

WPF & Silverlight

Daniel Beck

Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart

29. Februar 2008

Inhaltsverzeichnis

1	Windows Presentation Foundation	4
1.1	Einführung	4
1.1.1	Entstehung	4
1.1.2	Ziele	4
1.2	Grundlagen der Windows Presentation Foundation	6
1.2.1	Attached Properties und Attached Events	7
1.3	Vektor-Orientierung	8
1.3.1	Rastergrafik	9
1.4	Deklarative Programmierung	9
1.4.1	Allgemeines	9
1.4.2	Namensräume	9
1.4.3	Property Element Syntax	10
1.4.4	Content Properties	10
1.4.5	Markup Extensions	11
1.4.6	Code-behind	12
1.5	Elementkomposition	13
1.5.1	Ressourcen	13
1.5.2	Control Templates	13
1.6	Datenbindung	14
1.6.1	Herstellen einer Verbindung	14
1.6.2	DataContext	15
1.6.3	Value Converter	15
1.6.4	Data Templates	16
1.7	Bedienelemente und Layouts	17
1.7.1	Layouts	17
1.8	Styles	18
1.9	Themes	19
1.10	Weitere Funktionen	20
1.10.1	Dokumente und Text	20
1.10.2	Medien und 3D	21
1.11	Entwicklungswerkzeuge	21
1.12	Deployment	21
1.12.1	Stand-Alone	21
1.12.2	XBAP	22
1.12.3	ClickOnce	22

2	Silverlight	23
2.1	Version 1.0	23
2.1.1	Browser-Integration	23
2.1.2	Audio- und Video-Funktionalität	23
2.1.3	Programmierung	23
2.2	Weitere Entwicklung	24
2.2.1	Angekündigte Funktionen	24
2.3	Werkzeuge	25
2.4	Vergleich zu Flash	25
2.4.1	Vorteile von Silverlight	25
2.4.2	Vorteile von Flash	26
3	Vergleich von WPF und Silverlight	27
3.1	Kontext	27
3.2	WPF-Applikation	27
3.3	XAML Browser Application (XBAP)	27
3.4	Silverlight 1.0	27
3.5	Silverlight 2.0	28
3.6	Fazit und Empfehlung	28
4	Appendix	29

Abbildungsverzeichnis

1	Architektur von WPF	6
2	Anzeige bei Aufruf von <i>Listing 1</i>	7
3	Verweis-Semantik von Markup Extensions	12
4	Anzeige einer Collection ohne passendes DataTemplate	16
5	Anzeige einer Collection mit passendem DataTemplate	17

Listingverzeichnis

1	Deklaration eines Layouts mit <i>Attached Properties</i> in XAML	8
2	Deklaration eines Buttons ohne <i>Property Element Syntax</i>	10
3	Deklaration eines Buttons mit <i>Property Element Syntax</i>	10
4	Deklaration eines Buttons unter Verwendung der <i>Content Property</i>	11
5	Einbinden von Ressourcen mit Hilfe von <i>Markup Extensions</i>	11
6	Partielle Klassendefinition in XAML	12
7	Partielle Klassendefinition im <i>code behind</i>	13
8	<i>Value Converter</i> zur Umwandlung von Celsius in Fahrenheit	16
9	DataTemplate für eine Collection von Movie-Objekten	17
10	Beispiel für <i>Styles</i> mit <i>Trigger</i>	19

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Windows Presentation Foundation (WPF), der in Microsoft [.NET](#) ab Version 3.0 enthaltenen, neuen Bibliothek für grafische Bedienoberflächen ([GUI](#)). Darauf aufbauend wird im Anschluss Microsoft Silverlight, eine Laufzeitumgebung zur Darstellung von Medieninhalten im Web-Browser, die auf der WPF-Technologie aufbaut, vorgestellt. Zuletzt werden die verschiedenen Möglichkeiten, Software basierend auf WPF oder Silverlight zu entwickeln, verglichen.

1 Windows Presentation Foundation

Dieser Teil der Arbeit befasst sich mit der Windows Presentation Foundation. Der Leser sei darauf hingewiesen, dass der Umfang der WPF es nicht zulässt, jeden Teilbereich davon angemessen zu behandeln und auf das Literaturverzeichnis im Anhang verwiesen.

1.1 Einführung

1.1.1 Entstehung

Die Erstellung grafischer Benutzeroberflächen in den ersten Versionen von Windows war kompliziert: Unter direkter Verwendung von Betriebssystem-Funktionen der Windows-APIs *user* und *gdi* wurden Fenster und Bedienelemente erzeugt. Anfang der neunziger Jahre wurde die Handhabung dieser Bibliotheken mit der Bibliothek *Ruby* für *Visual Basic 1.0* vereinfacht.

Wenige Jahre später wurde eine weitere Bibliothek für grafische Benutzeroberflächen von Microsoft unter der Bezeichnung *Trident* entwickelt: Die später als *MSHTML* bekannte [HTML](#)-Rendering-Engine des Microsoft Internet Explorers (ab Version 4.0) wurde 1997 veröffentlicht.

Rund zehn Jahre nach der Veröffentlichung von Ruby wurde im Rahmen der ersten .NET-Framework-Versionen eine neue Technologie entwickelt: Windows Forms. Doch auch Windows Forms verwendet nur die Windows APIs und ist keine Neuentwicklung. Die Windows-API wurde zwar im Laufe der Jahre erweitert, an der grundlegenden Architektur wurde jedoch nichts verändert. So arbeiten selbst modernste Windows-Forms-Oberflächen mit Clipping-basierter Elementdarstellung, was es unmöglich macht, Elemente halbtransparent oder überlappend darzustellen.

Entschlossen, diese sich großteils überlappenden Technologien zu vereinen, wurde unter dem Codenamen *Avalon* eine neue Bibliothek für grafische Oberflächen von Microsoft entwickelt. Veröffentlicht wurde sie unter dem Namen *Windows Presentation Foundation* als Teil von .NET 3.0 im Jahr 2006.

1.1.2 Ziele

Mit der Entwicklung der WPF wurden mehrere Ziele verbunden: WPF sollte nicht nur die vorhergehenden Technologien ersetzen, sondern auch die Erstellung moderner, multime-

dialer Benutzeroberflächen ermöglichen.¹

- **Auflösungsunabhängigkeit**

Mit der Entwicklung immer höher auflösenderer Displays wurde ein fundamentaler Mangel bisheriger grafischer Oberflächen offensichtlich: Bedienelemente sind, in Pixeln gemessen, auf allen Ausgabegeräten gleich groß. Während ein normaler Button mit 20 Pixeln Höhe bei einem Display mit 72 ppi (Pixel pro Zoll) noch rund 7 mm hoch ist, ist er bei 130 ppi lediglich nur noch gut halb so groß, was die Bedienbarkeit erschweren kann.

Aus diesem Grund wurde, auch in Erwartung bald kommender hochauflösender Displays mit noch wesentlich mehr Pixeln pro Zoll, großer Wert auf Auflösungsunabhängigkeit der Grafikbibliothek gelegt.

- **Erstellung der Oberfläche durch Designer**

Der herkömmliche Ansatz bei der Erstellung grafischer Benutzeroberflächen ist, dass Designer die Oberflächen in einem Grafikprogramm entwerfen und diese Bilder dann von Entwicklern in grafische Benutzeroberflächen umgesetzt werden. Diese Arbeitsweise stößt jedoch schnell an ihre Grenzen, wenn die Designer nicht mit den Eigenschaften der von den Entwicklern verwendeten Technologie vertraut sind. Zudem wird ein Teil der Arbeit hierbei unnötigerweise zweimal erledigt. WPF sollte es besonders einfach machen, die Oberfläche eines Programms komplett von Designern erstellen zu lassen, während die Funktionalität der Software von Entwicklern umgesetzt wird. Da Designer häufig nicht mit herkömmlichen, imperativen Programmiersprachen vertraut sind, sollten sie bei der Erstellung grafischer Bedienoberflächen auch nicht damit in Berührung kommen.

- **Trennung von Darstellung und Verhalten**

Eines der wichtigsten Architekturprinzipien in der Programmierung interaktiver Anwendungen dreht sich um die Trennung von Darstellung und Verhalten. Im Architekturmuster “Model-View-Controller” ist die Anzeige eines Bedienelements getrennt von dessen Verarbeitung von Benutzeraktionen. Dieses Prinzip sollte sich in der Struktur von Bedienelementen in WPF niederschlagen.

- **Gemeinsame Plattform für Bedienoberfläche, Dokumente und Medien**

Windows Presentation Foundation sollte sowohl dazu dienen, Benutzeroberflächen zu erstellen, als auch die Darstellung von Medieninhalten und Dokumenten übernehmen. Normalerweise stellt eine GUI-Bibliothek die Bedienelemente dar, und durch eine weitere Bibliothek wird, in einem eigenen Bereich, beispielsweise ein Video angezeigt. Diese Trennung sollte in WPF aufgehoben werden.

Dies ist einer der Gründe, weshalb WPF auf [DirectX](#) von Microsoft basiert. Es ermöglicht die performante Darstellung von Videos und 3D-Szenen.

¹Nach [Anderson 2007]

- **Eigenschaften von Web und Desktop vereinen**

Das Web bietet viele Vorteile gegenüber herkömmlichen Desktop-Programmen, beispielsweise einfaches Deployment von Webanwendungen und die Verwendung von Markup-Sprachen wie HTML zur Erzeugung der Benutzeroberfläche. Diese Vorteile sollten in WPF Verwendung finden, in Kombination mit den bekannten Vorteilen von Desktop-Anwendungen, wie beispielsweise eine gute Integration in die Benutzerumgebung (wie Desktop-Icon, assoziierte Dateitypen) und Kontrolle über das Look & Feel.

Zudem sollte die Funktionsweise von Webseiten, mit Navigation und den Funktionen *Vor* und *Zurück* in WPF speziell unterstützt werden, um sogenannte *Browser-Applications* zu ermöglichen. Diese verhalten sich wie Webseiten und sind sowohl im Web-Browser, als auch als eigenständiges Programm ausführbar. Bekannte Beispiele für diese Funktionsweise sind Installationsprogramme und Assistenten (*Wizards*), die den Anwender Schritt für Schritt durch die Ausführung einer komplexen Aufgabe führen.

