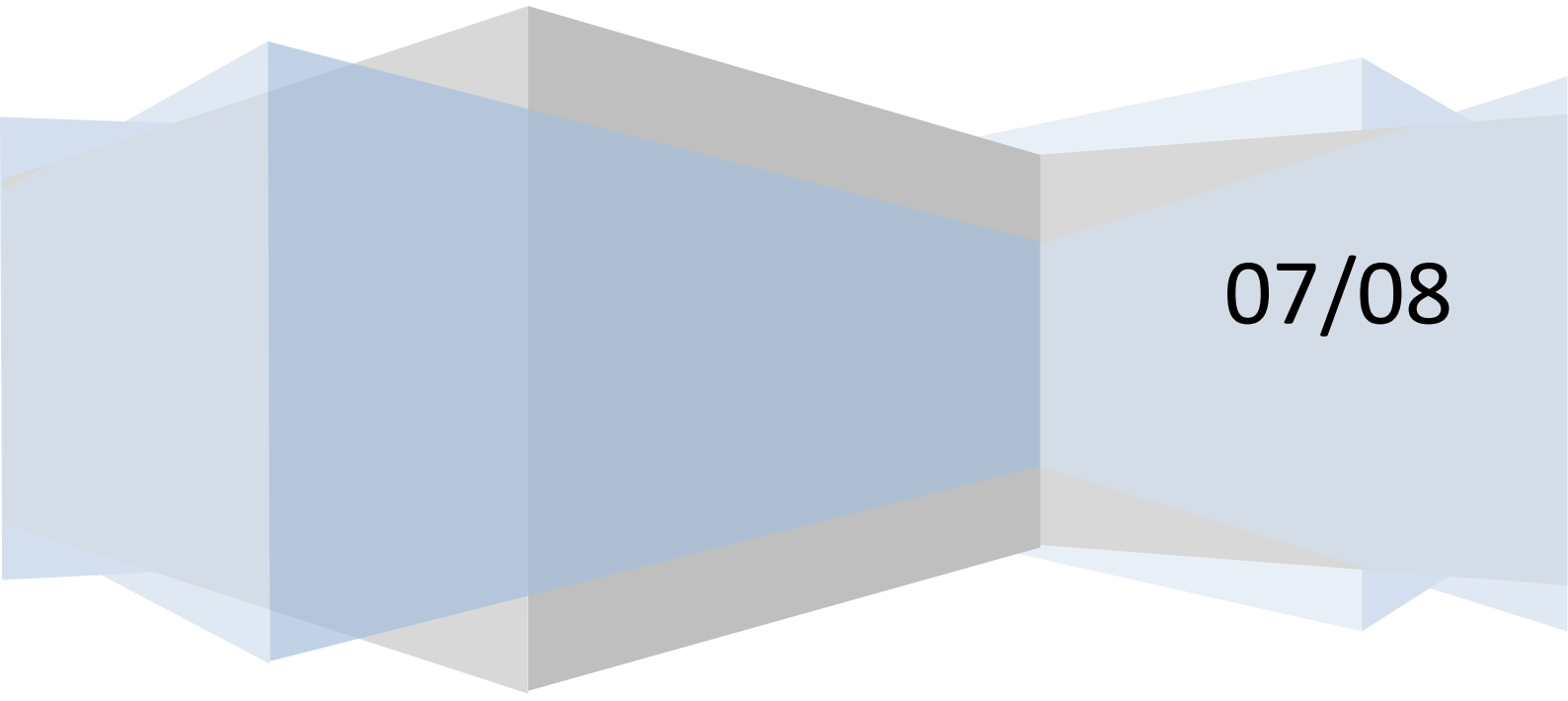
Hochschule Karlsruhe Technik & Wirtschaft

