniger die technische Innovation als vielmehr die strategische Entscheidung von Microsoft für ein laufzeitbasiertes System. Es soll unter anderem helfen, Systemabstürze zu vermindern, da die Runtime Applikationsfehler fangen kann. Damit entschied sich Microsoft erstmals gegen die bisher angewandte direkte Kompilierung in den Maschinencode des Zielsystems. Zusammen mit der Marktmacht von Java und dem Erfolg von Skriptsprachen ist damit ein Trend zu identifizieren. Dieser stellt einen Bruch mit den direktkompilierenden Programmiersprachen (insbesondere C++ und C) dar.

Mittels sog. „Reflection“ ist es möglich, zur Laufzeit Programmcode über ein Objektmodell zu generieren und es direkt im Speicher in lauffähigen Code zu überführen.

### 1.1.3 Managed und Unmanaged

Die .NET-Terminologie unterscheidet dabei zwischen Bytecode, welcher von der CLR verwaltet und in Maschinensprache umgesetzt wird – sogenannter Managed Code (v. engl. managed code ‚verwaltete Maschinensprache‘) –, und Teilen, die nicht innerhalb der CLR ausgeführt werden (unmanaged). Daneben gibt es noch die Möglichkeit in .NET unsicheren Code zu schreiben, um weiterhin z. B. klassische Zeiger-Operationen direkt auf einem Speicherbereich durchführen zu können.

Mit Hilfe der Interop-Technik lassen sich alle klassischen, binär kompilierten Windows-Bibliotheken mit .NET-Kapseln (sogenannten „Wrappern“) versehen und danach deren Programmfunctionen wie normale .NET Programmfunctionen aufrufen. Technisch gesehen gibt die CLR allerdings im Moment des Aufrufs einer Funktion einer nicht überwachten DLL einen großen Teil der Kontrolle über den Programmfluss ab.

Umgekehrt lassen sich auch .NET-Funktionen wie COM-Funktionen aufrufen. Damit soll eine fließende Migration von Software-Projekten auf .NET ermöglicht werden und die Integration von .NET-Modulen in eine bestehende Umgebung erleichtert werden.

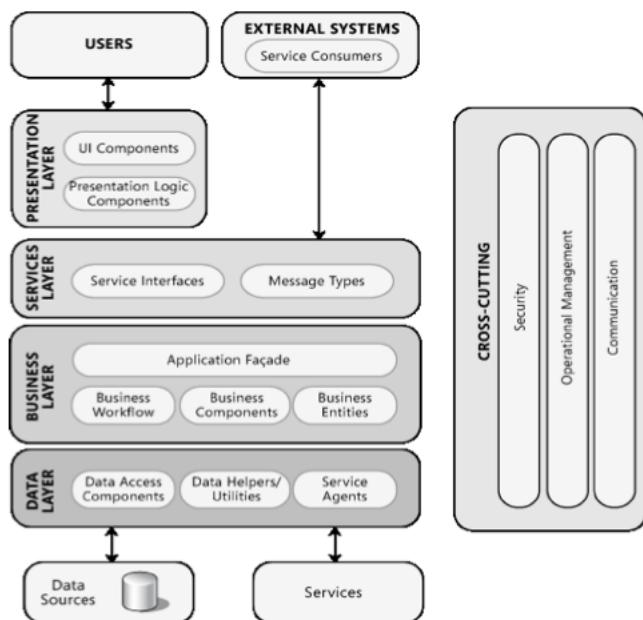
#### 1.1.4 Sicherheit

Eines der wichtigsten Konzepte von .NET ist die Sicherheit. Das Sicherheitskonzept beginnt bei Mechanismen, die die Identität des Programmherstellers gewährleisten sollen (Authentizität), geht über in solche zum Schutz der Programme vor Veränderung (Integrität) und reicht bis hin zu Techniken, die den Ort der Herkunft bzw. Programmausführung (zum Beispiel das Internet) einbeziehen. Es gibt sowohl ein codebasiertes (Code-based Security) als auch ein nutzerbasiertes (Role-based Security) Sicherheitsmodell.

#### 1.1.5 Attribute

Eine programmiertechnisch interessante Neuerung von .NET ist die Einführung von Attributen: gekennzeichnete Metadaten als Bestandteil der Programmiersprache. Beispielsweise können im Rahmen der komponentenbasierten Programmierung Komponenteneigenschaften ausgedrückt werden. Für die Verteilung, Installation und Konfiguration, für die Sicherheit, für Transaktionen und andere Programme können dem Code beschreibende Eigenschaften hinzugefügt werden.

Innerhalb eines Programmes kann mit Hilfe von Reflection auf die Attribute eines .NET-Programms, Assembly genannt, und die in ihr enthaltenen Elemente zugegriffen werden.



### 1.2 .NET Compact Framework

Das .NET Compact Framework ist ein Teil des .NET-Frameworks, der speziell für die Nutzung auf mobilen Endgeräten wie beispielsweise Pocket-PCs, Smartphones und PDAs ausgerichtet ist. Es soll Entwicklern erleichtern, Anwendungen für mobile Geräte zu schreiben oder sie auf diese zu portieren. Microsoft wird so dem eigenen Anspruch auf Plattformunabhängigkeit, den sie mit dem .NET Framework verfolgen, noch etwas mehr gerecht.

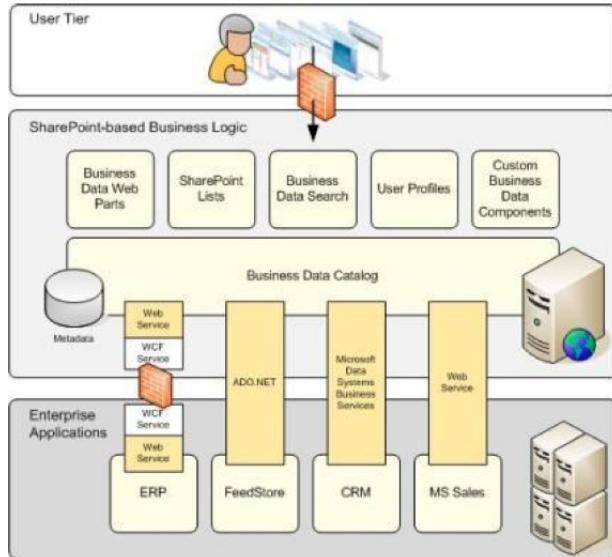
Technisch gesehen ist das Compact Framework eine Laufzeitumgebung bzw. eine Klassenbibliothek (vgl. Framework). Diese Version ist gegenüber dem normalen .NET-Framework für den PC um eine größere Zahl von Klassen reduziert, die für Kleingeräte nicht benötigt werden oder zu viel Speicherplatz belegen.

### 1.3 SharePoint

SharePoint ist eine Webanwendung von Microsoft, die folgende Anwendungsgebiete abdeckt:

- Zusammenarbeit, beispielsweise das Verwalten von Projekten oder die Koordination von Aufgaben,

- Soziale Netzwerke, beispielsweise über persönliche Webseiten, Team-Webseiten, Diskussionsgruppen und Blogs,
- Intranetportale,
- Content-Management über Dokumentenmanagement-Funktionen, Inhaltsverwaltung, Metadaten und benutzerangepasste Suchfunktionen,
- Geschäftsanwendungen.