1.2 Grundlagen der Windows Presentation Foundation

Ein Großteil von WPF wird in der .NET-Laufzeitumgebung ausgeführt. Nur eine Komponente, der “Media Integration Layer” (*milcore*), ist als nativer Code implementiert. Dieser kümmert sich um die Kommunikation mit DirectX, um eine hohe Geschwindigkeit der Darstellung zu erreichen.

In *Abbildung 1*² ist die Architektur der WPF und deren Integration in das System veranschaulicht. Die roten Bereiche stellen die WPF dar, welches Dienste der blau dargestellten Technologien nutzt.

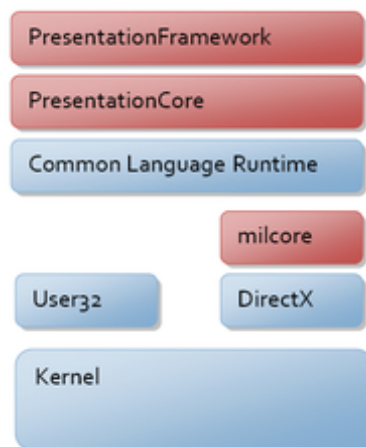


Abbildung 1: WPF besteht aus den roten Teilen und nutzt Dienste der blauen Systeme.

²Bildquelle: <http://msdn2.microsoft.com/en-us/library/ms750441.aspx>

Die meisten Klassen in WPF sind von `DispatcherObject` abgeleitet. Diese Klasse stellt Messaging-Funktionen für die Kommunikation zwischen mehreren Threads mittels Dispatcher zur Verfügung.

Von `DispatcherObject` erbt `DependencyObject`. Diese Klasse umfasst ein erweitertes Eigenschaftensystem, basierend auf `DependencyProperty`, das über das der [CLR](#) hinaus geht. So erlauben es `DependencyObjects`, Beobachter mittels `Events` über Veränderungen von Attributwerten zu informieren³. Ebenso werden Änderungen bei weiter oben im Elementbaum angeordneten Containern an die enthaltenen Elemente propagiert (beispielsweise die Vergrößerung durch `ScaleTransform`), sofern diese nicht andere Werte zugeordnet bekommen haben. Zudem erlauben erst `DependencyObjects` die sogenannten *Attached Properties*.

1.2.1 Attached Properties und Attached Events

WPF erlaubt die Angabe bestimmter Attribute und Events auf jedem beliebigen Element. Diese werden als *Attached Properties* bzw. *Attached Events* bezeichnet. Dieser Weg wurde gewählt, da die Basisklassen (wie `UIElement`) nicht mit für deren Funktionalität irrelevanten Eigenschaften überladen werden sollten und das System prinzipiell durch Entwickler erweiterbar sein sollte.⁴ Ein Beispiel hierfür sind Layouts⁵ (siehe *Abbildung 2*: Layouts benötigen oft spezielle Informationen über die enthaltenen Elemente, wie in *Listing 1* deren gewünschte Position innerhalb des Panels⁶. Es ist aber nicht möglich, `UIElement` um spezielle Informationen für jedes Layout anreichern. So definiert der Layout-Container `DockPanel` eine Attached Property `Dock`, die angibt, an welcher Seite des `DockPanel` das jeweilige Element platziert wird. Hierbei wird `DockPanel` als *Property Owner* bezeichnet.

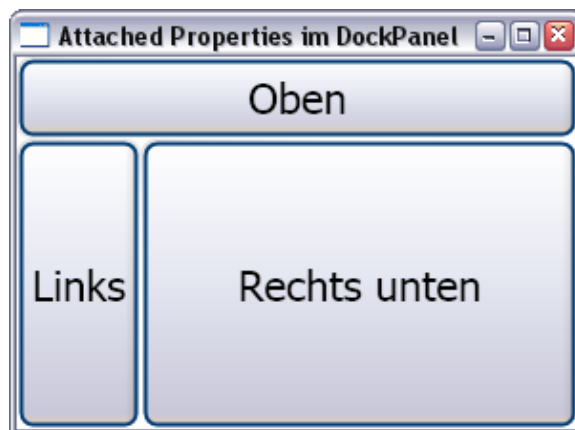


Abbildung 2: Anzeige bei Aufruf von *Listing 1*

³Dies wird unter anderem verwendet, um den Aktualisierungsmechanismus der Datenbindung in WPF (siehe *Kapitel 1.6 auf Seite 14*) zu realisieren.

⁴Siehe den zweiten Teil von [\[Smith 2007\]](#)

⁵Layouts werden ausführlich in [1.7.1](#) vorgestellt

⁶Hier wird die Oberfläche in [XML](#) deklariert. Dies wird in [1.4](#) ausführlich vorgestellt

```

<DockPanel>
  <Button DockPanel.Dock="Top" ... />
  Oben
</Button>
<Button DockPanel.Dock="Left" ... />
  Links
</Button>
<Button ... />
  Rechts unten
</Button>
</DockPanel>

```

Listing 1: Deklaration eines Layouts mit *Attached Properties* in XAML

Attached Events funktionieren auf ähnliche Weise. So können Container-Elemente wie das `DockPanel` in *Listing 1* ein Attribut `Button.Click` erhalten, und somit für alle enthaltenen Buttons einen gemeinsamen Event-Handler bestimmen. Events wie `MouseDown` sind intern als *Attached Events* realisiert. `UIElement`, eine der Oberklassen der Anzeige-Elemente, erstellt Aliase auf diese *Attached Events*, die nicht die *Attached-Event-Syntax* verwenden, somit ist keine *Attached-Event-Syntax* zur Verwendung notwendig. *Attached Events* sind primär ein Element der zukunftsfähigen Architektur von WPF und müssen nur selten explizit von Entwicklern verwendet werden.

1.3 Vektor-Orientierung

Frühere GUI-Technologien basierten auf Pixelangaben, die den Pixeln der Bildschirme entsprachen: Eine Linie mit einer Länge von 10 Pixeln wurde auf dem Ausgabegerät mit genau 10 Pixeln Länge dargestellt. Auf vielen modernen Displays sind die Pixel jedoch kleiner und dichter gepackt, so dass sich auf einem Zoll des Displays meist mehr Pixel befinden, als erwartet wird. Dies führt zu einer (manchmal deutlich) schlechteren Lesbarkeit.

Um eine auflösungsunabhängige Oberfläche erzeugen zu können, basiert WPF auf Vektorgrafik. Vektorgrafiken bestehen, im Gegensatz zu herkömmlicher Rastergrafik, aus sogenannten Grafikprimitiven: Beispiele hierfür sind Linien, Kurven und Polygone. Diese Primitive sind mathematisch durch ihre Eigenschaften definiert – ein Kreis beispielsweise über seinen Mittelpunkt und Radius –, im Gegensatz zu Rastergrafik, die eher einer abschnittswisen Definition entspricht (mit einer Schrittweite von einem Pixel). Dies führt dazu, dass in Vektorgrafiken ohne Qualitätsverlust skaliert werden kann.

Größen und Positionen in WPF werden in logischen Pixeln angegeben, anstatt wie in herkömmlichen GUI-Bibliotheken in physischen Pixeln. So steht ein logischer Pixel auf heutigen Systemen für eine Größe von 1/96 Zoll, was in etwa der Pixel-Dichte der meisten modernen Bildschirme entspricht, doch kann problemlos gezoomt werden, um die Darstellung auf hochauflösenden Displays anzupassen.

Da logische Pixel nicht notwendigerweise 1:1 auf physische Pixel abgebildet sind, können diese auch als Gleitkommawerte angegeben werden. Diese werden entsprechend gerundet.

Damit Bedienelemente, bei denen es nicht auf Pixelgenauigkeit ankommt, mit hoher Schärfe dargestellt werden können, sind insbesondere Linien standardmäßig genau auf Hardware-Pixel abgebildet, so dass keine Unschärfe-Effekte auftreten.

1.3.1 Rastergrafik

Eingebundene Rastergrafiken, wie Hintergrundbilder oder Icons, werden weiterhin als Rastergrafik dargestellt. Wird die Anzeige skaliert, so treten bei diesen Elementen Qualitätsverluste auf. Diese können vermieden werden, indem beispielsweise Icons in mehreren, insbesondere hohen, Auflösungen vorliegen. Ein 512x512 Pixel großes Icon, welches mit 64 Pixeln Kantenlänge dargestellt wird, bietet Spielraum hinsichtlich der verlustfreien Skalierung.

1.4 Deklarative Programmierung

Windows Presentation Foundation erlaubt die Definition der Benutzeroberfläche durch XML. Hierzu hat Microsoft XAML entwickelt: XAML steht für *Extensible Application Markup Language* und ermöglicht die Programmierung großer Teile von WPF-Oberflächen. An dieser Stelle wird vorausgesetzt, dass der Leser über Grundkenntnisse von XML verfügt.

1.4.1 Allgemeines

Nahezu alles in WPF kann in XAML, oder wahlweise auch in prozeduralem Code, definiert werden. Bevorzugt ist jedoch die Deklaration der Benutzeroberfläche in XAML, um die Trennung vom Verhalten des Programms zu erreichen.

Wird ein Element in XAML deklariert, so wird eine Instanz der Klasse in der CLR erzeugt. Hierzu müssen die Klassen er zu erstellenden Elemente mehrere Voraussetzungen erfüllen. Beispielsweise müssen sie über einen Standardkonstruktor verfügen, da bei der Deklaration in XML keine Konstruktor-Parameter angegeben werden können.

1.4.2 Namensräume

Zwei XML-Namensräume sind für die Verwendung von XAML in WPF besonders wichtig:

- `http://schemas.microsoft.com/winfx/2006/xaml/presentation`
Der WPF-Namensraum definiert alle Elemente der Anzeige, wie Buttons, Fenster und Vektorprimitive. Es ist Konvention, diesen Namensraum ohne Präfix zu verwenden.
- `http://schemas.microsoft.com/winfx/2006/xaml`
Der XAML-Namensraum enthält die von der Anzeige unabhängigen Elemente, welche in XAML deklariert werden können. Dies umfasst Programmlogik-bezogene Elemente, wie zum Beispiel Arrays, statische und dynamische Ressourcen, sowie Bindings. Es ist Konvention, diesen Namensraum mit dem Präfix `x` zu versehen⁷.