XAML – WPF per XML

Ausarbeitung

Manuel Naujoks



07/08

I. Abstrakt

Im Rahmen dieser Seminararbeit wird die Sprache XAML vorgestellt. Diese XML basierte Sprache dient in erster Linie der Beschreibung von WPF Oberflächen für .NET Anwendungen. Zusammen mit dem .NET Framework 3.0 wurde sie von Microsoft veröffentlicht.
Bevor auf die technischen Aspekte von XAML eingegangen wird, werden Vor- und Nachteile der Sprache gegenüber bekannten Beschreibungssprachen für Oberflächen erörtert. Dabei wird auch auf die Wichtigkeit von ansprechenden Oberflächen eingegangen.
Anschließend wird der Aufbau einer typischen XAML Datei untersucht und auf wichtige Elemente, wie zum Beispiel Namensräume eingegangen. Außerdem wird die Möglichkeit erläutert, wie beliebige Datenstrukturen beschrieben werden können. Nachdem die Struktur von XAML beschrieben wurde, wird der Kompilierungsprozess einer entsprechenden Datei erklärt. Dabei wird ein Schwerpunkt darauf gelegt, wo Oberflächenelemente in Code abgelegt werden und wie diese zur Laufzeit initialisiert werden können. Um das Aussehen der Oberfläche mit einer Logik zu versehen, gibt es in WPF spezielle Events. Es werden diese neuen Techniken sowie deren Anwendung mit XAML erläutert.

II. Inhaltsverzeichnis

1 Einführung

Das Thema Oberflächen beschäftigt die Entwickler seit es die ersten Anwendungen für den Computer gab. Am Anfang gab es beispielsweise nur eine Konsole. Im Lauf der Zeit hat sich auf diesem Gebiet dann aber viel getan, sodass Benutzerschnittstellen bis heute immer grafischer geworden sind. Der Gedanke der Oberfläche ist es, die Funktionalität einer Software so darzustellen, dass sie einfach gefunden und benutzt werden kann. Die Oberfläche ist es auch, die am ehesten mit dem Programm in Verbindung gebracht wird. Oberflächen nehmen also bei der Entwicklung von Programmen eine primäre Stellung ein, und sollten aus diesem Grund nicht vernachlässigt werden.
Um das zu gewährleisten, werden heute eigene Teams aufgestellt, die sich nur darum kümmern, wie eine Oberfläche auszusehen hat. Die Designer werden also immer mehr in die Entwicklung eines Programms mit einbezogen. Da hier aber zwei unterschiedliche Gruppen von Menschen, mit unterschiedlichen Einstellungen und Ansichten aufeinander treffen, ist die Zusammenarbeit meistens nicht trivial. Um dennoch einen Erfolg zu erzielen, ist eine Möglichkeit erforderlich, Designer und Entwickler optimal in den Arbeitsablauf bei der Softwareerstellung zu integrieren.
Das zu erreichen, hat sich die Firma Microsoft zum Ziel gesetzt. Die Lösung lautet XAML. Eine Beschreibungssprache für Oberflächen, die die Bedürfnisse der Entwickler und der Designer adressiert. Wie könnte es heute auch anders sein, als Sprachbasis kommt XML zum tragen. Mit dieser neuen Sprache von Microsoft ist es jedenfalls möglich, dass beide Parteien, Entwickler und Designer, besser zusammenarbeiten können.

In dieser Ausarbeitung geht es um diese neue Beschreibungssprache, XAML. Sie dient der Beschreibung von .NET Benutzerschnittstellen und hilft die .NET 3.0 Komponente WPF zu benutzen. Zunächst werden im Rahmen dieser Seminararbeit ein paar prinzipielle Gedanken zu dem Thema User Experience beschrieben. Daraus folgt dann eine Überlegung, was eine Markup Sprache für Oberflächen bieten muss, sodass sie wirklich benutzt werden kann und die Entwickler und Designer auch einen Nutzen daraus ziehen können. Es wird dabei gezeigt, dass es mehrere Varianten gibt, eine Oberfläche zu beschreiben.
Um zu zeigen, warum XAML, die neue Microsoft Sprache hier benutzt werden sollte, werden im späteren Verlauf einige der bekanntesten XML Sprachen auf dem Gebiet der Oberflächenbeschreibung vorgestellt sowie dessen Vor- und Nachteile beschrieben. Anschließend werden diese Sprachen grob mit XAML verglichen, indem die Vorteile der Microsoft Sprache, sowie deren Nachteile aufgezeigt werden.
Anschließend werden die technischen Aspekte von XAML erläutert. Es wird dabei die Technik hinter der Sprache im Vordergrund stehen. So wird der Aufbau von XAML mit seinen wichtigsten Elementen eingeführt, sowie ein Schwerpunkt auf das Thema Namensräume und den hierarchischen Aufbau gelegt. Die neue Sprache von Microsoft bietet hier einige interessante Möglichkeiten, sodass auch beliebige Objekt-Hierarchien problemlos beschrieben werden können.
Bei XAML handelt es sich um eine Sprache, die kompiliert, aber auch dynamisch zur Laufzeit geladen werden kann. Es wird im weiteren Verlauf der Ausarbeitung der Kompilierungsprozess, sowie die Verarbeitung von XAML zur Laufzeit vorgestellt. Dabei werden unteranderem Fragen, wie zum Beispiel, wo steht der Code, wo landen neue Steuerelemente, beantwortet.
Anschließend wird gezeigt, wie das Verhalten einer XAML Oberfläche implementiert werden kann. Dabei werden verschiedene Arten von Event Mechanismen eingeführt, die es erlauben auf das