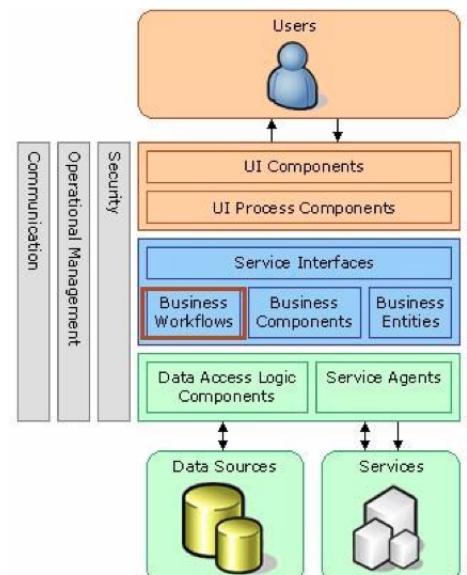


## 1.4 BizTalk

Microsoft BizTalk Server, oft einfach nur als „BizTalk“ bezeichnet, ist ein Enterprise Service Bus. Durch Adaptoren, die speziell darauf ausgelegt sind, in großen Unternehmen zwischen Systemen zu kommunizieren, wird eine Automatisierung von Geschäftsprozessen ermöglicht. Dieses Microsoft-Produkt enthält folgende Funktionen: Enterprise Application Integration (EAII), Business Process Automation, Business-to-business Communication, Message broker, and Business Activity Monitoring.

Im Standardfall ermöglicht BizTalk Firmen durch den Austausch von Geschäftsdokumenten, wie Bestellungen und Rechnungen zwischen zwei getrennten Applikationen, automatisierte Geschäftsprozesse zu integrieren und zu verwalten. Das ist innerhalb einer Organisation sowie über Unternehmensgrenzen hinweg möglich. Auf Anwender bezogene Prozesse können nicht direkt mit BizTalk implementiert werden. Das dafür entsprechende Microsoft-Produkt wäre hier der Microsoft-Sharepoint-Server.

Die Entwicklung für BizTalk erfolgt durch Microsoft Visual Studio. Ein Entwickler kann „Transformations Maps“ erzeugen, die einzelne Nachrichtentypen in andere verwandelt. Beispielsweise kann eine XML Datei in ein SAP-IDocs-Format übersetzt werden. Diese Maps können in Visual Studio mit einem grafischen Designer erstellt werden. Weitere Funktionalität kann über .NET Assemblies bereitgestellt werden, die von bestehenden Modulen, wie „Instance Maps“ oder Adaptoren aufgerufen werden können. Datentransformationen (Maps) und auch Prozesse ist durch sogenannte „Orchestrierungen“ organisiert, die eine Visualisierung des Prozesses ermöglicht.



## 1.5 Summary

- The .Net Framework includes **MS proprietary frameworks** for application development
- The application development frameworks are **not ECMA/ISO standard**
- Microsoft offers many **additional components** for Enterprise Application Development. Often still not .Net technology
- The **Microsoft Application Architecture Guide** describes reference architectures for different enterprise application scenarios.

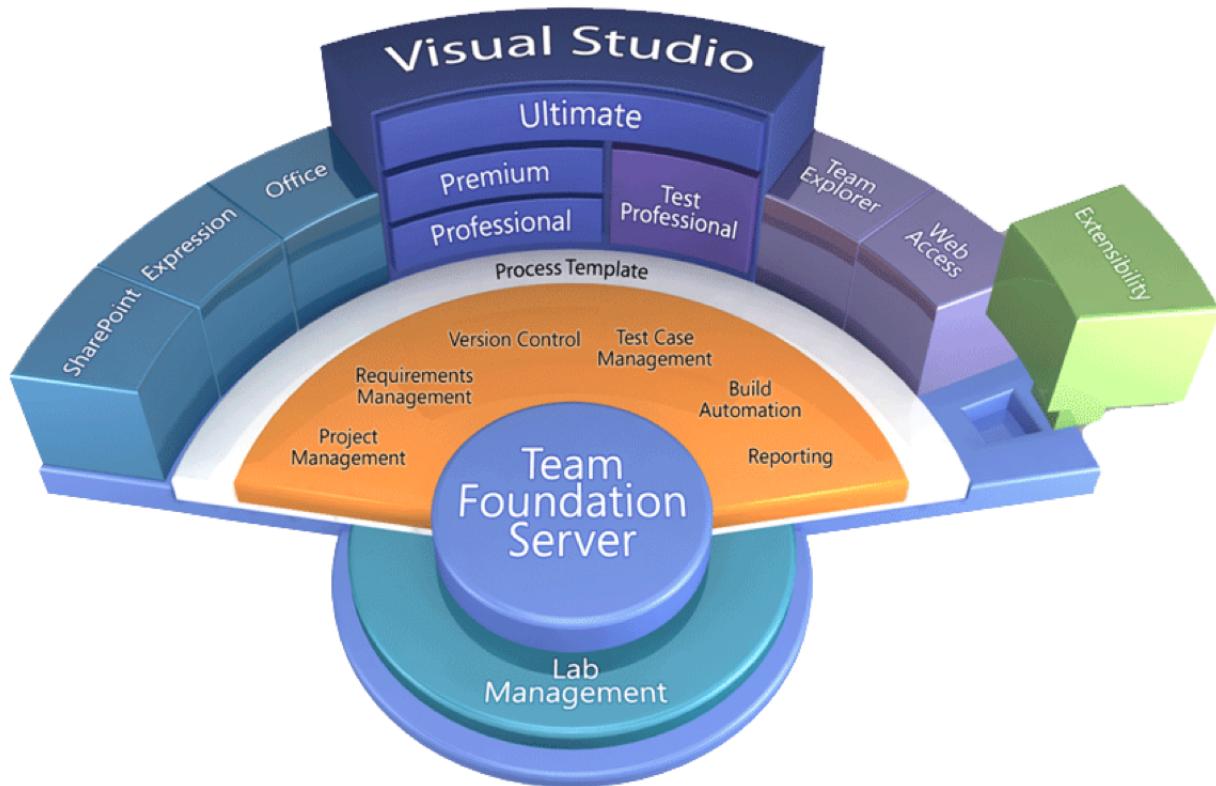
## 2 Team Foundation Server

Der Team Foundation Server (TFS) von Microsoft ist eine Windows-Plattform für kollaborative Softwareprojekte. Über den TFS können Projekte geplant, erstellt und verwaltet werden. Er kann dabei bis zu 2000 Entwickler und 500 Projekte verwalten.[ Für kleine Projekte gibt es die Workgroup-Edition, die maximal fünf Benutzer erlaubt.

Auf Basis der Prozessvorlagen unterstützt der TFS verschiedene Entwicklungsverfahren. Vorlagen für die Standardverfahren CMMI, Agile Softwareentwicklung oder Scrum werden mitgeliefert. Andere Hersteller bieten weitere Prozessvorlagen an. Alle Prozessvorlagen liegen in Form von XML-Dateien vor, so dass grundsätzlich ein (XML-)Editor für deren Bearbeitung ausreicht. Für eine einfachere und schnellere Anpassung steht allerdings ein Werkzeug zur Verfügung, mit dem die Anpassungen direkt in der Entwicklungsumgebung vorgenommen werden können. Die beim Prozess mitgelieferte Dokumentation („Process Guidance“) liegt statisch vor, kann aber dank verfügbaren Quelldateien angepasst und neu erstellt werden.

Die involvierten Teammitglieder können mit verschiedenen Werkzeugen (zum Beispiel Visual Studio, Excel, Project, Infopath, Office, Outlook oder Web) Prozessschritte bearbeiten und die entsprechenden Arbeitsschritte („workflows“) anstoßen. Die genannten Programme integrieren sich direkt in den TFS, so dass auf einer einheitlichen Plattform gearbeitet werden kann.

Bestandteile einer Prozessvorlage sind Work Items, Reports, Abfragen und diverse Dokumente.