⁷Siehe <http://msdn2.microsoft.com/en-us/library/ms747086.aspx>

Auf selbst definierte Klassen kann zugegriffen werden, indem im XAML-Dokument die CLR-Namensräume dieser Klassen angegeben werden. Angenommen, diese Klassen liegen im Namensraum `MyApplication`, so gibt man anstatt einer URI für den Namensraum Folgendes an:

```
xmlns:local="clr-namespace:MyApplication"
```

Wenn hierbei auf eine andere [Assembly](#) verwiesen werden muss, würde die Namespace-Angabe folgendermaßen aussehen:

```
xmlns:local="clr-namespace:MyApplication;assembly=TheAppAssembly"
```

1.4.3 Property Element Syntax

Als *Property Element Syntax* wird eine alternative Angabe von Elementattributen bezeichnet. Normalerweise werden alle Eigenschaften eines Objekts innerhalb dessen XML-Element als Attribute angegeben. Dies ist jedoch insbesondere bei komplexeren Werten ungünstig: Es können keine Elementbäume als Wert angegeben werden, sondern nur Strings, die trivial in den eigentlichen Datentyp des Attributs umgewandelt werden können.

```
<Button Content="Press me!" />
```

Listing 2: Deklaration eines Buttons ohne *Property Element Syntax*

Als Alternative kann man Eigenschaften eines Elements in einem zusätzlichen Tag innerhalb des Elements angeben (*Listing 3*). Der gewünschte Wert steht hierbei zwischen dem öffnenden und schließenden Tag. Diese Attribute können auch komplexere Werte, wie Element-Bäume, annehmen.

```
<Button>
  <Button.Content>
    Press me!
  </Button.Content>
</Button>
```

Listing 3: Deklaration eines Buttons mit *Property Element Syntax*

1.4.4 Content Properties

Viele Elemente erlauben die Angabe eines Inhalts, wie beispielsweise die Beschriftung eines `Buttons` oder `TextBlocks`.⁸ Solche Inhalte sind normale Attribute der Objekte: `Content` im Fall des `Buttons`, `Text` beim `TextBlock`.

Eine einfachere Syntax zur Angabe dieser Werte erlauben die *Content Properties*. Klassen, die das Konzept eines *Inhalts* haben, ernennen in ihrer Klassendefinition ein Attribut

⁸Eine Auflistung des Attributs verschiedener Klassen bietet [\[WPF SDK 2006\]](#)

zur *Content Property*.⁹ Dieses Attribut bekommt den zwischen dem Start- und Endtag des Element angegebenen Textknoten oder Elementbaum zugewiesen. *Listing 4* zeigt die Verwendung der *Content Property* bei einer Elementdeklaration (vgl. *Listing 2*)

```
<Button>
    Press me!
</Button>
```

Listing 4: Deklaration eines Buttons unter Verwendung der *Content Property*

Normalerweise kann nur ein einzelnes Element oder ein einzelner Textknoten *Content Property* sein, beispielsweise bei der Beschriftung eines Buttons. Objekte wie Panels oder Listen, die mehrere Elemente darstellen, können auch mehrere Elemente als *Content Property* enthalten, da diese explizit eine *Collection* von Elementen enthalten.

Der XAML-Parser verwirft standardmäßig führende und angehängte *Whitespaces* bei der Verarbeitung von XAML. Dies ist insbesondere im Kontext von *Content Properties* relevant: So kann die *Content Property* mit Zeilenumbruch formatiert zwischen die Tags des umschließenden Elements geschrieben werden, ohne dass sich dies, beispielsweise durch eine mehrzeilige Button-Beschriftung, im Wert niederschlägt. Es ist jedoch zu beachten, dass der Inhalt eines Elements kontinuierlich angegeben werden muss und nicht durch *Property Elements* unterbrochen werden darf.

1.4.5 Markup Extensions

Deklariert man in XAML ein Element, so wird eine neue Instanz der jeweiligen Klasse angelegt. Da XML-Dokumente, ausgehend von einem Wurzelement, hierarchisch aufgebaut sind, kann man allein mit XML keine Querverweise auf andere Elemente anlegen. Hierfür enthält WPF das erweiterbare Konzept der *Markup Extensions*.

Markup Extensions sind in geschweiften Klammern angegebene Elemente, die auf andere Elemente im XML-Elementbaum oder Programm verweisen können. Die geschweiften Klammern geben dem XAML-Parser an, dass er das enthaltene Element nicht als String, oder einen davon durch Umwandlung erstellbaren Typ, behandeln soll.

```
<Window ...>
  <Window.Resources>
    <LinearGradientBrush x:Key="theBrush" .... />
  </Window.Resources>
  <DockPanel>
    <Button Background="{StaticResource theBrush}" ... />
    <Button Background="{StaticResource theBrush}" ... />
  </DockPanel>
</Window>
```

Listing 5: Einbinden von Ressourcen mit Hilfe von *Markup Extensions*

⁹Vgl. [Petzold 2006]

Abbildung 3 zeigt die Semantik dieses Listings. Die Buttons verweisen auf den Brush, anstatt einen neuen Brush zu erzeugen.

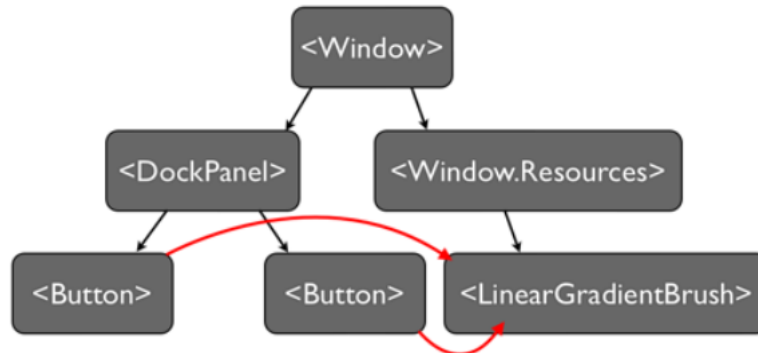


Abbildung 3: Die roten Pfeile stellen die Verbindung der Elemente über *Markup Extensions* dar.

Markup Extensions werden insbesondere in der Elementkomposition und Datenbindung verwendet, welche in folgenden Kapiteln behandelt werden.

1.4.6 Code-behind

Um eine in XAML deklarierte Oberfläche mit Funktionalität zu füllen, muss man eine sogenannte Code-behind-Datei anlegen. Diese ist der *Controller* für eine XAML-Oberfläche. In einer Code-behind-Datei wird eine *partial class* mit demselben Namen definiert (Attribut **Class** im XAML-Namensraum) wie das Wurzelement der zugehörigen XAML-Datei. Bei der Kompilierung werden die beiden Teildefinitionen vereint.

Um vom Code-behind auf die in XAML deklarierten Elemente zuzugreifen, muss man diesen Elementen bei ihrer Deklaration einen Namen geben. Hierzu wird das Attribut **Name** aus dem XAML-Namensraum verwendet. Unter dieser Bezeichnung ist der Zugriff auf diese Elemente von Code-behind aus möglich.

Benutzereingaben Um auf Benutzereingaben reagieren zu können, kann man in XAML das Event-Handling der Anzeige-Elemente steuern. Beispielsweise hat **Button** ein Attribut **Click**, das einen Event-Handler enthält. Dieser kann in XAML durch seinen Namen angegeben werden. Wenn der Button das **Click**-Ereignis behandelt, wird der so angegebene Event-Handler aufgerufen und dessen im Code-behind definierter Code ausgeführt.

```

<Window x:Class="MyApplication.MyWindow" ...>
  ...
  <Button x:Name="theButton" Content="Press Me!"
    Click="HandleButtonClick" ... />
  ...
</Window>
  
```

Listing 6: Partielle Klassendefinition in XAML

```

namespace MyApplication {
    public partial class MyWindow {
        ...
        private void HandleButtonClick(object sender, ...) {
            theButton.Content="Pressed!";
        }
        ...
    }
}

```

Listing 7: Partielle Klassendefinition im *code behind*

1.5 Elementkomposition

Um komplexere Elemente, wie beispielsweise einen Button, mithilfe von Grafikprimitiven definieren zu können, bedient sich WPF der Elementkomposition: Hierdurch können komplexe Elemente, wie Bedienelemente, aus anderen, einfacheren Elementen, wie Linien, Verläufen und Rechtecken, zusammengesetzt werden.

Das Prinzip der Elementkomposition zieht sich durch ganz WPF: So können die meisten Controls mit Inhalt (wie `TextBlock` und `Button`) verschiedenste Inhaltstypen anzeigen. Beispielsweise ist es möglich, als Button-Beschriftung weitere Buttons anzuzeigen, oder innerhalb der Beschriftung eines Texts ein (mittels `VisualBrush`) Video wiederzugeben.

1.5.1 Ressourcen

Ressourcen sind eine Möglichkeit, mehrmals benutzte Objekte und Werte auszulagern, um von mehreren Stellen darauf zu verweisen. Jedes Objekt, dessen Klasse von `FrameworkElement` oder `FrameworkContentElement` ableitet, besitzt ein `Resources`-Attribut. Dieses enthält ein *Dictionary* von Ressourcen, auf die anhand des Attributs `x:Key` von anderen Stellen durch die *Markup Extensions* `StaticResource` und `DynamicResource` verwiesen werden kann.

Dies wird in *Kapitel 1.4.5 auf Seite 11* verwendet, um von zwei Buttons aus auf denselben `Brush` zu verweisen. Weitere Informationen zu Ressourcen und dem Unterschied zwischen dynamischen und statischen Ressourcen bietet das .NET Framework Developer Center.¹⁰

1.5.2 Control Templates

Die Möglichkeiten der Elementkomposition können durch *Templates* genutzt werden. *Control Templates* sind Vorlagen, die angeben, wie ein Bedienelement dargestellt werden soll, und eingesetzt werden, wenn assoziierte Bedienelemente dargestellt werden sollen.

Hierzu kann man innerhalb eines `ControlTemplate`-Elements beliebige Anzeige-Elemente deklarieren. Diese werden dargestellt, wenn ein Control mit diesem Template verknüpft

¹⁰<http://msdn2.microsoft.com/en-us/library/ms750613.aspx>

wird. Control Templates können über das Attribut `TargetType` einschränken, auf welche Typen von Zielobjekten sie angewendet werden sollen.