Verhalten des Benutzers zu reagieren. Es wird ebenfalls gezeigt, wie eine Reaktion aussehen kann um solche Events abzufangen.
Die Quintessenz wird sein, dass mit XAML von Microsoft die Zusammenarbeit von Entwicklern und Designern wesentlich verbessert werden kann. Außerdem wird das Gestalten von Oberflächen leichter und komfortabler, sodass es keinen Grund gibt, Oberflächen für .NET mehr per C# oder VB.NET zu implementieren.

2 User Experience

Das Aussehen einer Anwendung, egal ob für Desktop oder Web, ist heutzutage sehr wichtig. Nur eine optisch ansprechende Benutzerschnittstelle findet bei der Mehrzahl aller Nutzer die erwünschte Akzeptanz. Obwohl sich Minderheiten an Eingabekonsolen gewöhnt haben und das Thema User Experience (UX) mit einem Lächeln verachten, ist es wichtig, Oberflächen so zu gestalten, dass die Schnittstellen zu den Funktionalitäten eine positive Wirkung auf die Benutzer haben. Was ist damit aber genau gemeint? Thomas Baekdal schreibt in seinem Blog eine sehr treffende Definition.

> *"User-experience is not like usability - it is about feelings. The aim here is to create happiness. You want people to feel happy before, during and after they have used your product."* [Tho06]

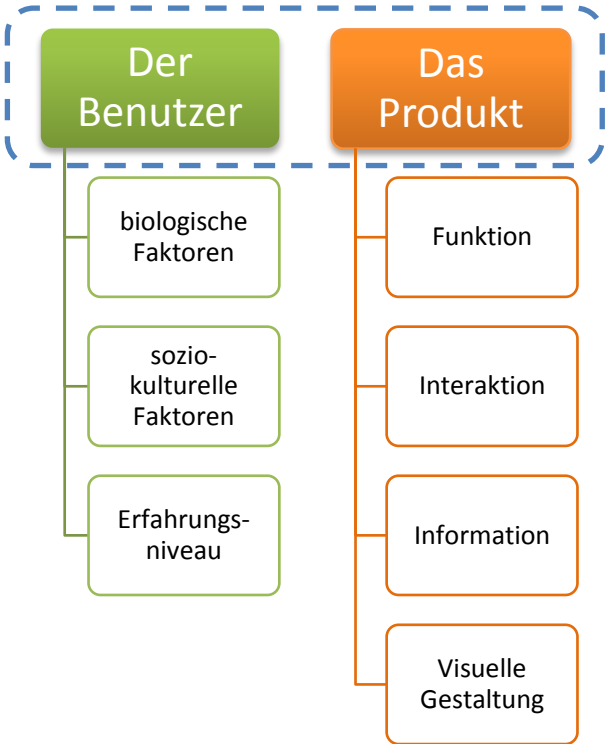


Abbildung 1 Konflikt der User Experience

Unterschiedlichste Nutzer müssen beispielsweise alle mit dem gleichen Programm arbeiten. Die Oberfläche dieses Programm muss dafür so gestaltet sein, dass sich alle Benutzer zurechtfinden können. Die Unterschiedlichkeit der Benutzer wird durch die Abbildung 1 veranschaulicht. Laut Clemens Lutsch, zusammengefasst von Andreas Dittes [And07] auf dessen Blog, liegt die User Experience irgendwo zwischen diesen beiden Punkten. Das Problem ist es, sie zu finden. Da man den Benutzer nicht direkt beeinflussen kann, liegt die gesamte Anstrengung der Designer und der Entwickler also darin, die neue Anwendung möglichst komfortabel zu gestalten. Das Programm kann noch so genial sein, die Benutzer sehen lediglich die Oberfläche. Unbewusst entwickeln die meisten Benutzer eine Sympathie für etwas das sie sehen und das trifft auch auf Benutzerschnittstellen zu. So kann eine optimal gestaltete Oberfläche dazu führen, dass es dem Benutzer Spaß macht, das Programm zu bedienen. Im Gegensatz kann eine schlechte Oberfläche den Benutzer nerven und ihm das Arbeiten unnötig erschweren.