### 2.1 Features

#### 2.1.1 Versionskontrolle

TFS integriert eine eigene Versionskontrolle für den Quellcode der verwalteten Projekte. Die gängigen Operationen eines Versionskontrollsysteams werden unterstützt.

#### 2.1.2 TFS-Build

Die Buildengine des TFS unterstützt das automatische Erstellen des entwickelten Produkts („build“), optional z.B. auch mit Dokumentation. Dabei kann man auch Unitests ausführen und Statistiken bzw. Berichte

generieren lassen.

### **2.1.3 Reports**

Über ein integriertes Data-Warehouse werden automatisch Berichte erstellt (etwa mit Metriken, Fehlerstatistik, Leistungsanalyse usw.) Die Berichte sind für unterschiedliche Zielpersonen zugeschnitten (Kostenverantwortliche, Entwickler, Projektleiter) und geben jeweils einen Überblick über den Projektstand. Technische Grundlage ist ein sogenannter "report server", der seine Ausgabe über einen Microsoft SharePoint Server generiert. Dadurch können die Berichte sowohl direkt als auch in Microsoft Project, Microsoft Excel und innerhalb von Microsoft Visual Studio benutzt werden.

### **2.1.4 Benutzerverwaltung**

Der TFS kann entweder als Server in einem Active Directory oder einzeln betrieben werden. Für die Benutzerverwaltung kennt der Server die Windows-Benutzer und Gruppen sowie weitere Gruppen im TFS. Beim Anlegen eines Projekts werden vier Gruppen automatisch erstellt: Lesezugriff, Schreibzugriff, Administratoren und eine interne Gruppe zum Buildmanagement.

Die Berechtigungen für den SSharepoint Server sowie das "reporting system" müssen vom Administrator von Hand gesetzt werden. Aus diesem Grund empfiehlt es sich, Windows-Gruppen zu definieren und zu verwenden.

### **2.1.5 Serveraufbau**

Der TFS ist auf dem Prinzip einer Schichtenarchitektur entwickelt worden. Anwendungs- und Datenschicht können auf einem einzelnen Server oder auf separaten Servern installiert werden.

Der TFS benötigt folgende Software:

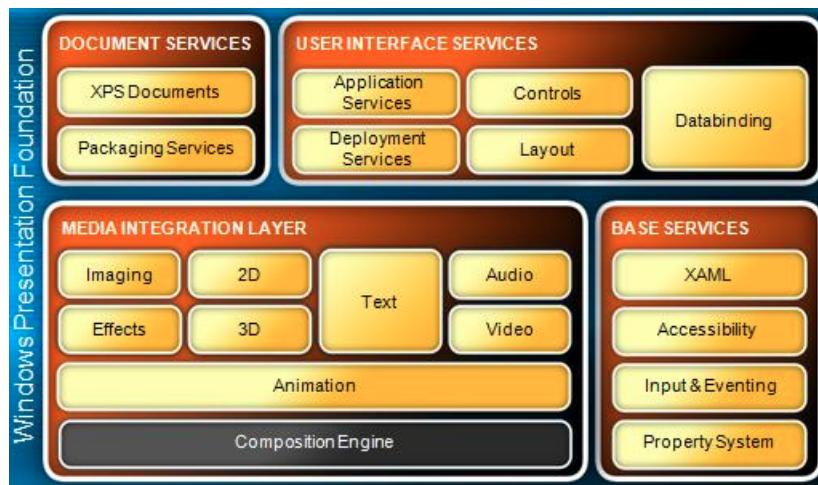
- Microsoft SQL Server für die Datenhaltung und das Data-Warehouse (x86 oder x64)
- Windows Server 2003, Windows Server 2008 für die Anwendung (x86 oder x64)
- TFS-Build entweder integriert oder separat
- Microsoft Internet Information Server und Windows SharePoint Services
- Microsoft Report Server

Ab der Version 2010 kann der Team Foundation Server auch auf einem Client-Betriebssystem installiert werden. Hierfür wird eine Basis-Konfiguration angeboten, welche die Express-Version des Microsoft SQL Servers zur Datenhaltung benutzt. Diese Installationsform ist für Einzelentwickler gedacht, die den Team Foundation Server benutzen wollen. Microsoft möchte hierdurch den Team Foundation Server als Nachfolger des Produkts Microsoft Visual SourceSafe etablieren, welches nicht mehr von Microsoft gepflegt wird.

Die einzelnen Komponenten, mit Ausnahme des Windows-Server-Betriebssystems und des SQL Servers, sind Bestandteil des Produkts.

### 3 Windows Presentation Foundation

Windows Presentation Foundation (kurz WPF), auch bekannt unter dem Codenamen Avalon, ist ein Grafik-Framework und Teil des .NET Frameworks von Microsoft, das mit Windows Vista, Windows 7 und Windows 8 ausgeliefert wird, sich aber auf Windows XP (bis zur Version 4.0) und Server 2003 nachinstallieren lässt. WPF stellt ein umfangreiches Modell für den Programmierer bereit. Dabei werden die Präsentation und die Geschäftslogik getrennt, dies wird vor allem durch die Auszeichnungssprache XAML (basierend auf XML) unterstützt. XAML beschreibt Oberflächen-Hierarchien deklarativ als XML-Code. WPF-Anwendungen können sowohl Desktop- als auch Web-Anwendungen sein und benutzen, wenn möglich, auch Hardwarebeschleunigung. Das Framework versucht, die verschiedenen Bereiche, die für die Präsentation wichtig sind (Benutzerschnittstelle, Zeichnen und Grafiken, Audio und Video, Dokumente, Typographie), zu vereinen.



#### 3.1 Leistungsmerkmale

##### 3.1.1 Eigenschaften und Ereignisse

Im Gegensatz zu normalen Anwendungen benutzt WPF eine eigene Art von Eigenschaften, dependency properties genannt. Diese sind notwendig, da in WPF einige Eigenschaften von anderen abhängig sein können, z. B. die Position eines Bilds während einer Animation. Zudem bieten diese Eigenschaften Unterstützung für Datenbindung und Validierung.

Auch die Ereignisse unterscheiden sich. In WPF werden so genannte routed events benutzt. Dies ergibt sich daraus, dass Elemente andere Elemente enthalten können. Wenn ein Kindelement ein Ereignis auslöst, so wird dieses auch an das Elternelement geleitet, um nicht jedes einzelne Kindelement zu überwachen. Dies nennt sich bubble event. Umgekehrt kann es sinnvoll sein, ein Ereignis als Elternelement vor dem Kindelement zu empfangen (tunnel event).

Dependency properties und routed events können auch attached sein, d. h. ein Element kann je nach Kontext eine Eigenschaft bzw. ein Ereignis von einem anderen Element erhalten. Dies ist z. B. der Fall, wenn eine Schaltfläche in einem Raster steckt: es werden Eigenschaften für die Positionierung (Spalte und Zeile) zur Verfügung gestellt.

##### 3.1.2 Grafik

Alle Grafikelemente (auch Fenster, etc.) werden via Direct3D gerendert.[1] Dies hat zur Folge, dass einige Aufgaben hardwarebeschleunigt von der GPU der Grafikkarte übernommen werden anstatt von der CPU. Zudem können 3D-Grafiken in 2D-Anwendungen angezeigt werden. Auch Vektorgrafiken werden unterstützt. Bis zur Version 3.5 der WPF werden Bitmap-Effekte angeboten, diese werden allerdings ohne Hardwarebeschleunigung gerendert,[2] weshalb sie in der aktuellen Version 4.0 als veraltet deklariert werden. Anstelle der Bitmap-Effekte sollen nun "normale" Effekte wie z. B. der DropShadowEffect verwendet werden, welche durchgängig die Hardwarebeschleunigung der Grafikkarte verwenden.