Eine wichtige Rolle nimmt innerhalb des `ControlTemplate`s der `ContentPresenter` ein. Dieses Element ist für die Darstellung der *Content Property* des Controls zuständig, also beispielsweise die Beschriftung `Content` eines Buttons. Wird dieses Element im `ControlTemplate` nicht angegeben, so wird ein Button, der mittels dieses Templates dargestellt wird, keine Beschriftung haben.

1.6 Datenbindung

Als Datenbindung (engl. “Data Binding”) bezeichnet man die Kopplung der Anzeige an eine Datenquelle, beispielsweise das Domänenmodell des Programms. Hierdurch wird die Anzeige automatisch aktualisiert, wenn sich die Datenquelle ändert.

1.6.1 Herstellen einer Verbindung

Als Vermittler zwischen einer Datenquelle, der sogenannten *Binding Source*, und dem gebundenen Element (*Binding Target*) dient eine Instanz der `Binding`-Klasse. Eine solche Verbindung besteht aus vier Komponenten:

- Zielobjekt

Das *Zielobjekt* ist dasjenige Objekt, dessen Attribut seinen Wert aus der Datenquelle bezieht. Dies ist üblicherweise ein Anzeige-Element wie ein `Button` oder eine `ListBox`. Das *Zielobjekt* muss eine Instanz der Klasse `DependencyObject` (oder einer davon abgeleiteten Klasse) sein.

- Zieleigenschaft

Die Zieleigenschaft ist das Attribut des Zielobjekts, das durch das *Binding* gebunden wird. Soll die Beschriftung eines Buttons an eine Datenquelle gebunden werden, so ist die Eigenschaft `Content` von `Button` die Zieleigenschaft des *Bindings*.

- Datenquelle

Dasjenige Objekt, das die Daten bereitstellt, beispielsweise ein Objekt des Domänenmodells.

- Pfad zum gebundenen Wert

Bei einem *Binding* muss der Pfad zum gebundenen Wert angegeben werden. Dies ist der Name der gebundenen Eigenschaft. Dies soll an einem Beispiel verdeutlicht werden: Als Datenquelle wird eine Instanz einer Klasse `Movie` angenommen, die Informationen zu einem Spielfilm enthält. Einen Pfad stellen hier die Eigenschaften `Title` oder `Director` dar, um an den Titel oder Regisseur dieses Films zu binden. Ein Pfad kann auch komplizierter sein und über mehrere Knoten eines Objektgraphs gehen: `Actors[0].Name.LastName`.

Beispiel für eine Binding-Angabe per Markup Extension:

```
<TextBlock Text="{Binding Path=Title Source={StaticResource theMovie}}">
```

Es gibt mehrere Möglichkeiten, wie ein Binding auf Veränderungen der assoziierten Objekte reagieren kann:

- **OneWay** übernimmt Veränderungen der Datenquelle in das Zielobjekt.
- **OneWayToSource** funktioniert genauso, allerdings in die Gegenrichtung. Dies ist nützlich bei Eingabe-Elementen auf der Oberfläche, deren Werte nicht automatisch angepasst werden sollen.
- **TwoWay** vereint die beiden obigen Modi: Änderungen der Datenquelle werden vom Zielobjekt übernommen, und dessen Änderungen werden zurück in die Datenquelle geschrieben. Dies kann z. B. in Formular-Oberflächen verwendet werden, die Informationen in Eingabefeldern anzeigen.
- **OneTime** ist ein Sonderfall: Die Richtung der Datenübertragung ist dieselbe wie bei **OneWay** (von Datenquelle zu Zielobjekt), allerdings nur ein einziges Mal bei der Initialisierung des Bindings. Dies ist bei statischen Datenquellen nützlich, oder wenn eine Momentaufnahme der Datenquelle angezeigt werden soll.

1.6.2 DataContext

Es gibt eine weitere Möglichkeit, die Datenquelle für Bindings anzugeben. Hierzu setzt man das Attribut **DataContext** eines Container-Elements auf die gewünschte Datenquelle. Jedes innerhalb des Container-Elements gebundene Element wird, sofern die Datenquelle nicht im Binding angegeben ist, auf die im Datenkontext eingetragene Datenquelle zurückgreifen. Dies kann hilfreich sein, wenn innerhalb eines Container-Elements mehrere Elemente auf dieselbe Datenquelle zugreifen sollen, jedoch an verschiedene Pfade gebunden sind.

1.6.3 Value Converter

WPF kennt mehrere Möglichkeiten zur automatischen Umwandlung von Werten einer Datenquelle, um diese darzustellen. Beispielsweise können primitive Datentypen problemlos dargestellt werden, und die meisten anderen Typen werden per `ToString()` in einen String umgewandelt, der anschließend dargestellt wird. Um einfach eine sinnvolle Ausgabe zu erhalten, wird `ToString` des Datums überschrieben, so dass es die gewünschten Ausgabe liefert.

Eine weitere Möglichkeit, darstellbare Informationen aus dem internen Datum zu erstellen, die keine Veränderung der Datenobjekt-Klasse erfordert, sind *Value Converter*. Diese dienen der Umwandlung einer internen Datenrepräsentation in die darzustellende Form (und umgekehrt, soll eine formatierte Eingabe in die Datenhaltung übernommen werden). Auf

diese Weise kann zum Beispiel ein Temperatur-Datum in Grad Celsius für die Anzeige in Grad Fahrenheit umgewandelt werden (siehe *Listing 8*).

```
public class CelsiusToFahrenheitConverter : IValueConverter {
    public object Convert(object value, ...) {
        if (((string)value) == "")
            return 0.0;
        return value*9/5 + 32;
    }
    ...
}
```

Listing 8: *Value Converter* zur Umwandlung von Celsius in Fahrenheit

1.6.4 Data Templates

Für anspruchsvollere Umwandlungsoperationen, die nicht nur die dargestellten Werte, sondern die Darstellung ansich beeinflussen, gibt es *Data Templates*. Diese erlauben die Veränderung des Baums visueller Elemente, der zur Wertdarstellung erzeugt wird. *Abbildung 4* zeigt eine Collection in einer *ListBox* an, ohne dass *Value Converter* oder *Data Templates* eingesetzt werden. Überschreiben der *ToString()*-Methode, oder das Erstellen eines *Value Converters* kann jedoch nicht den Elementbaum beeinflussen, so dass hierbei komplexere Formatierung ausgeschlossen ist.

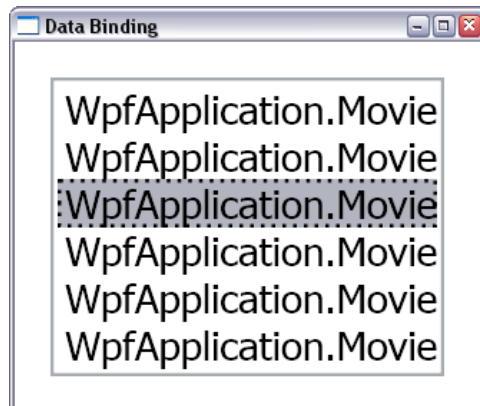


Abbildung 4: Darstellung einer *Collection* mehrerer Objekte ohne *Data Template*

Für diesen Fall gibt es Datenvorlagen (*Data Templates*). Die Funktionsweise ist ähnlich derer von Bedienelementvorlagen: Wann immer ein Objekt dargestellt werden soll, für das ein geeignetes Template existiert (s.u.), werden die Angaben des Templates zur Darstellung verwendet (*Abbildung 5*).

Innerhalb einer Datenvorlage wird der implizite Datenkontext des dargestellten Datenobjekts angenommen. So kann man den Inhalt einzelner Bedienelemente an Eigenschaften des Objekts binden. *Listing 9* ist ein Auszug der Datenvorlage für die oben gezeigte Objektliste.

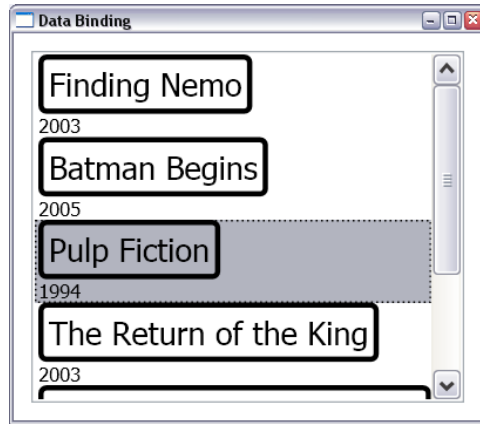


Abbildung 5: Darstellung mit passendem *Data Template*

```
<DataTemplate x:Key="MovieTemplate" DataType="{x:Type l:Movie}">
  <StackPanel ...>
    <Border ...>
      <TextBlock FontSize="24" Text="{Binding Title}"/>
    </Border>
    <TextBlock Text="{Binding Year}"/>
  </StackPanel>
</DataTemplate>
```

Listing 9: *DataTemplate* für eine Collection von *Movie*-Objekten

Datenvorlagen sind insbesondere bei der Auflistung mehrerer, gleichartiger Domänenmodell-Objekte sinnvoll.

1.7 Bedienelemente und Layouts

WPF stellt bereits von Haus aus eine Vielzahl von Bedienelementen bereit. Eine vollständige Referenz bietet das .NET Framework Developer Center¹¹.

1.7.1 Layouts

Windows Presentation Foundation verfügt über sechs *Panel*-Klassen, die enthaltene Oberflächen-Elemente auf verschiedene Arten anordnen:

- **Canvas**

Der Element-Container ohne Layout. Enthaltene Elemente müssen relativ zu den Grenzen des Canvas positioniert werden.

¹¹<http://msdn2.microsoft.com/en-us/library/ms752324.aspx>

- **DockPanel**

Im Dock-Panel werden die enthaltenen Elemente nacheinander an einer Seite der noch freien Fläche ausgerichtet. Jedes Element nimmt dadurch einen Teil des Platzes der noch freien Fläche ein. Das zuletzt hinzugefügte Element nimmt den gesamten restlichen Platz ein.

- **StackPanel**

Die Kindelemente des **StackPanel**s werden in einer Reihe angeordnet, entweder oben beginnend nach unten oder links beginnend nach rechts.

- **WrapPanel**

WrapPanel ordnet Kindelemente vergleichbar einem Textfluss an: Oben beginnend, von links nach rechts, und in die nächste Zeile umbrechend, falls ein Element aufgrund seiner Größe nicht mehr in die aktuelle Zeile passt.