Das Gestalten einer Oberfläche ist also gar nicht so einfach. Es gibt sogar Studien, die sich mit dem Thema User Experience beschäftigen. So hat zum Beispiel eine bahnbrechende Studie von IBM laut Andreas Dittes [And07] gezeigt, dass ein in die Findung der UX investierter Dollar das 10 bis 100 fache zurück bringt. Doch nicht nur IBM hat den Nutzen von ansprechenden Oberflächen erkannt. Auch Microsoft stellt mehrere Richtlinien zur Verfügung, die den Entwicklern und Designern helfen können, ihre Anwendungen an die User Experience der Windows Betriebssysteme anzupassen. In diesem Dokument von Microsoft [Mic072] werden verschiedene Modelle vorgestellt, wie Benutzer Oberflächen effektiv wahrnehmen und deren Inhalte einfacher lesen können. Ein weiteres Modell befasst sich beispielsweise mit der Effektivität der Platznutzung auf Oberflächen. Aus diesen Beispielen lässt sich schließen, dass UX wirklich nicht nur ein modisches Schlagwort ist, sondern auch von großen Firmen, wie zum Beispiel IBM und Microsoft, durchaus ernst genommen wird.

3 Anforderungen

Nachdem die Bedeutung der Oberflächengestaltung und der User Experience erläutert wurde, stellt sich die Frage, wie eine ansprechende Benutzeroberfläche überhaupt implementiert werden kann.
Um Oberflächen gestalten und programmieren zu können, gibt es heutzutage zwei Ansätze. Zum einen können Programmiersprachen Oberflächenbestandteile imperativ initialisieren und zusammensetzen. Die andere Möglichkeit besteht darin, die Oberfläche deklarativ zu beschreiben. Für diese Möglichkeit gibt es eigene, XML basierte Sprachen. Zu den Vorteilen der imperativen Vorgehensweise zählen unter anderem, dass das komplette Programm in einer Sprache implementiert werden kann und keine Einarbeitung in andere Techniken erforderlich ist.
Damit eine Oberflächenbeschreibungssprache zu einer echten Alternative gegenüber dem herkömmlichen Ansatz wird, muss Sie mehreren Anforderungen gerecht werden. Diese Anforderungen werden im Folgenden erläutert.

- Zum ersten muss sie einfach und schnell erlernbar sein. Die Sprache kann noch so umfangreich und mächtig sein, wenn sie deswegen aber zu kompliziert wird, wird sie auf wenig Begeisterung bei den Entwicklern und Designern stoßen und sich deswegen auch nicht durchsetzen können.

- Ein weiteres Kriterium ist die Ergebnisorientierung. Eine Sprache zur Oberflächenbeschreibung muss schnelle Ergebnisse liefern, sodass sich eine Umstellung von der imperativen auf die deklarative Programmierung lohnt. Die Verwendung einer solchen Sprache sollte gegenüber dem imperativen Ansatz schneller zu Ergebnissen führen.

- Zusätzlich muss die Sprache übersichtlich strukturiert sein, um ein späteres Verständnis der Realisierung zu gewährleisten. Beim Gestalten der Oberfläche sollte die Übersicht nie verloren gehen, sondern stets besser als bei der imperativen Programmierung sein.

- Des Weiteren muss die Beschreibungssprache einen großen Funktionsumfang bieten. Um das Aussehen einer Oberfläche vollständig mit einer Beschreibungssprache zu realisieren, muss sie die imperativen Möglichkeiten ersetzen können.

- Um den Funktionsumfang einer Sprache auch vollständig nutzen zu können, muss es für diese Beschreibungssprache eine geeignete Unterstützung durch Werkzeuge sowohl für Entwickler als auch für Designer geben.

- Da die Gestaltung der Oberfläche meistens nicht von Entwicklern, sondern von Designern vorgenommen wird, muss die Sprache die Zusammenarbeit dieser beiden Gruppen optimal unterstützen.