### 3.1.3 Interoperabilität

Windows-Forms-Steuerelemente können in WPF-Anwendungen benutzt werden; umgekehrt können auch WPF-Elemente in Windows Forms gehostet werden.

Zudem unterstützt WPF Win32: WPF ist mittels Hosting auch in Win32-Code benutzbar und Win32-Code kann auch in WPF-Anwendungen weiterbenutzt werden.

### 3.1.4 Medien und Dokumente

WPF stellt 2D-Primitive mit vordefinierten Transformationen, Texturen, etc. bereit. Die 3D-Funktionalitäten sind ein Unterteil von Direct3D. Diese Funktionalitäten sind allerdings auch für Dokumente und Benutzerschnittstellen verfügbar.

Auch individuelle Animationen sind möglich. Diese können auch zeitgesteuert ablaufen. Die meisten Grafikformate und Videos im WMV oder MPEG-Format werden unterstützt, wobei hierfür ein installierter Windows Media Player ab Version 9 notwendig ist.

Auch Dokumente, insbesondere XPS-Dokumente werden mit vordefinierten Steuerelementen unterstützt.

### 3.1.5 Text und Typographie

WPF unterstützt viele Features von OpenType, z. B. Ligaturen, Kapitälchen und Ruby. Es werden OpenType- und TrueType-Schriftarten unterstützt. WPF behandelt Text, da es auf .NET aufsetzt, immer als Unicode unabhängig von der Zeichenkodierung.

### 3.1.6 Benutzerschnittstelle

WPF enthält schon einige vordefinierte Steuerelemente, wie Menüs, Listen, etc.

Zudem wird das Aussehen von der Steuerelementlogik getrennt. Das Aussehen eines Steuerelements kann unabhängig davon mit Styles (Eigenschaften anpassen) und Templates (Festlegung, wie das Steuerelement aufgebaut ist) geändert werden.

Steuerelemente können beliebige andere Steuerelemente oder Inhalte (z. B. Bilder) enthalten.

## 3.2 XAML

- eXtensible Application Markup Language
- XML based language to instantiate and initialize Objects with hierarchical relationships
- Also used in WF (Workflow Foundation) und Silverlight

Listing 1: xaml

```
1 <Window xmlns="http://schemas.microsoft.com/winfx/...">
  <StackPanel HorizontalAlignment="Center" >
    <Image Source="Images/hello.jpg" Height="80" />
    <TextBlock Text="Welcome to WPF!" FontSize="14"/>
    <Button Content="OK" Padding="10,4" />
  </StackPanel>
</Window>
```

### 3.2.1 Declarative Programming

Markup for Windows

- Build applications in simple declarative statements
- Can be used for any CLR object hierarchy (not just WPF)

Code and content are strictly separate

- Streamline collaboration between designers and developers

### 3.2.2 XAML vs. Code

- What can be done in XAML can also be done in code
- XML-Tags correspond to objects (using the default constructor)
- XML-Attributes correspond to properties

Listing 2: XML

```
<StackPanel HorizontalAlignment="Center" >
    <TextBlock Margin="20">Hello</TextBlock>
3 </StackPanel>
```

Listing 3: C#

```
StackPanel stackPanel = new StackPanel();
2 TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to WPF";
stackPanel.Children.Add(textBlock);
```

### 3.2.3 Namespaces

XML-Prefixes correspond to CLR-Namespaces

Listing 4: Namespaces

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://
schemas.microsoft.com/winfx/2006/xaml" xmlns:cc="clr-namespace:MyCoolControls">
    <Grid>
        <cc:MyControl x:Name="myControl" />
    </Grid>
5 </Window>
```

### 3.2.4 Property Element Syntax

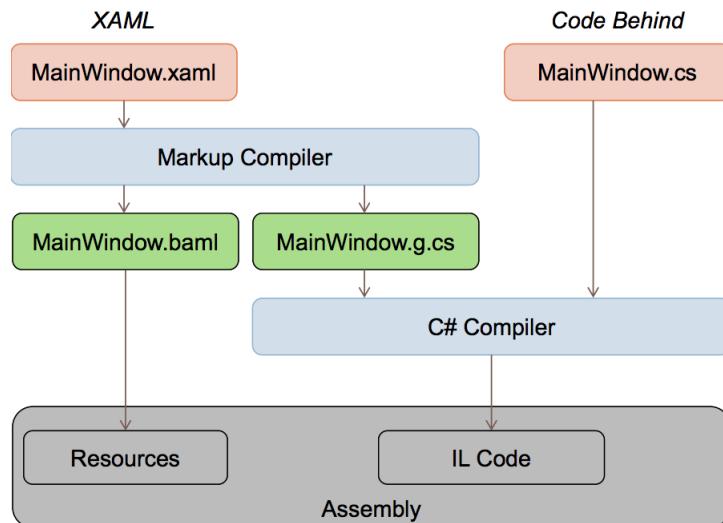
Complex properties can be defined as XML child elements

Listing 5: Property Element Syntax

```
<Rectangle Fill="Red" />

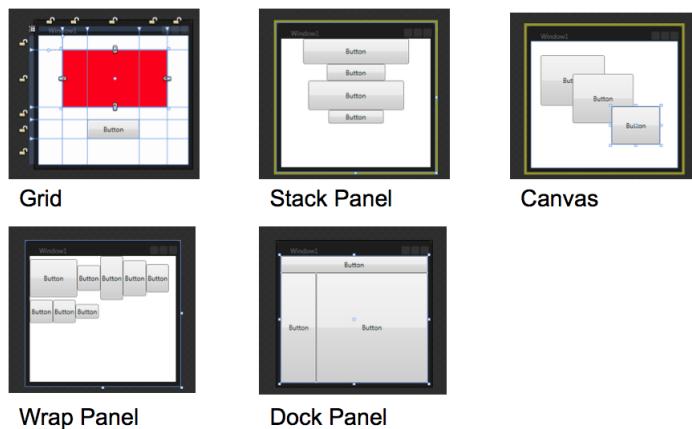
<Rectangle Width="20" Height="20">
    <Rectangle.Fill>
5     <LinearGradientBrush>
        <GradientStop Color="Red" Offset="0" />
        <GradientStop Color="Blue" Offset="1" />
    </LinearGradientBrush>
    </Rectangle.Fill>
10 </Rectangle>
```

### 3.3 XAML compilation



### 3.4 XAML Layout

#### 3.4.1 Panels



#### 3.4.2 Attached Properties

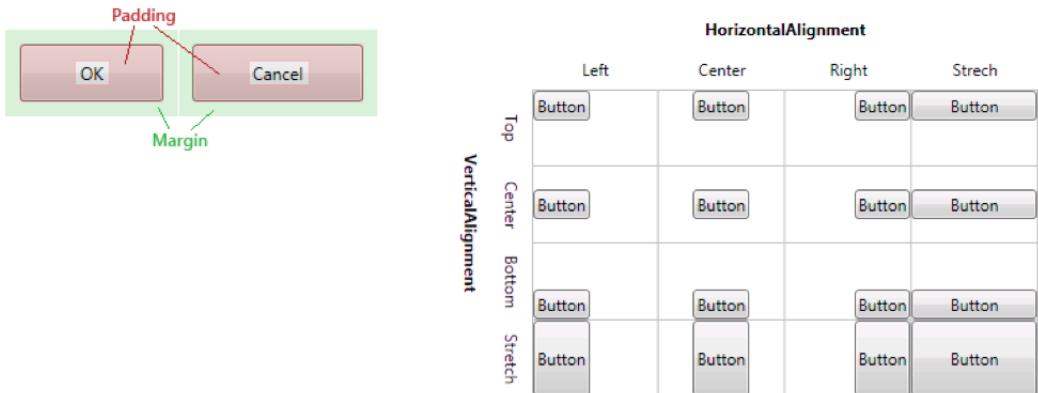
Attached Properties allow to add properties to WPF Controls

Listing 6: Attached Properties