- **VirtualizingStackPanel**

Dieser Container verwendet *Virtualisierung*, eine Technik, um performant einige wenige Elemente von vielen anzuzeigen. **ListBox** verwendet dieses Panel, um enthaltene Listenelemente anzuzeigen. Anstatt möglicherweise die Anzeige vieler hundert Elemente (oder mehr) zu berechnen, und nur einen Teil davon im Inhaltsbereich des Containers anzuzeigen, berechnet **VirtualizingStackPanel**, welche Elemente angezeigt werden können. Nur deren Ausgabe wird berechnet.

- **Grid**

Grid ist das wohl flexibelste Layout-Panel in WPF. Es stellt Elemente tabellarisch in Zeilen und Spalten dar, wobei die Elemente jeweils mehrere Zeilen und Spalten einnehmen können. Hierbei sind die Größen der Zeilen und Spalten voneinander unabhängig, und können mittels **GridSplitter** vom Anwender in der Größe individuell angepasst werden.

1.8 Styles

Styles werden in WPF verwendet, um die Darstellung von Elementen anzupassen. Im Gegensatz zu Control Templates können Styles jedoch nur die Eigenschaften der Elemente verändern, nicht jedoch den Element-Baum selbst. Dies ist Templates vorbehalten.¹²

Styles werden mit den gleichnamigen **Style**-Elementen erstellt. Genau wie **ControlTemplate** haben diese Elemente ein **TargetType**-Attribut zur Angabe des Elementtyps, auf den der Style anzuwenden ist. Dies bestimmt, welche Eigenschaften von Attributen durch den Style gesetzt werden können.

¹²Styles können den Wert des Attributs **Template** von Elementen ändern, so dass andere *Control Templates* über den Style angegeben werden können

Styles können Setter und Trigger enthalten. Setter geben an, welches Attribut eines Elements welchen Wert zugewiesen bekommen soll. So kann man beispielsweise die Hintergrundfarbe oder den Außenabstand (**Margin**) eines Elements verändern.

Trigger werden in Styles genau wie in Templates verwendet, um je nach Wahrheitswert einer Bedingung weitere Setter anzuwenden. Styles können mehrere Trigger haben, die jeweils beliebig viele Setter anwenden können.

Listing 10 stellt einen Button-Style dar, der die Breite und Höhe der Buttons bestimmt, sowie deren Vorder- und Hintergrundfarben, falls die Maus des Anwenders über den Button fährt¹³.

```
<Style TargetType="{x:Type Button}" ...>
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="Button.Background" Value="Yellow" />
      <Setter Property="Button.Foreground" Value="White" />
    </Trigger>
  </Style.Triggers>
  <Setter Property="Button.Width" Value="50" />
  <Setter Property="Button.Height" Value="30" />
</Style>
```

Listing 10: Beispiel für *Styles* mit Trigger

1.9 Themes

Um alle darstellungsbezogenen Informationen an einer Stelle zu vereinen, können alle Ressourcen, Stile und Bedienelementvorlagen in einem *Theme* gebündelt werden. Ein Theme wird durch eine XAML-Datei im Verzeichnis **themes** definiert, die einen beliebigen Namen tragen kann. Auf diese Weise kann das Aussehen einer Applikation von Grund auf geändert werden, wenn ein anderes Theme geladen wird.

Themes sind des Weiteren ein Konzept in der Windows-Betriebssystem-Familie. Sie erlauben dem Anwender, das Erscheinungsbild des Systems zu variieren. Möchte man die Bedienelemente einer Anwendung für die Darstellung in diesen Themes optimieren, so kann man die Unterstützung von WPF für Themes verwenden, um Darstellungsvarianten eigener Bedienelemente zu erschaffen.¹⁴ Auf diese Weise fügen diese sich in das Gesamterscheinungsbild des Systems harmonisch ein. Hierzu muss man für jedes zu unterstützende Theme eine Datei anlegen, die ein **ResourceDictionary** als Wurzel enthält. Diese Datei muss eine bestimmte Bezeichnung tragen:

- **themes\ aero.xaml** für die Verwendung im Windows-Vista-Theme
- **themes\ royal.xaml** für das Windows-Media-Center-Theme

¹³Dieses Beispiel ist angelehnt an Listings von [Cleeren 2006]

¹⁴Siehe [WPF SDK 2007]

- `themes\luna.normalcolor.xaml`, `themes\luna.metallic.xaml`,
`themes\luna.homestead.xaml` für die verschiedenen Farbvarianten des Windows-XP-Themes
- `themes\classic.xaml` für das klassische Windows-Theme (z. B. Windows 98, 2000)
- `themes\generic.xaml` als generische Definition für die Verwendung in allen Themes, die nicht durch eine spezielle Theme-Definition unterstützt werden

Die von Microsoft in WPF mitgelieferten Bedienelemente nutzen dieselbe Funktion, um je nach Betriebssystem-Theme anders dargestellt zu werden. Aufgrund dessen kann eine Applikation auch angewiesen werden, die Theme-Informationen eines anderen Systems zu verwenden, um beispielsweise den Aero-Look von Windows Vista auf Windows XP zu erzwingen¹⁵.

1.10 Weitere Funktionen

WPF bietet mehr als die bislang angesprochenen Grundlagen. Weitere Merkmale von WPF werden in den folgenden Abschnitten vorgestellt. Für weitergehende Informationen zu diesen Themen sei der interessierte Leser auf die Dokumentation von Microsoft¹⁶ verwiesen.

1.10.1 Dokumente und Text

WPF bietet umfangreiche Unterstützung für die Darstellung von Dokumenten. Bei Dokumenten wird zwischen zwei Arten unterschieden: Zum Einen Dokumente, deren Layout sich der Ausgabegröße anpassen kann, indem Elemente anders positioniert, und Zeilen an anderen Stellen umgebrochen werden und zum Anderen Dokumente, deren Layout fixiert ist. Dokumente können beispielsweise Paragraphen, Listen, Tabellen und Bilder enthalten; selbst die Einbettung von funktionalen Bedienelementen ist möglich. WPF bietet Unterstützung für die Speicherung von Dokumenten in den Formaten [XPS](#) und Office Open XML ([.docx](#))¹⁷.

Windows Presentation Foundation beherrscht Silbentrennung und die Darstellung von Blocksatz und besitzt umfangreiche Unterstützung für die Darstellung von Text mit Unterschneidung, Ligaturen und Ähnlichem.

Weiterführende Informationen zu Dokumenten in WPF bietet das .NET Framework Developer Center¹⁸.

¹⁵Microsoft hat die von WPF-Controls verwendeten Theme-Dateien für die Betriebssystem-Themes veröffentlicht. Sie sind einsehbar unter <http://msdn2.microsoft.com/en-us/library/aa358533.aspx>

¹⁶<http://msdn2.microsoft.com/en-us/library/ms754130.aspx>

¹⁷Wobei ggf. enthaltene Bedienelemente natürlich ihre Funktion verlieren.

¹⁸<http://msdn2.microsoft.com/en-us/library/ms748388.aspx>

1.10.2 Medien und 3D

WPF bietet die Wiedergabe von Video- und Audiodateien. Hierzu kann man die Klasse `MediaElement` verwenden, die beides unterstützt, und im Layout-System verwendet werden kann. Auf diese Weise werden visuelle Medieninhalte, Text und Bedienelemente gemeinsam verwendet. Einen guten Einstieg hierzu bietet die Übersichtsseite “Multimedia” im .NET Framework Developer Center¹⁹.

Dreidimensionale Objekte werden in WPF innerhalb einer Instanz des `Viewport3D` angezeigt. Diese Objekte sind durch Punkte im dreidimensionalen Raum und die dadurch aufgespannten Dreiecke definiert. Den so entstandenen Oberflächen können verschiedene Beschaffenheiten zugeordnet werden, die Licht unterschiedlich reflektieren. Licht kann aus verschiedenen Lichtquellen stammen, und beispielsweise Umgebungslicht (ohne Richtung oder Ursprung), gerichtetes Licht oder punktförmiges Licht, das von einem Ort in jede Richtung strahlt, sein. All dies ist objektorientiert umgesetzt und kann in XAML deklariert werden. Weitere Informationen hierzu bietet die Dokumentation von WPF²⁰.

1.11 Entwicklungswerkzeuge

Microsofts Entwicklungsumgebung für .NET ist Visual Studio. Die Version 2008 bietet Unterstützung für das .NET-Framework in Version 3.5. Zur Erstellung von XAML-Benutzeroberflächen können mehrere Werkzeuge verwendet werden: Visual Studio bietet einen visuellen Editor für Benutzeroberflächen. Zudem bietet Microsoft mit der Expression-Produktreihe mehrere Programme zur Verwendung durch Designer: Blend ist zur Erstellung komplexer Oberflächen gedacht und bietet WYSIWYG-Unterstützung bei der Erstellung von XAML-GUIs. Die Erstellung von Vektorgrafiken und deren Export in XAML kann mit Expression Design geschehen.

1.12 Deployment

WPF-Anwendungen können eine der beiden in diesem Kapitel diskutierten Ausprägungen annehmen.

1.12.1 Stand-Alone

Mit WPF entwickelte Anwendungen können als “Stand-Alone”-Anwendungen verteilt werden. Diese werden installiert und dadurch permanent auf dem Computer des Anwenders gespeichert, haben Zugriff auf das Dateisystem und können beispielsweise Dateiendungen registrieren.

¹⁹<http://msdn2.microsoft.com/en-us/library/aa970915.aspx>

²⁰<http://msdn2.microsoft.com/en-us/library/ms747437.aspx>

1.12.2 XBAP

Es gibt zusätzlich das Deployment-Modell der *XAML Browser Applications* (XBAP). Dies sind WPF-Anwendungen, die im Web-Browser aufgerufen werden, und nicht permanent auf dem Computer des Anwenders gespeichert sind.

XBAPs werden in einer *Partial-Trust*-Umgebung ausgeführt²¹, ähnlich wie Java Applets, in der einige Funktionen gesperrt sind. Es ist beispielsweise nicht möglich, aus einer XBAP-Anwendung auf das Dateisystem des Nutzers zuzugreifen. Zudem kann man keine Programmfenster erzeugen²².