- Es muss also eine konsequente Trennung zwischen dem Aussehen und der Logik der Oberfläche erfolgen können. Dafür muss die Sprache geeignete Schnittstellen zur Verfügung stellen und einen reibungslosen Austausch von Information gewährleisten.

Im Rahmen dieser Seminararbeit wird die XML basierte Oberflächenbeschreibungssprache XAML (Extensible Application Markup Language) anhand dieser Kriterien analysiert.

4 Warum XAML?

Die Anforderungen an eine Markup Sprache für Oberflächen sind hoch. Dabei ist die Idee, Oberflächen für Desktop Anwendungen und das Web per XML deklarativ zu initialisieren, nicht neu. Es gibt sogar eine Vielzahl an Sprachen, die neben XAML alle auf diesem Gebiet angesiedelt sind. Dem zufolge stellt sich natürlich die Frage, welche Sprachen gibt es bereits und wieso gibt es noch eine neue Oberflächenbeschreibungssprache?

4.1 Beschreibungssprachen für Oberflächen

Im Folgenden werden einige der bekanntesten und am weitesten verbreiteten Sprachen vorgestellt, die für die Beschreibung von ansprechenden Benutzerschnittstellen verwendet werden. Dabei werden die Vor- und Nachteile von XUL, MXML, SwingML, XHTML, ZUML und LZX aufgezeigt. Natürlich gibt es noch jede Menge ähnlicher Sprachen, wie zum Beispiel AXUL, XULUX und SwixML, auf die hier aber nicht näher eingegangen werden kann, da der Umfang dieses Seminars begrenzt ist.
Von den sechs Sprachen, die im Folgenden betrachtet werden, sind die ersten drei Sprachen für Web und Desktop, die restlichen drei nur für das Internet verwendbar. Dabei ist diese gruppierte Auflistung nach der Informationsvielfalt im Internet geordnet. Außerdem sei zu beachten, dass es sich hier um XML basierte Sprachen handelt.

4.1.1 XUL

Eine der bekanntesten Markup-Sprachen für Oberflächen ist XUL (XML User Interface Language). In seinem Tutorial auf [Uch07] schreibt Uche Ogbuji, das die Sprache im Rahmen des Mozilla Projekts entwickelt wurde und der Beschreibung von Oberflächen für plattformunabhängige Anwendungen dient. Als populäres Beispiel führt Ogbuji die Anwendung Firefox an, die mit XUL erstellt wurde.
Im Weiteren schreibt er eine Einführung über die Mozilla Sprache. Dabei setzt XUL für die Gestaltung des Aussehens einer Oberfläche auf den bestehenden Web-Standard CSS und ermöglicht dadurch eine Trennung von Layout und Aussehen, sowie einfaches Austauschen von Skins. Das Mozilla Projekt schreibt auf ihrer Entwicklerseite [Moz07] , dass XUL zur Laufzeit durch die „Gecko Rendering Engine“ interpretiert wird. Folglich ermöglicht dies, das Aussehen der Anwendung auch dynamisch zu verändern. Daraus schließt sich, dass der Umfang von XUL auf Kosten der Bedienung geht, da die

Oberfläche langsamer ist als vergleichbare kompilierte Oberflächen. Außerdem ist es dann wahrscheinlich, dass sich das Verhalten der Benutzerschnittstelle auch von anderen Anwendungen unterscheiden kann. Da XUL von der Gecko Engine gerendert wird, lässt darauf schließen, dass neben Oberflächen für Desktop Anwendungen auch Oberflächen für das Web beschrieben werden können, die allerdings nur in einem unterstützenden Browser angezeigt werden können.

4.1.2 MXML

Eine weitere Beschreibungssprache ist MXML (MX Markup Language). Sie wurde von Macromedia, jetzt Adobe, entwickelt und übernimmt bei der Entwicklung von flashbasierten Anwendungen die Gestaltung von Oberflächen. Das schreibt Christophe Coenraets von Adobe in seiner Übersicht [Chr04] über die Sprache. Er schreibt, MXML unterstütze CSS und sei relativ vielseitig, sodass nicht nur grafische Benutzerelemente beschrieben werden können, sondern auch andere logische Zusammenhänge. Anders als XUL wird MXML nicht interpretiert, sondern in ActionScript, dem imperativen Gegenstück der Beschreibungssprache, kompiliert. Zusammen mit ActionScript, lassen sich mit MXML moderne Flash und Flex Anwendungen beschreiben. Dies lässt die Schlussfolgerung zu, dass der Entwickler bei der Benutzung dieser Beschreibungssprache an ActionScript gebunden ist und ohne erheblichen Mehraufwand nicht auf Funktionalität aus anderen Sprachen zurückgreifen kann.