```
<DockPanel>
    <Button DockPanel.Dock="Left" Content="Button" />
</DockPanel>

5 <Canvas>
    <Button Canvas.Top="20" Canvas.Left="20" Content="Button" />
</Canvas>
```

### 3.4.3 Margin, Padding und Alignment

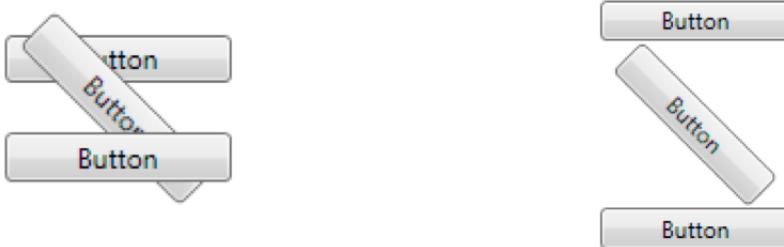


Listing 7: Alignment

```
<Button HorizontalAlignment="Left" VerticalAlignment="Top" Margin="8,0,8,8" >
    Test
</Button>
```

### 3.4.4 Transformation

- Element can be transformed in WPF
- LayoutTransform influences the layout, RenderTransform does not



```
<Button>
    <Button.RenderTransform>
        <RotateTransform Angle="30" />
    </Button.RenderTransform>
</Button>
```

```
<Button>
    <Button.LayoutTransform>
        <RotateTransform Angle="30" />
    </Button.LayoutTransform>
</Button>
```

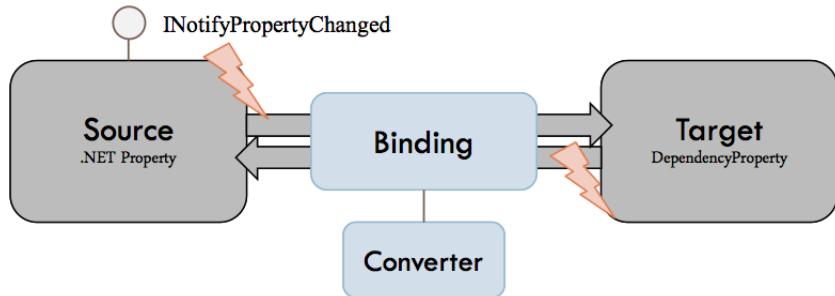
## 3.5 Data Binding

Markup Extensions are a kind of XAML- Macros, resolved at runtime.

Listing 8: Data Binding

```
<TextBlock Text="{Binding Path=Vorname}" />
```

- DataBinding synchronizes the values of two properties
- Typically a UI element is connected to an entity object from a datasource, e.g. the database
- Uni- oder bidirectional
- A ValueConverter can adapt the data format



Listing 9: Sources Data Binding

```

// Other WPF elements
{Binding Path=Text, ElementName(textBox}

4 // Parent WPF elements
{Binding Path=Text, RelativeSource={RelativeSource Mode=FindAncestor, AncestorType=ListBox
    }}

//Explicit Objekts
{Binding Path=Text Source={StaticResource myObject}}
9
//DataContext
{Binding Path=Text}

```

### 3.5.1 DataContext

- Every WPF element has a DataContext property
- The DataContext is inherited to children
- Allows the Binding to a Data-Object
- The default source of a {Bindings} is always the DataContext

### 3.5.2 UpdateTrigger & Binding Direction

**UpdateSourceTrigger** Defines when is the DataBinding is triggered:

- PropertyChanged
- LostFocus
- Explicit

**Mode** Defines the DataBinding direction:

- OneWay
- TwoWay
- OneWayToSource

### 3.5.3 INotifyPropertyChanged

Every Data-Object must implement INotifyPropertyChanged to allow the propagation of changes:

Listing 10: INotifyPropertyChanged

```

public class Customer : INotifyPropertyChanged {
    private string _name;
    public string Name {
        get { return _name; }
        set { _name = value; OnPropertyChanged("Name"); }
    }
}
```

```

        set {
            _name = value;
            NotifyPropertyChanged("Name");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged( string name) {
        if( PropertyChanged != null )
            PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
}

```

---

### 3.5.4 ObservableCollection

Use `ObservableCollections` to allow the propagation of collection changes:

Listing 11: ObservableCollection

```
ObservableCollection<Auction> auctions = new ObservableCollection<Auction>();
```

---

## 3.6 Value Converter

`ValueConverters` can change the format of the data in both directions:

Listing 12: Value Converter

```

public class BoolToVisibilityConverter : IValueConverter {
    #region IValueConverter Members
    public object Convert(object value, Type targetType, object parameter, CultureInfo
        culture) {
        4   return (bool) value ? Visibility.Visible : Visibility.Collapsed;
    }
    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
        culture) {
        throw new NotImplementedException();
    }
    9   #endregion
}

```

---

How to use a `ValueConverter` as a Resource in XAML:

Listing 13: Value Converter