Stattdessen muss man eine navigationsbasierte Applikation mit Seiten (*Pages*) schreiben, die in einem einzigen Fenster mit Web-ähnlicher Navigation ausgeführt wird. Die Navigationsleiste kann jedoch ausgeblendet werden, falls die Anwendung nicht innerhalb eines Browsers ausgeführt wird.

Der Aufruf einer XBAP im Browser ist im Internet Explorer und, seit dem .NET-Framework 3.5, auch in Firefox möglich. Hierbei wird kein Sicherheitsdialog angezeigt.

Der Grundgedanke hinter diesen Browser-Applikationen ist die Übertragung des Navigationsmodells des Web auf Desktop-Anwendungen. So kann man in diesen Applikationen wie in einem Web-Browser vor und zurück gehen, sowie zu verschiedenen Stellen des Verlaufs springen.²³ Werden diese Applikationen direkt im Browser ausgeführt, so ist der Wechsel von der Webseiten-Navigation zur XBAP-Programmnavigation transparent.

1.12.3 ClickOnce

ClickOnce ist eine Installations-Technologie von Microsoft für Windows-Forms- und WPF-Programme. Die Installation wird durch den Aufruf einer Website nach Bestätigen eines Dialogs gestartet. Mit ClickOnce installierte Programme können sehr leicht aktualisiert werden. Die Verwendung von ClickOnce hat den weiteren Vorteil, dass diese Programme einen integrierten Update-Mechanismus haben²⁴. Zudem werden sie nur für den aktuellen Anwender installiert, die Installation benötigt kein Administrator-Passwort.

Anstatt mit einem Installationsprogramm können WPF-Anwendungen (sowohl “stand-alone” als auch XBAP) mittels ClickOnce installiert werden. ClickOnce ermöglicht sowohl die permanente Installation von “Stand-Alone”-Software, als auch das vorübergehende Zwischenspeichern und Ausführen von Browser-basierten Applikationen (XBAPs).

²¹Siehe auch [Corby 2006]

²²Weitere Informationen zum Sicherheitsmodell unter <http://msdn2.microsoft.com/en-us/library/aa480229.aspx> und <http://msdn2.microsoft.com/en-us/library/aa970910.aspx>

²³Weitere Informationen unter [MSDN Magazine 2006]

²⁴[MSDN ClickOnce]

2 Silverlight

Microsoft Silverlight ist eine Laufzeitumgebung für browserbasierte Applikationen. Silverlight wurde unter dem Codenamen “WPF/E” (für “WPF Everywhere”) entwickelt. Es soll die Windows Presentation Foundation mit dem Web-Browser verbinden, um sogenannte [Rich Internet Applications](#) zu ermöglichen.

2.1 Version 1.0

Silverlight 1.0 wurde von Microsoft im September 2007 veröffentlicht. Dieses Release beschränkt sich auf die XAML-basierten Funktionen von WPF und ermöglicht die Anzeige von Grafikprimitiven innerhalb eines **Canvas**-Container-Elements, welches keine Layout-Funktionalität bereitstellt. Jegliche darüber hinausgehende Funktionalität muss mittels [JavaScript](#) erfolgen.

2.1.1 Browser-Integration

Silverlight-Anwendungen werden in HTML eingebettet, ähnlich Adobe Flash. Dadurch ist eine Interaktion zwischen Elementen der Webseite und der Silverlight-Applikation über JavaScript möglich.

Silverlight ist ein Plugin für Microsoft Internet Explorer und Mozilla Firefox auf Windows, sowie Safari für Mac OS X, Unterstützung für Opera soll mit einer späteren Version folgen. Silverlight gibt es nicht für Linux. Im Rahmen des [Mono](#)-Projekts, das eine freie Implementierung des .NET-Frameworks anstrebt, und mit Unterstützung durch Microsoft²⁵, wird unter der Bezeichnung *Moonlight* versucht, die Silverlight-Technologie auch für andere Plattformen, speziell Linux, verfügbar zu machen.²⁶ Jedoch wurde noch kein Veröffentlichungsdatum einer funktionsvollständigen, kompatiblen Version bekanntgegeben²⁷.

2.1.2 Audio- und Video-Funktionalität

Silverlight unterstützt das Abspielen der Audioformate [MP3](#) und [WMA](#), sowie der Videoformate [WMV7](#), [WMV8](#) und [WMV9/VC-1](#).

2.1.3 Programmierung

Silverlight selbst stellt nur die Anzeige eines XAML-Canvas bereit. Dieser kann weitere Elemente enthalten, gibt jedoch kein Layout vor – enthaltene Elemente müssen absolut positioniert werden. Silverlight 1.0 enthält keine Controls wie Buttons oder Listen. Events auf dargestellten Elementen (wie beispielsweise der Mausklick des Anwenders) können, vergleichbar wie der Event-Handler-Aufruf in WPF, an JavaScript-Funktionsaufrufe gebunden werden. Allerdings fehlt die Möglichkeit, *Code behind* in einer .NET-Programmiersprache

²⁵Siehe [[Guthrie 2007](#)]

²⁶Weitere Informationen zu Moonlight bietet die offizielle Website [[Moonlight](#)]

²⁷Miguel de Icaza berichtete Anfang Dezember 2007 in [[de Icaza 2007](#)] von erheblichen Fortschritten.

zu erstellen und mit der Oberfläche zu verbinden. Somit müssen komplexe Benutzerinteraktionen über JavaScript und Browser-Bedienelemente verlaufen. Silverlight stellt allerdings ein Downloader-Objekt bereit, mit dessen Hilfe Daten von einem Server nachgeladen werden können, ohne dass eine neue Website aufgerufen werden muss.²⁸²⁹

2.2 Weitere Entwicklung

Zum Zeitpunkt der Veröffentlichung dieses Dokuments gibt es zusätzlich zur fertigen Version 1.0 von Silverlight ein Alpha-Release von Version 1.1³⁰, die jedoch noch nicht funktionsvollständig ist. Auf deren genauen Funktionsumfang soll an dieser Stelle nicht weiter eingegangen werden. Microsoft hat angekündigt, die funktionsvollständige Betaversion im März 2008 zu veröffentlichen, welche ab dann mit der Versionsnummer 2.0 ausgezeichnet ist.³¹ Ein endgültiger Veröffentlichungstermin wurde noch nicht bekannt gegeben.³² Der folgende Abschnitt gibt die von Microsoft-Entwicklern veröffentlichten Informationen zu Silverlight 2.0 wieder.

2.2.1 Angekündigte Funktionen

Silverlight 2.0 wird laut Microsoft in vielen zentralen Bereichen Fortschritte machen. So soll es die meisten Bedienelemente von WPF enthalten, um eine größere Konsistenz zwischen den beiden Technologien zu schaffen. Im Gegensatz zu WPF verwendet Silverlight jedoch kein DirectX, um die Anzeige der Grafikelemente zu beschleunigen.

Die Steuerung dieser Bedienelemente wird zusätzlich zu JavaScript über [C#](#) und VB.NET erfolgen.³³

Silverlight 2.0 wird eine [CLR](#) für .NET enthalten. Diese wird die *Base Class Library* von .NET umfassen, die unter anderem [LINQ](#)³⁴, Collections und reguläre Ausdrücke unterstützt. Darüber hinaus wird Silverlight die *Dynamic Language Runtime* enthalten, die im Quellcode vorliegende Scripts dynamischer Programmiersprachen wie *Python* und *Ruby* ausführen kann.³⁵

Die Bedienelemente werden Unterstützung für Datenbindung, Templates und Styles haben. Zudem wird es mehrere einfache Layout-Container geben. Ziel in der Entwicklung von Silverlight ist unter anderem die Unterstützung eines großen Teils von WPF, um die

²⁸[\[MSDN Silverlight\]](#)

²⁹Weitere Informationen zur Programmierung von Silverlight bietet [\[Silverlight\]](#)

³⁰Ankündigung siehe [\[Van Patten 2007\]](#)

³¹Aus [\[Sneath 2007\]](#)

³²Microsoft entwickelt laut [\[Somasegar 2008\]](#) eine Silverlight-Anwendung für die Olympischen Spiele 2008, so dass angenommen werden kann, dass Silverlight 2.0 bis August veröffentlicht wird.

³³Vermutlich werden alle .NET-Programmiersprachen unterstützt, [\[Sneath 2007\]](#) beschränkt sich aber auf die Nennung dieser beiden.

³⁴Eine Untermenge von LINQ ist laut [\[Hamilton 2007\]](#) bereits in Silverlight 1.1 enthalten

³⁵In anderen Programmiersprachen, wie C# oder VB.NET, geschriebener Code muss vor der Ausführung mit Silverlight kompiliert werden.

Umwandlung einer Silverlight-Anwendung in WPF zu ermöglichen.³⁶

Zudem kann aus dem *Code-behind* von Silverlight heraus auf das [DOM](#) der HTML-Seite zugegriffen werden, die das Silverlight-Objekt umgibt, um diese ähnlich wie mit JavaScript zu manipulieren. Silverlight wird den Zugriff auf Web Services, sowie die Verarbeitung der Datenaustauschformate [XML](#) und [JSON](#) unterstützen.

[Guthrie 2008] beschreibt die Erstellung einer einfachen Anwendung mit Silverlight 2.0 und bietet dadurch einen Ausblick auf kommende Funktionen.

2.3 Werkzeuge

Für Silverlight bieten sich dieselben Werkzeuge wie für die Entwicklung von WPF-Applikationen an: Microsoft Visual Studio.NET und Microsoft Expression Blend. Die Anfang Dezember 2007 veröffentlichte Beta-Version von Expression Blend 2 unterstützt die Erstellung von Silverlight-Oberflächen.

2.4 Vergleich zu Flash

Microsoft positioniert Silverlight als Konkurrenzprodukt zu Adobe Flash. Durch die fundamentalen Veränderungen mit Silverlight 2.0 lassen sich diese Produkte jedoch nur schwer vergleichen. Zudem sind noch nicht alle Eigenschaften von Silverlight 2.0 bekannt. Aus diesem Grund verzichtet der Autor auf ein abschließendes Fazit am Ende dieser Gegenüberstellung der beiden Technologien.