4.1.3 SwingML

Auch im Bereich der Java Entwicklung gibt es eine Beschreibungssprache für Oberflächen. Auf der Seite des SwingML Projekts [Swi07] , wird die Swing Markup Sprache vorgestellt. Sie dient der Beschreibung von Swing Elementen für Desktop Oberflächen, sowie für Applets. Dabei wird ein SwingML Renderer verwendet, der eine entsprechende XML Datei in eine grafische Benutzeroberfläche übersetzt. In einem Tutorial des Projekts [Swi071] wird das Prinzip dieser Sprache vorgestellt. Nach der dortigen Information kann der Renderer auch mit einem dynamisch erstellten XML Text umgehen und so auch auf entsprechende Webinhalte zugreifen, die dann eine Oberfläche auf dem Client erzeugen. Dabei ist es mit SwingML möglich, Events komplett in XML zu implementieren. Das ist zwar hilfreich, entspricht aber nicht dem Gedanken, die Logik der Oberfläche von dessen Aussehen zu trennen. Einem Bericht des Projektteams über die Erstellung von eigenen Events [Swi072] kann entnommen werden, dass es grundsätzlich zwar möglich ist, auf Java Code zu referenzieren, der dann in Rahmen eines Events ausgeführt wird, dies allerdings mit relativ hohem Aufwand verbunden ist. Dies ist nämlich nur möglich, indem man bestimmte Schnittstellen auf Klassen Ebene implementiert. Daraus lässt sich schließen, dass der Weg zurück, also vom Code zu den Oberflächen Elementen, erheblich problematischer wird. Laut diesem Bericht [Swi072] kann einem solchen EventHandler nämlich nur eine Referenz auf ein einziges Element übergeben werden. Die Ermittlung aller weiteren Elemente der Swing Oberfläche ist dann nicht mehr trivial, sodass ein umfangreicheres Verhalten der Oberfläche nicht in realistischem Aufwand in Java implementiert werden kann. SwingML wird nämlich nicht in Java Klassen kompiliert, sondern, wie bereits erwähnt, von dem im Framework enthaltenen Renderer interpretiert. Dateien, die in diesem Format geschrieben wurden, können also auch nicht in bestehende Anwendungen eingebettet werden, da sie nur über den Aufruf des SwingML Renderers zur Oberfläche werden.

4.1.4 XHTML

Für das Beschreiben von Oberflächen im Web wird häufig XHTML (Extensible Hypertext Markup Language) als Beschreibungssprache verwendet. Die XHTML Working Group [XHT07] des W3C beschreibt XHTML als einen HTML-Dialekt mit XML konformer Syntax. Dabei setzt die Sprache bei der

Beschreibung von Oberflächen genau wie XUL auf CSS. Dadurch kann eine klare Trennung zwischen dem Layout und dem Aussehen einer Oberfläche erreicht werden.
Daraus lässt sich schließen, dass die Beschreibung von Oberflächen für Client Anwendungen auf dem Desktop mit dieser Sprache ohne weiteres nicht möglich ist, da sie genau wie HTML nicht kompiliert, sondern von Browsern interpretiert werden muss. Das schränkt die Verwendung der Sprache erheblich ein und macht sie für Entwickler weniger interessant als für Webdesigner.