```
<Window.Resources>
    <conv:BooleanToStatusTextConverter x:Key="booleanToStatusTextConverter" />
</Window.Resources>
<Button Content="{Binding IsOpen, Converter={booleanToStatusTextConverter}}" />
```

---

## 4 Model-View-ViewModel

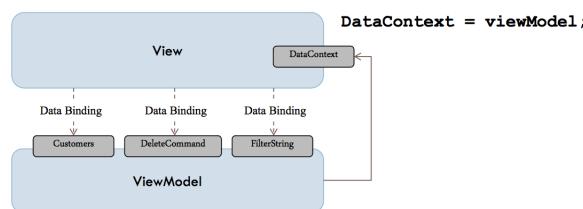
### 4.1 Einleitung

- Separation of GUI design ("style") and logic "behavior"  
Ability to use Expression Blend
- No duplicated code to update views  
No "myLabel.Text = newValue" sprinkled in code behind everywhere.
- Testability:  
Since your logic is completely agnostic of your view (no "myLabel.Text" references), unit testing is made easy.

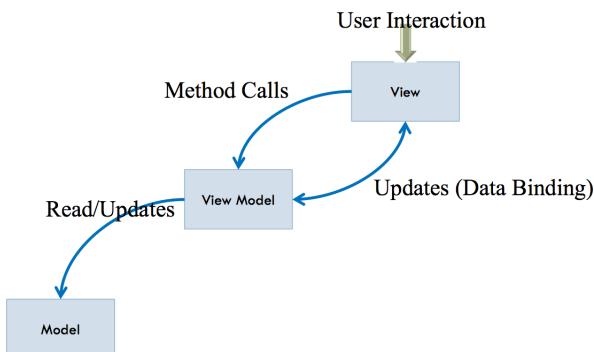
Each View binds to a single ViewModel that provides all functionality and data

- Makes the ViewModel unit-testable
- Simplifies the data binding

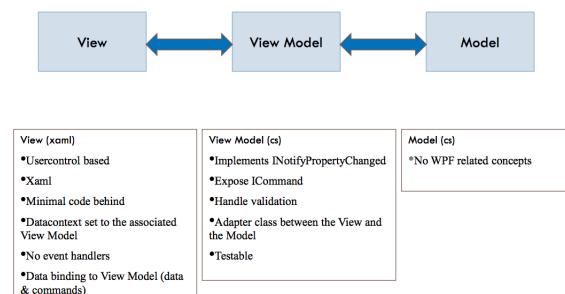
#### 4.1.1 Pattern



#### 4.1.2 User Interaction



#### 4.1.3 Overview



## 4.2 Commands

Commands are used to bind UI actions to the ViewModel functionality. A Command implements the following 3 functions of the ICommand Interface:

- Execute(object param);
- CanExecute(object param);
- event CanExecuteChanged;

Commands can be used on different UI Elements (Button, Menu, KeyboardShortcut).

To use commands in the ViewModel you have to implement your own generic command class:

Listing 14: Commands:

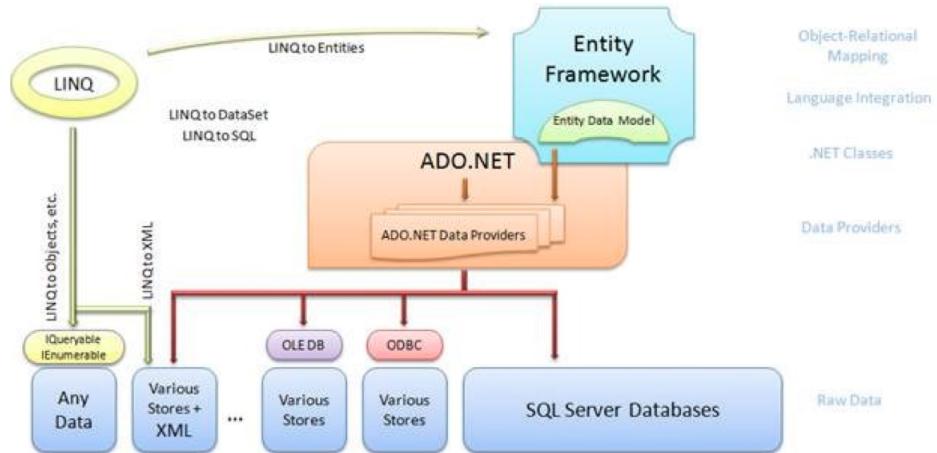
```
1 public class FooCommand : ICommand {  
2     public Action<object> ExecuteDelegate { get; set; }  
3     public Func<object, bool> CanExecuteDelegate { get; set; }  
4  
5     #region ICommand Members  
6     public bool CanExecute(object parameter) {  
7         return CanExecuteDelegate(parameter);  
8     }  
9  
10    public event EventHandler CanExecuteChanged;  
11  
12    public void Execute(object parameter) {  
13        ExecuteDelegate(parameter);  
14    }  
15    #endregion  
16 }
```

---

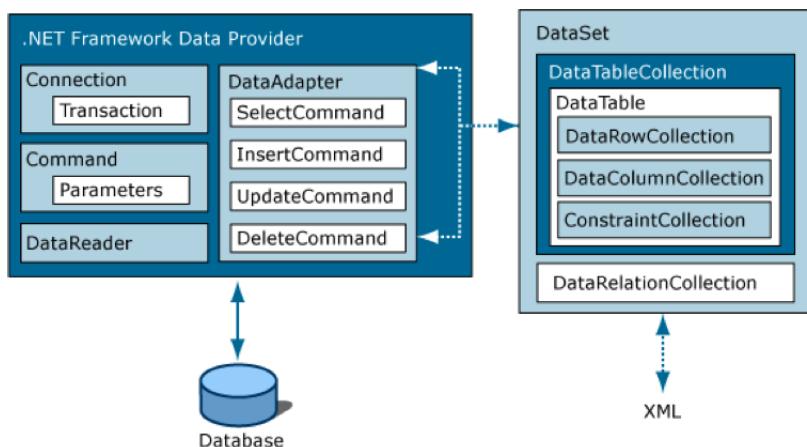
## 5 Entity Framework

### 5.1 ADO.NET

#### 5.1.1 Architecture



#### 5.1.2 Data Provider



### 5.2 Considerations

- Low level, full control
- Connected or disconnected
- You are responsible for entity mapping
- Flexible, high performance

→ ADO.NET is the basis for other data access technologies and is going to stay

#### 5.2.1 Sample

Listing 15: Sample:

```
using (SqlConnection conn = new SqlConnection("<connection string>")) {  
    conn.Open();
```

---

```

5    SqlCommand cmd = new SqlCommand("select * from products", conn);
4    using (SqlDataReader rd = cmd.ExecuteReader()) {
3      while (rd.Read()) {
2        Console.WriteLine(rd["ProductName"]);
1      }
9  }

```

---

Listing 16: DataSet Sample:

```

1 using (SqlConnection cn = new SqlConnection("<connection string>") {
2   cn.Open();
3   OleDbDataAdapter ad = new OleDbDataAdapter("select * from products", cn);
4   DataSet ds = new DataSet();
5   ad.Fill(ds, "Products");
6   foreach(DataRow dr in ds.Tables["Products"].Rows) {
7     Console.WriteLine(dr["ProductName"]);
8   }
9 }

```

---

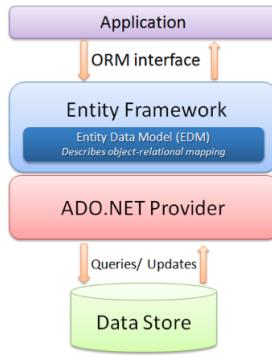
### 5.2.2 the base for ORMs

- ADO.NET Entity Framework (EF)
  - Maps conceptual model to physical tables
  - LINQ
- LINQ to SQL
  - Lightweight, no mapping
  - Replaced by EF but still supported by MS
- NHibernate
  - Open Source
  - LINQ
- 3rd Party O/RMs
  - Telerik OpenAccess & others

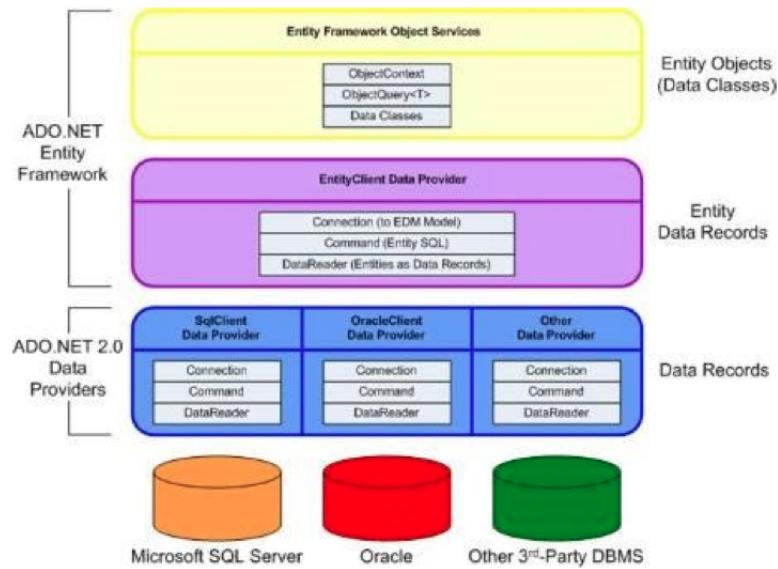
## 5.3 Entity Framework

### 5.3.1 Overview

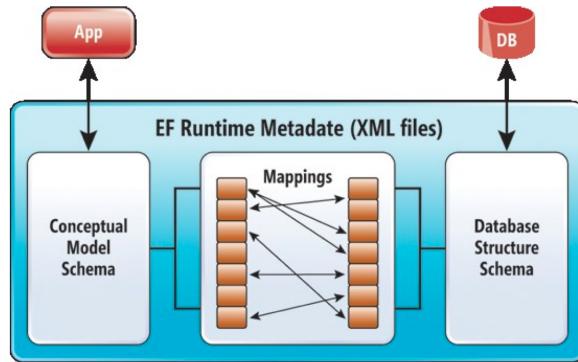
- Object / Relational Mapping
- LINQ as query language
- Features like: change tracking, identity resolution, lazy loading, etc.



### 5.3.2 Architecture



### 5.3.3 Entity Data Model