2.4.1 Vorteile von Silverlight

Als Vorteil von Silverlight ist hierbei zunächst das einfache Dateiformat zu nennen: In der aktuellen Implementierung basiert Silverlight lediglich auf JavaScript zur Programmierung und XAML als XML-Beschreibung der Oberfläche. Diese beiden Formate können mit jedem Editor bearbeitet werden; zudem stellt Microsoft kostenlos Werkzeuge zur Arbeit mit XAML zur Verfügung. Außerdem kann in XAML geschriebener Text von Suchmaschinen berücksichtigt werden, da es ein XML-Format ist. Auf der anderen Seite gibt es das kommerzielle Flash von Adobe, dessen Dateiformat nur schlecht dokumentiert ist.³⁷

Des Weiteren beherrscht Windows Presentation Foundation, und somit auch Silverlight, zeitabhängige Animationen. Je nach Rechengeschwindigkeit können diese Animationen aus vielen oder wenigen Einzelbildern bestehen. So kann man Start- und Endpunkt einer Animation, sowie die gewünschte Dauer in Sekunden angeben, und Silverlight übernimmt die exakte Berechnung. Im Gegensatz hierzu arbeitet Flash mit Transformationsmatrizen, welche auf jedes Einzelbild angewendet werden. Soll ein Element bewegt werden, muss zuerst die Anzahl der Einzelbilder berechnet und danach für jedes Einzelbild eine Transformationsmatrix erstellt werden, die den Bildinhalt um einen Bruchteil der Gesamtstrecke

³⁶[Guthrie 2008] demonstriert diese Umwandlung an einer Beispielanwendung.

³⁷[Ezell 2007]

verschiebt. So können Animationen auf schnellen Rechnern insgesamt schneller ablaufen als auf langsamen.³⁸

Wenn Silverlight 2.0 veröffentlicht wird, hat es bei der Programmierung einen weiteren Vorteil gegenüber Flash, da dieses auch in C# und VB.NET programmiert werden kann – und Entwickler die umfangreiche Klassenbibliothek von .NET und ihre Vorkenntnisse in der .NET-Entwicklung verwenden können.

2.4.2 Vorteile von Flash

Silverlight unterstützt nur das Audioformat MP3 und die Windows-Media-Produkte WMA und WMV. Flash bietet seit Version 9 eine Vielzahl von Alternativen, unter anderem [MPEG-4](#), [3GP](#) und [QuickTime](#). Zudem können mit Flash Webcam und Mikrofon angesteuert werden.³⁹

Flash wird bereits mit einer Vielzahl von Bedienelementen ausgeliefert, wohingegen Silverlight 1.0 so etwas nicht bietet. Drittanbieter entwickeln jedoch bereits Bedienelemente für Silverlight 1.0.

Flash bietet ein einfacheres Deployment: Es genügt eine [SWF](#)-Datei, die alle benötigten Ressourcen enthält. Bei Silverlight 1.0 müssen mehrere Dateien ausgeliefert werden: Mehrere JavaScript-Dateien, alle Mediendateien, sowie eine XAML-Datei. Die Verwendung von XAML führt zu größeren Dateien. Dieser Nachteil wird durch komprimierte [.xap](#)-Dateien, die alle benötigten Ressourcen enthalten können, in Silverlight 2.0 behoben.⁴⁰ Man kann Flash auch als selbstständiges, ausführbares Programm ausliefern; dies ist bei Silverlight nicht möglich.

Darüber hinaus hat Flash Anfang 2008 noch den Vorteil, dass es auf sehr vielen Computern bereits installiert ist. Neue Silverlight-Anwendungen müssen die Hürde der Installation der Laufzeitumgebung auf den Computern der Anwender erst noch überwinden.

³⁸Nach [Ezell 2007]; eine andere Sichtweise beschreibt [Pfeil 2007]

³⁹[Pfeil 2007]

⁴⁰Siehe [Guthrie 2008]

3 Vergleich von WPF und Silverlight

Im Folgenden werden vier unterschiedliche Zielplattformen miteinander verglichen: “Stand-Alone”-WPF-Anwendungen und XAML Browser Applications basieren zwar auf derselben Technologie, unterscheiden sich jedoch in zentralen Punkten der Darstellung. Bei Silverlight werden die Versionen 1.0 und 2.0 getrennt betrachtet, da sie sich hinsichtlich der Programmierung grundlegend unterscheiden.

Im Folgenden werden die Vor- und Nachteile jeder Möglichkeit genannt. Im Anschluss gibt der Autor eine Wertung der Vor- und Nachteile ab.

3.1 Kontext

Diese Arbeit vergleicht die oben genannten Plattformen hinsichtlich ihrer Tauglichkeit für das Studienprojekt *ViCCC*⁴¹ an der Universität Stuttgart. Hierbei handelt es sich um eine interaktive Mehrspieler-Simulation, in welcher jeder Spieler ein Bauunternehmen führt. Alle Nutzer verwenden (oder haben Zugang zu) Microsoft Windows. Der Anwenderkreis ist bekannt und beschränkt sich auf einen Teil der Studenten der Universität Stuttgart.

3.2 WPF-Applikation

Windows Presentation Foundation ist ein mächtiges Werkzeug zur Erstellung grafischer Benutzeroberflächen. Die einzigen Nachteile betreffen das Deployment der Anwendung: Anwender müssen WPF-Applikationen vor der Verwendung herunterladen und installieren. Durch ClickOnce ist dies allerdings komfortabel. Ein weiterer Nachteil ist, dass WPF-Applikationen nur auf Microsoft Windows ausgeführt werden können.

3.3 XAML Browser Application (XBAP)

XBAP hat den Vorteil, dass es, anders als eine normale WPF-Applikation, keine dauerhafte Installation benötigt. Problematisch ist allerdings, dass XBAPs in einer Sandbox ausgeführt werden, in der sie nur wenige Rechte haben (*partial trust*). Deshalb steht nicht die ganze Funktionalität des .NET-Frameworks zur Verfügung. Dies führt unter anderem dazu, dass keine weiteren Programmfenster erzeugt werden können. Ansonsten sind die beiden Varianten bezüglich der Entwicklung identisch.

3.4 Silverlight 1.0

Die Nutzung von Silverlight-Anwendungen ist aus Benutzersicht ähnlich der von XBAPs. Darüber hinaus ermöglicht Silverlight auch die Verwendung von Mac OS X als Betriebssystem mit Safari als Web-Browser.

Die Silverlight-Oberfläche kann mit JavaScript gesteuert werden. Silverlight-Elemente können in HTML-Seiten, ähnlich wie Flash-Elemente, eingebettet werden.

⁴¹<http://www.iste.uni-stuttgart.de/se/stupros/a/>

Demgegenüber stehen die Nachteile von Silverlight 1.0. Es ist keine Entwicklung mit C# und dem .NET-Framework möglich; des Weiteren steht lediglich JavaScript als Programmiersprache zur Verfügung. Es fehlen die für anspruchsvolle Benutzeroberflächen unverzichtbaren Steuerelemente und flexible Layout-Möglichkeiten, was den Entwicklungsaufwand erhöht.

3.5 Silverlight 2.0

Silverlight 2.0 enthält wesentlich mehr Funktionalität als sein Vorgänger und steht XBAP-Anwendungen in nichts nach. Der große Vorteil wird die Verfügbarkeit für Opera und Mac OS X sein. Silverlight 2.0 befindet sich bis März 2008 im Alpha-Stadium seiner Entwicklung.⁴² Es wurde noch kein Veröffentlichungstermin für das finale Release genannt.

3.6 Fazit und Empfehlung

Obwohl Silverlight als Technologie äußerst interessant ist, sieht der Autor keine Vorteile in dessen Einsatz als Entwicklungsplattform. Silverlight enttäuscht in der aktuellen Fassung durch mangelhafte Funktionalität. Zudem entsteht ein großer Mehraufwand durch die Nutzung von JavaScript. Da noch kein Veröffentlichungstermin für Silverlight 2.0 genannt wurde, erscheint dieses Release wohl frühestens Mitte 2008. Hierdurch disqualifiziert es sich für den produktiven Einsatz in naher Zukunft. Die Verfügbarkeit von Silverlight für eine größeren Anzahl von Browser-Plattformen kann diese Nachteile gegenüber WPF nicht aufwiegen.

XBAP überzeugt durch das einfache Deployment-Modell ohne Installation. Wenn die eingeschränkte *partial trust*-Umgebung mächtig genug für den Einsatzzweck ist und keine Notwendigkeit für die Unterstützung von Opera oder Mac OS X besteht, bietet sich die Entwicklung von Software als XBAP an. Diese kann auch als "Stand-Alone"-Programm angeboten werden. Auch die gleichzeitige Verwendung verschiedener Deployment-Modelle ist denkbar.

⁴²Bis dahin als Silverlight 1.1 alpha, siehe [2.2](#)

4 Appendix

Glossar

Zu vielen Begriffen ist nach der Kurzbeschreibung eine Webreferenz angegeben, die weitere Informationen zum Begriff bietet.

3GP 3GP ist eine vereinfachte Version des MPEG-4-Formats für die Verwendung in Mobiltelefonen.

<http://www.3gp.se/>

API Als "Application Programming Interface" bezeichnet man die Programmierschnittstelle einer Programmbibliothek oder vom Betriebssystem. Ein Entwickler verwendet APIs, um so Zugriff auf angebotene Dienste oder Funktionen zu erhalten.

Assembly *Assemblies* sind das Standard-Format für Komponenten von .NET-Anwendungen. Es gibt sie als ausführbare Prozess-Assembly (.exe) oder Bibliotheks-Assembly (.dll)

[http://msdn2.microsoft.com/en-us/library/hk5f40ct\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/hk5f40ct(VS.71).aspx)

CLR Die *Common Language Runtime* ist die virtuelle Maschine, in der .NET-Anwendungen ausgeführt werden. Diese müssen als *Common Intermediate Language* (CIL) vorliegen. Während der Ausführung wird dieser Code *just-in-time* in nativen Code kompiliert.

<http://msdn2.microsoft.com/en-us/library/ddk909ch.aspx>

und <http://research.microsoft.com/~emeijer/Papers/CLR.pdf>

C# C# ist eine von Microsoft im Rahmen der Entwicklung des .NET-Frameworks entstandene Programmiersprache. C# ist prozedural und objektorientiert und wurde von der ISO standardisiert.

<http://msdn2.microsoft.com/en-us/vcsharp/aa336809.aspx>

DirectX DirectX ist eine Sammlung von [APIs](#) für Multimedia-Anwendungen. DirectX wird insbesondere bei der Spiele-Programmierung für Windows-Systeme verwendet.