4.1.5 ZUML

Auch bei ZUML (ZK User Interface Markup Language) handelt es sich um eine Beschreibungssprache für Oberflächen im Web. ZUML ist die Sprache des ZK Frameworks, einem AJAX Web-Anwendung Framework. In seinem Artikel zum Thema AJAX mit ZK [Klabe] schreibt Michael Klaene eine Übersicht und eine Einführung zu ZUML. Laut ihm können mit dieser Beschreibungssprache XHTML- und XUL-Elemente vermischt werden, um die Vorteile der beiden Sprachen zu vereinen. Das ZUML nicht durch die Gecko Rendering Engine interpretiert wird, lässt darauf schließen, dass im Rahmen des ZK Frameworks die XUL Elemente in HTML oder XHTML Elemente übersetzt werden. Aus diesem Grund können nicht alle nativen XUL Element sofort unterstützt werden, da sie im Rahmen des ZK Frameworks implementiert werden müssen. Das wiederum führt dazu, dass durch ZUML beschriebene Elemente ein individuelles Aussehen haben.
Da das ZK Framework nur auf das Web ausgelegt ist und laut der Seite des Frameworks [Pot07] für ZUML Kenntnisse in Java, XHTML und XUL erforderlich sind, eignet sich diese Beschreibungssprache nicht für die universelle Beschreibung von Oberflächen.

4.1.6 LZX

LZX (Laszlo XML) ist die XML basierte Beschreibungssprache der OpenLaszlo Platform. Laut der mangelhaft übersetzten Webseite des Projekts [Ope06] wird LZX zunächst an einen Server geschickt, der eine Art Servlet hosted, dass die Datei dann in eine Flash Anwendung kompilieren. Da diese Übersetzung erst bei Anfrage an die LZX Datei erfolgt, begrenzt sich die Verwendung der Sprache auf das Internet.
In dem Kurs „AJAX - Frische Ansätze für das Web-Design“ [TEIbe] der TEIA AG werden Möglichkeiten der Benutzung von LZX beschrieben. Zum Beispiel kann mit dieser Sprache im Gegensatz zu vielen anderen Beschreibungssprachen laut der TEIA AG objektorientiert beschrieben werden. So lassen sich Klassen definieren, dessen Verhalten über Objektmethoden in JavaScript programmiert werden kann. Daraus folgt, dass sich aufgrund dieser Möglichkeit sehr umfangreiche Anwendungen mit LZX realisieren lassen, die aber schnell unübersichtlich werden. So ist es laut der TEIA AG [TEIbe] nicht möglich Layout, Aussehen und Verhalten der Anwendung zu trennen, was schnell zu einem unübersichtlichen und schwer zu verstehenden Programmcode führt.

4.2 Vorteile von XAML

Obwohl es schon diese Beschreibungssprachen für Oberflächen gibt, hat Microsoft etwas Neues entwickelt. Zusammen mit dem .NET Framework 3.0 ist XAML dann als XML basierte Sprache veröffentlicht worden, um Oberflächen mit der neuen Oberflächen API, Windows Presentation Foundation (WPF) zu beschreiben, so Microsoft [Micbe] .
Es gibt zwar Ansätze die Idee von XAML schon mit .NET 1.1, 2.0 und ASP.NET zu benutzen, das MyXaml Projekt [Marbe] behauptet aber sogar von sich selbst, unterschiedlich zu Microsofts XAML zu sein.

> *„Is It The Same As Longhorn's Markup? No. There are similarities because both use XML, but they are different."* [Marbe1]

Ob MyXaml auch für .NET Versionen höher als 2.0 verwendet werden kann, war aus den Informationen auf der Webseite des Projekts übrigens nicht zu entnehmen. Aus diesem Grund wäre ein detailierter Vergleich aufgrund der unterschiedlichen .NET Frameworks nicht sinnvoll.

Wie schon erwähnt, wurde XAML von Microsoft entwickelt. Das Software Unternehmen aus Redmond ist ja dafür bekannt, es mit bestehenden Standards nicht so genau zu nehmen und deswegen gibt es auch mit XAML einen neuen offenen Standard. Man kann sich natürlich darüber streiten, wieso Microsoft schon wieder etwas Eigenes gemacht hat und nicht auf Standards des W3C oder anderer Organisationen setzt. Auf dem Gebiet der Oberflächenbeschreibung mit XML basierten Sprachen gibt es ja schon viele Sprachen, die alle mehr oder weniger dasselbe Ziel wie XAML haben. Wie oben beschreiben, haben alle dieser vielen Sprachen Stärken und Schwächen. Natürlich ist das bei XAML auch so, aber Microsoft hat bei der Entwicklung ganz besonders Wert auf die Akzeptanz bei Entwicklern und vor allem Designern gelegt und versucht zu ermöglichen, das diese beiden Gruppen besser zusammen arbeiten können. Das lässt sich aus den neuen Produkten der Redmonder, dem Expression Studio [Mic07] und dem neuen Visual Studio 2008 [Micbe1] , schließen.