```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="2.0" xmlns:edmx="http://schemas.microsoft.com/ado/2008/10/edmx">
    <!-- EF Runtime content -->
    <edmx:Runtime>
        <!-- SSDL content -->
        <edmx:StorageModels>...</edmx:StorageModels>
        <!-- CSDL content -->
        <edmx:ConceptualModels>...</edmx:ConceptualModels>
        <!-- C-S mapping content -->
        <edmx:Mappings>...</edmx:Mappings>
    </edmx:Runtime>
    <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
    <Designer xmlns="http://schemas.">...</Designer>
</edmx:Edmx>
```

### 5.3.4 Benefits & Considerations

- Decouples logical model from database model
- Consistent, independent query language
- Rich designer support
- Most common patterns (1-to-many, many-to-1, many-to-many, self-referencing, inheritance,
- Slower than ADO.NET core
- Must regenerate EDM after DB changes

## 6 Dependency Injection

### 6.1 DI and IoC

#### 6.1.1 Overview

- Higher level modules should not depend on lower level modules
- Both should depend on abstractions (Interfaces or Abstract classes)
- Abstractions should not depend on details

#### 6.1.2 Implications

- Layers / Modularization
- Interface based programming
- Separated Interface (put interface in separate package than implementation)
- Implementation through Dependency Injection, for example

#### 6.1.3 Advantages of IoC

- Increase loose coupling
  - Abstract interfaces don't change
  - Concrete classes implement interfaces
  - Concrete classes easy to throw away and replace
- Increase mobility
- Increase isolation
  - decrease rigidity
  - Increase testability
  - Increase maintainability

## 6.2 The DI Container Unity

### 6.2.1 Options and Types

Dependency Injection Options:

- Factories
- Locator/Registry/Directory
- DI Containers (aka IoC Containers)

Configuration Options:

- Code Based
- XML Files
- Autowiring

### **6.2.2 Options**

Option 1 – Factory:

- User depends on factory
- Factory depends on destination

Option 2 – Locator/Registry/Directory

- The component still controls the wiring
- Instantiation Sequence
- Dependency on the Locator

Option 3 – Dependency Injection

- An assembler controls the wiring

### **6.2.3 Pros & Cons**

Pros:

- Loosely Coupled
- Increases Testability (ALOT!)
- Separates components cleanly
- Allows for use of Inversion of Control Container

Cons:

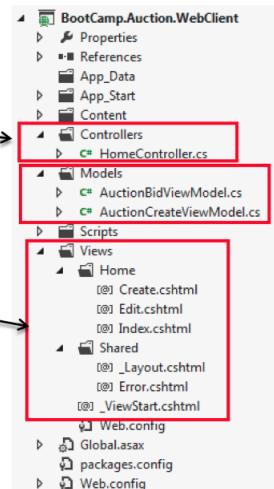
- Increases code complexity
- Some Jr. Developers find it difficult to understand at first
- Can Complicate Debugging at First
- Complicates following Code Flow

### **6.2.4 Services of IoC Container**

- Service locator
- Managing lifetime of depended-on objects
- Automatic injection of dependencies
- Configuration

## 7 ASP.NET MVC

- **Controllers:**
  - Suffix “Controller”
  - Controllers Folder
- **Views:**
  - Views folder
  - Subfolder with controller name
  - Each view maps to an action name in the controller
  - Folder “Shared” for common views (templates, user controls)



### 7.1 Controllers

- Controllers handle all incoming requests
- Retrieve data from storage
- Store posted data in storage (http post)
- Pass the data to a View to generate the HTML/CSS/JavaScript for display
- Controllers can implement REST APIs (derive controller from ApiController)

#### 7.1.1 Actions

The simplest controller just returns HTML to the Web browser

IMPORTANT: By default, all public methods in a controller class can be called from the Web browser

Controllers can easily be tested (without ASP.NET runtime)

Most controller methods return a type of action results:

Return Type	Content	Example
ViewResult	HTML-Page	return View();
PartialViewResult	Part of a HTML page	return PartialView();
RedirectResult	Redirection to other page	return Redirect("http://sbb.ch")
FileResult	Binary data	return File("images\myimage.png");
JsonResult	Serialized JavaScript objects	return Json(auction);
JavaScriptResult	JavaScript Code (Ajax)	return JavaScript("...")

#### 7.1.2 Ajax

Ajax calls can also be handled by the controller

Listing 17: Ajax:

```

1 // View
@Ajax.ActionLink("Click me", "Click", null)

// Controller
public JavaScriptResult Click() {
6   return JavaScript("alert('Hello World');");
}

```

---

### 7.1.3 View typed

Typed way to pass data to the View:

Listing 18: typed view:

```

// Controller
public ActionResult Index() {
3   return View(db.Item.ToList());
}

// View
@model IEnumerable<Auction.Models.Item> Html.DisplayNameFor(model => model.StartPrice)

```

---

### 7.1.4 View untyped

The ViewBag is used to push untyped data to the view (Properties are created at runtime...)

Listing 19: typed view:

```

// Controller
ViewBag.Persons = new SelectList(context.People, "Id", "Name");
3

// View
@Html.DropDownListFor(model => model.SellerId, (SelectList)ViewBag.Persons)

```

---

## 7.2 Views

### 7.2.1 Code Blocks



## 7.2.2 HtmlHelpers

Listing 20: typed view:

```
// <a href="/Home/Edit/3">Edit Record</a>
@Html.ActionLink("Edit Record", "Edit", new {Id=3});

// <label for="FirstName">FirstName</label>
5 @Html.LabelFor(model => model.FirstName);

// <input class="text-box single-line" data-val="true" data-val-required="The FirstName
   field is required." id="FirstName" name="FirstName" type="text" value="" />
@Html.EditorFor(model => model.FirstName);
```

## 7.2.3 Layout Pages

```
_Layout.cshtml
<html>
<title>@ViewBag.Title</title>
<body>
    @RenderBody()
</body>
</html>

Index.cshtml
@{
    ViewBag.Title = "Your Page Title";
}
<div>Hello World!</div>
```

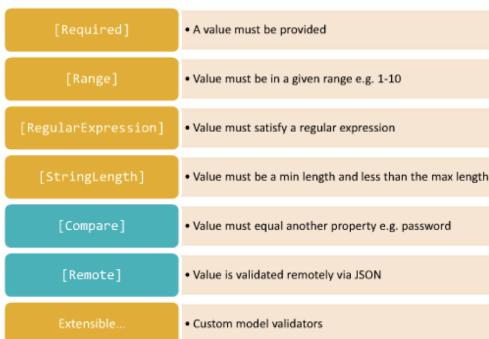
## 7.3 Routes

Listing 21: routes

```
routes.MapRoute(
2    "Default", // Route name
    "{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Default
);
```

## 7.4 Model

Validations can be specified for individual fields in the data model. Client and Server validation with no additional coding.



# 8 Codebeispiele

## 8.1 WPF XAML

Listing 22: MainWindow.xaml

```

<Window x:Class="Fhnw.Dnead.Auction.WpfClient.Views.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="596.317"
5     xmlns:my="clr-namespace:Fhnw.Dnead.Auction.WpfClient"
    xmlns:conv="clr-namespace:Fhnw.Dnead.Auction.WpfClient.Converter">