<http://msdn2.microsoft.com/en-us/xna/aa937781.aspx>

.docx .docx ist die Datei-Endung des Office Open XML-Dateiformats, das von Microsoft Word 2007 zur Speicherung von Dokumenten verwendet wird.

<http://msdn2.microsoft.com/en-us/library/aa338205.aspx>

DOM Das *Document Object Model* ist eine sprachneutrale, objektbasierte Schnittstelle für Zugriff und Veränderung eines [XML](#)-basierten Dokuments.

<http://www.w3.org/DOM/>

ECMAScript Die von Ecma International standardisierte Fassung von JavaScript, einer objektorientierten Skriptsprache. Verbreitung erlangt hat JavaScript insbesondere durch die Verwendung in Web-Browsern, für die es ursprünglich auch entwickelt

wurde.

Für den Ecma-Standard, siehe <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

GUI GUI steht für “Graphical User Interface” und bezeichnet die grafische Bedienoberfläche eines interaktiven Programms. Diese Oberfläche bietet Steuerelemente wie Buttons oder Auswahllisten und kann direkt durch den Anwender manipuliert werden.

HTML Die Hypertext Markup Language ist eine hierarchische Auszeichnungssprache für Dokumente, die vor allem zur Erstellung von Webseiten verwendet wird. XHTML ist eine auf [XML](#)-Grundlage aufsetzende Variante von HTML, mit den entsprechenden Anforderungen an die Struktur des Dokuments.

JavaScript JavaScript ist die vorherrschende Implementierung von [ECMAScript](#) und war die Vorlage für den Standard. Es wird vor allem in Web-Browsern verwendet.

JSON JavaScript Object Notation (gesprochen wie der Name Jason) ist ein von Menschen und Rechnern lesbares, im Gegensatz zu XML kompaktes Datenaustausch- und -speicherformat. Implementierungen von JSON gibt es für eine Vielzahl von Programmiersprachen.
<http://www.json.org>

LINQ Language Integrated Query ist eine in .NET Version 3.5 hinzugekommene Technologie, die mit der Datenbank-Zugriffssprache SQL vergleichbare Zugriffe auf Datenstrukturen wie Collections und XML-Daten ermöglicht.
<http://msdn2.microsoft.com/netframework/aa904594.aspx>

Markup Eine *markup language*, zu deutsch Auszeichnungssprache, dient zur Anreicherung von Daten um Zusatzinformationen. bekannte Beispiele hierfür sind [XML](#) und [HTML](#).

Mono Mono ist eine Entwicklungs- und Laufzeitumgebung, die auf den standardisierten Teilen der .NET-Plattform basiert. Ziel ist es, eine Alternative zu .NET von Microsoft zu schaffen, die insbesondere auch die Ausführung von .NET-Software unter Linux ermöglicht.
<http://www.mono-project.com>

MP3 *MPEG-1 Audio Layer 3* ist ein verlustbehaftetes Format für die digitale, komprimierte Speicherung von Audiodaten. Es ist das wohl bekannteste Audioformat und ist von der ISO standardisiert.

MPEG-4 MPEG-4 ist ein Standard zur Komprimierung von Audio- und Video-Daten.
<http://en.wikipedia.org/wiki/MPEG-4>

.NET Klassenbibliothek und Laufzeitumgebung von Microsoft. Unterstützt eine Vielzahl von Programmiersprachen und bietet dem Entwickler zahlreiche Funktionen, um Desktop-, Web- und Kommandozeilenapplikationen zu entwickeln.

QuickTime QuickTime ist ein Framework für multimediale Daten von Apple. QuickTime kann als Container-Format für MPEG-4 verwendet werden.

<http://www.apple.com/quicktime/>

und <http://developer.apple.com/quicktime/>

Rich Internet Application Hierbei handelt es sich um Webanwendungen mit einer Benutzeroberfläche, deren Funktionalität und Verhalten dem von Desktop-Anwendungen entspricht; beispielsweise durch Unterstützung von Drag&Drop. RIAs trennen die Darstellung im Browser konsequent vom Server, der die Applikationslogik zentralisiert bereitstellt. Der Begriff entstammt einem Whitepaper von Macromedia (heute Adobe), deren Flash eine Umgebung für RIAs darstellt.

<http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf>.

SWF Das Dateiformat von Adobe Flash mit der Dateiendung *.swf*

VC-1, WMA und WMV Windows Media Video ist eine proprietäre Video-Codec-Familie von Microsoft. Die neueste Version 9 dieser Video-Codexs ist auch als VC-1 bekannt. Windows Media Audio ist ein proprietärer Audio-Codec von Microsoft.

<http://www.microsoft.com/windows/windowsmedia/forpros/codecs/codecs.aspx>

<http://www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx>

WYSIWYG Kurz für "What You See Is What You Get". Steht für Programmfunktionalität, bei der ein Dokument während der Bearbeitung bereits so dargestellt wird, wie es später ausgegeben wird. Beispiele hierfür sind Web-Editoren wie Adobe Dreamweaver und Textverarbeitungen wie Microsoft Word.

XML Die Extensible Markup Language ist eine Auszeichnungssprache. Ihr Zweck ist die Darstellung hierarchischer Daten. XML wird oft als Austauschformat zwischen verschiedenen Systemen verwendet. Unter anderem basieren XHTML für Websites und das in diesem Dokument beschriebene XAML auf XML.

<http://www.w3.org/XML/>

XPS Die XML Paper Specification ist ein XAML-basiertes Dateiformat für Dokumente mit fixem Layout. Es wird von Microsoft als Konkurrenzformat zu Adobe PDF positioniert.

<http://www.microsoft.com/whdc/xps/default.aspx>

Literatur

[Anderson 2007] Anderson, Chris: Essential Windows Presentation Foundation, 2007, Addison-Wesley, ISBN: 978-0321374479

[Cleeren 2006] Cleeren, Gill: Styles and Triggers in WPF, 2006
http://www.microsoft.com/belux/msdn/nl/community/columns/gillcleeren/wpf_stylesandtriggers.aspx

[Corby 2006] Corby, Karen: WPF Internet Sandbox Feature List (XBAPS & Loose XAML), 9. November 2006
<http://scorbs.com/2006/11/09/wpf-internet-sandbox-feature-list-xbaps-loose-xaml/>

[de Icaza 2007] Miguel de Icaza: Track Moonlight Status, 7. Dezember 2007
<http://tirania.org/blog/archive/2007/Dec-07.html>

[Ezell 2007] Ezell, Jesse: Silverlight vs. Flash: The Developer Story, 3. Mai 2007
<http://weblogs.asp.net/jezell/archive/2007/05/03/silverlight-vs-flash-the-developer-story.aspx>

[Guthrie 2007] Guthrie, Scott: Silverlight 1.0 Released and Silverlight for Linux Announced, 4. September 2007
<http://weblogs.asp.net/scottgu/archive/2007/09/04/silverlight-1-0-released-and-silverlight-for-linux-announced.aspx>

[Guthrie 2008] Guthrie, Scott: First Look at Silverlight 2, 22. Februar 2008
<http://weblogs.asp.net/scottgu/archive/2008/02/22/first-look-at-silverlight-2.aspx>

[Hamilton 2007] Hamilton, Kim: Linq Support in Silverlight 1.1 Alpha, BCL Team Blog, 2. Mai 2007
<http://blogs.msdn.com/bclteam/archive/2007/05/02/linq-support-in-silverlight-1-1-alpha-kim-hamilton.aspx>

[Moonlight] Moonlight Website
<http://www.mono-project.com/Moonlight>

[MSDN WPF] Windows Presentation Foundation, MSDN, Microsoft, Stand: 19. Dezember 2007
<http://msdn2.microsoft.com/en-us/library/ms754130.aspx>

[MSDN ClickOnce] Choosing a ClickOnce Update Strategy, MSDN, o.J.
<http://msdn2.microsoft.com/en-us/library/s22azw1e.aspx>

[MSDN Magazine 2006] MSDN Magazine: Build A Great User Experience With Windows Presentation Foundation, Oktober 2006
<http://msdn.microsoft.com/msdnmag/issues/06/10/appfundamentals/>

- [MSDN Silverlight] Silverlight Architecture Overview, MSDN, Microsoft, Stand: 10. Februar 2008,
<http://msdn2.microsoft.com/en-us/library/bb428859.aspx>
- [Petzold 2006] Petzold, Charles: Content Properties, 7. Februar 2006
<http://www.charlespetzold.com/blog/2006/02/070922.html>
- [Pfeil 2007] Pfeil, Christian: Silverlight vs. Flash - ein Vergleich TEIL 2, 30. August 2007
<http://www.christianpfeil.com/silverlight-vs-flash-ein-vergleich-teil-2/>
- [Silverlight] Microsoft Silverlight, Stand: 14. Dezember 2007,
<http://silverlight.net/>
- [Smith 2007] Smith, Josh: A Guided Tour of WPF, April 2007
<http://joshsmithonwpf.wordpress.com/a-guided-tour-of-wpf/>
- [Sneath 2007] Sneath, Tim: Silverlight 1.1 is now Silverlight 2.0, 29. November 2007
<http://blogs.msdn.com/tims/archive/2007/11.aspx>
- [Somasegar 2008] Somasegar, S.: 2008 Olympics brought to you by Silverlight, 7. Januar 2008
<http://blogs.msdn.com/somasegar/archive/2008/01/07/2008-olympics-brought-to-you-by-silverlight.aspx>
- [Van Patten 2007] Van Patten, Justin: Introducing Microsoft Silverlight 1.1 [alpha], BCL Team Blog, 30. April 2007
<http://blogs.msdn.com/bclteam/archive/2007/04/30/introducing-microsoft-silverlight-1-1-alpha-justin-van-patten.aspx>
- [WPF SDK 2006] Windows Presentation Foundation SDK Weblog: What is ContentPropertyAttribute? 21. Dezember 2006
<http://blogs.msdn.com/wpfsdk/archive/2006/12/21/what-is-contentpropertyattribute.aspx>
- [WPF SDK 2007] Windows Presentation Foundation SDK Blog: Using Themes with Custom Controls, 31. Juli 2007
<http://blogs.msdn.com/wpfsdk/archive/2007/07/31/using-themes-with-custom-controls.aspx>