    <Window.Resources>
        <conv:BooleanToStatusTextConverter x:Key="booleanToStatusTextConverter" />
10       <BooleanToVisibilityConverter x:Key="booleanToVisibilityConverter" />
    </Window.Resources>

    <Grid>
        <DataGrid AutoGenerateColumns="False" Name="dgAuctionItems" ItemsSource="{Binding
            Path=Auctions}" Margin="0,48,0,0" IsReadOnly="True">
15           <DataGrid.Columns>
                <DataGridTemplateColumn>
                    <DataGridTemplateColumn.CellTemplate>
                        <DataTemplate>
                            <Button Background="Red" Name="btnBuy" Content="Buy" Width="44"
                                Margin="2" Visibility="{Binding Path=Open, Converter={
                                    StaticResource booleanToVisibilityConverter}}}" IsCancel="
                                False" Click="BtnBuyClick"></Button>
20
                        </DataTemplate>
                    </DataGridTemplateColumn.CellTemplate>
                </DataGridTemplateColumn>
                <DataGridTextColumn Binding="{Binding Path=Name}" Header="Artikel" />
                <DataGridTextColumn Binding="{Binding Path=StartPrice}" Header="Start Preis
                    " />
25           <DataGridTextColumn Binding="{Binding Path=Bid}" Header="Aktuelles Gebot"
                    />
                <DataGridTextColumn Binding="{Binding Path=StartTime}" Header="Auktionsstart" />
                <DataGridTextColumn Binding="{Binding Path=EndTime}" Header="Auktionsende"
                    />
                <DataGridTextColumn Binding="{Binding Path=Seller}" Header="Verkäufer"
                    />
                <DataGridTextColumn Binding="{Binding Path=Buyer}" Header="Käufer" />
30           <DataGridTextColumn Binding="{Binding Path=Open, Converter={StaticResource
                            booleanToStatusTextConverter}}}" Header="Status" />
                <DataGridTemplateColumn Header="Bild">
                    <DataGridTemplateColumn.CellTemplate>
                        <DataTemplate>
                            <Image Source="{Binding Path=Picture}" Width="64" Height="64"
                                />
                        </DataTemplate>
                    </DataGridTemplateColumn.CellTemplate>
                </DataGridTemplateColumn>
35
            </DataGrid.Columns>
        </DataGrid>
        <Button Content="Artikel verkaufen" HorizontalAlignment="Left" Margin="23,10,0,0"
            VerticalAlignment="Top" Width="162" Click="BtnSellClick"/>
    </Grid>
</Window>
```

Listing 23: BidView.xaml

```

1  <Window x:Class="Fhnw.Dnead.Auction.WpfClient.Views.BidView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Bieten" Height="176" Width="300">

6    <DockPanel>
        <StackPanel Orientation="Horizontal" DockPanel.Dock="Bottom" HorizontalAlignment="
            Right">
            <Button Content="Cancel" Height="23" HorizontalAlignment="Left" Margin="10"
                Name="btnCancel" VerticalAlignment="Bottom" Width="75" IsCancel="True" />

```

```

    <Button Content="OK" Height="23" HorizontalAlignment="Left" Margin="10" Name="
        btnOk" VerticalAlignment="Bottom" Width="75" Command="{Binding OkCommand}"
        CommandParameter="{Binding RelativeSource={RelativeSource FindAncestor,
        AncestorType={x:Type Window}}}" />
    </StackPanel>
11   <StackPanel Orientation="Horizontal" HorizontalAlignment="Right" DockPanel.Dock="
        Bottom">
        <TextBlock Height="23" HorizontalAlignment="Right" Margin="10" Text="Gebot"
            VerticalAlignment="Bottom" />
        <TextBox Height="24" Margin="10" HorizontalAlignment="Stretch" Width="100" Name
            ="txtBid" VerticalAlignment="Bottom" Text="{Binding Path=Bid}" />
    </StackPanel>
    <Viewbox Margin="10" >
        <Image Name="imgPicture" Source="{Binding Path=Item.Picture}">
            </Image>
    </Viewbox>
</DockPanel>
21 </Window>

```

---

Listing 24: MainViewModel.cs

```

using System;
using System.Collections.ObjectModel;
using System.Linq;
4 using System.Windows.Threading;
using Fhnw.Dnead.Auction.DAL;

namespace Fhnw.Dnead.Auction.WpfClient.ViewModels
{
9     public class MainViewModel : ViewModelBase
    {
        readonly ObservableCollection<AuctionViewModel> auctions = new ObservableCollection
            <AuctionViewModel>();

        public ObservableCollection<AuctionViewModel> Auctions
14       {
            get
            {
                return auctions;
            }
        }

19       }

        readonly AuctionContext context = new AuctionContext();

        /// <summary>
24        /// Timer to detect auction closes
        /// </summary>
        readonly DispatcherTimer auctionTimer = new DispatcherTimer();

        private void StartAuctionTimer()
        {
            auctionTimer.Tick +=
                delegate
                {
34                    // set all timed out auctions to closed
                    UpdateAuctionOpenStatus();
                    context.SaveChanges();
                };
        }

39        auctionTimer.Interval = new TimeSpan(0, 0, 1);
        auctionTimer.Start();
    }

        /// <summary>
44        /// Verifies expiration date of auctions and updates open status if expired.
        /// </summary>
        private void UpdateAuctionOpenStatus()
    {

```

```

        auctions.Where(a => a.EndTime < DateTime.Now).ToList().ForEach(a => a.Open =
            false);
    }

    public MainViewModel()
    {
        // read existing data from db and fill them in the observable collection
        context.Auctions.ToList().ForEach(a => auctions.Add(new AuctionViewModel {
            Auction = a }));
        StartAuctionTimer();
    }

    public void AddNewAuction(AuctionViewModel newAuctionEntity)
    {
        context.Auctions.Add(newAuctionEntity.Auction);
        context.SaveChanges();
        Auctions.Add(newAuctionEntity)
    }
}

```

---

Listing 25: BidViewModel.cs

```

1 using System;
2 using System.Windows;
3 using System.Windows.Input;
4 using Fhnw.Dnead.Auction.WpfClient.Commands;

5
6 namespace Fhnw.Dnead.Auction.WpfClient.ViewModels
{
    /// <summary>
    /// ViewModel for a bid for an auction.
    /// </summary>
11    class BidViewModel : ViewModelBase
    {
        private DelegateCommand<Window> okCommand;

16        public string Bid { get; private set; }
        private readonly AuctionViewModel item;

21        public BidViewModel(AuctionViewModel item)
        {
            this.item = item;
            Bid = ((item.Bid == 0 || item.Bid == null? item.StartPrice : item.Bid) + 1).
                ToString();
        }

26        public bool ValidateEntries()
        {
            int newBid;
            if (!int.TryParse(Bid, out newBid)) return false;
            if (newBid == 0) return false;
            if (newBid <= item.Bid) return false;
            if (newBid <= item.StartPrice) return false;

31            item.Bid = newBid;
            item.Buyer = Environment.UserName;

36            return true;
        }

41        public ICommand OkCommand
        {
            get
            {
                if (okCommand == null)
                {
                    okCommand = new DelegateCommand<Window>(DoOk, CanDoOk);
                }
                return okCommand;
            }
        }
}

```

```
        }
    }

51     void DoOk(Window win)
{
52     if (ValidateEntries())
53     {
54         win.DialogResult = true;
55     }
56 }

57     bool CanDoOk(Window win) { return true; }
58 }

61 }
```

---