In der MSDN Library wird XAML beschrieben. Dort wird beschrieben [Micbe2] , dass die Microsoft Sprache in der Regel kompiliert wird. Im Gegensatz zu vielen anderen Sprachen auf dem Gebiet der Oberflächenbeschreibung, die nur zur Laufzeit interpretiert werden können, ist XAML dank der Kompilierung wesentlich flexibler verwendbar. Die Kommunikation mit dem eigentlichen Code der Anwendung wird durch wesentlich einfacher als bei anderen vergleichbaren Sprachen. Desweiteren bleibt eine mit dieser Sprache beschriebene Oberfläche höchst performant, da keine Interpretation notwendig ist. Stattdessen existiert zu jeder XAML Datei eine Code-Behind Klasse und eine Generated Code Klasse, aber darauf werden in den Unterabschnitten 5.2 XAML & Code und 5.3 Kompilierung näher eingegangen.

Es gibt hier eine Ausnahme, sodass bei der Verwendung von XAML Inhalt in Silverlight [Lau07] die Oberfläche nur interpretiert werden kann. Es lässt sich daraus schließen, dass dies notwendig ist, damit Silverlight nicht den gleichen Nachteil wie MXML und Flash hat. XAML wird nämlich in diesem Fall nicht in ein Binärformat kompiliert, sondern bleibt XML, das von Browsern mit dem Silverlight Plugin angezeigt werden kann. Das macht es für Suchmaschinenagenten möglich diese Seiten durchzuparsen, was bei dem binären Flash nicht möglich ist.
Bestehende XAML Inhalte können in Silverlight also wiederverwendet werden. Lediglich sehr WPF-lastige Oberflächen müssen minimal angepasst werden, da nicht alle Möglichkeiten von WPF auch in Silverlight vorhanden sind. So besitzt Silverlight zum Beispiel kein eigenes 3D Modell [Dir07] . Daraus folgt, dass abgesehen von den spezifischen Elementen, die Beschreibungssprache dafür benutzt werden kann, eine .NET Oberfläche ohne viel Aufwand in eine Silverlight Oberfläche umzugestalten. Dafür muss die XAML Datei einfach portiert werden. Diese Möglichkeit macht XAML zu einer sehr vielseitigen Beschreibungssprache, die sogar in der Linux Welt Verwendung findet, denn auch die Silverlight Variante Moonlight [Monbe] kann mit XAML verwendet werden.

Ein weiterer Vorteil von XAML besteht in der sehr guten Unterstützung durch Werkzeuge. Sowohl für Entwickler, als auch für Designer gibt es freie und kommerzielle Tools, die das Erstellen von XAML

Code sehr vereinfachen. Natürlich kann eine XAML Datei auch mit einem beliebigen Texteditor erstellt werden, doch bieten speziell dafür konzipierte Werkzeuge viel mehr Möglichkeiten. Ein kostenfreies Tool wird mit dem .NET Framework 3.0 gleich mit installiert. Das XAMLPad eignet sich zum schnellen Testen einer XAML Datei, kann den Designer aber beim Erstellen großer Oberflächen nicht sonderlich unterstützen. Stattdessen haben Designer die Möglichkeit Oberflächen mit dem kommerziellen Programm Microsoft Expression Blend [Mic07] zu erzeugen. Blend ist Bestandteil des Expression Studios von Microsoft und adressiert genau die Bedürfnisse der Anwender. So lassen sich mit Blend auf einfache Weise sehr umfangreiche Animationen erstellen, die beispielsweise mit dem XAMLPad nur mit wesentlich mehr Aufwand machbar wären.
Zu dem arbeitet Blend fast nahtlos mit dem Microsoft Visual Studio [Micbe1] zusammen. Visual Studio 9 (2008) unterstützt die Entwicklung von XAML Oberflächen nativ, bei VS 8 (2005) müssen die „.NET Framework 3.0 Development Tools“ nachträglich installiert werden. Diese Sachverhalte wurden bei der Evaluierung der beiden Programme festgestellt. Visual Studio ist das geeignete Programm für Entwickler. Im Gegensatz zu Blend bietet es in Bezug auf XAML eine umfangreichere IntelliSense und eignet sich daher zum Beispiel besser für die Definition von Events. Sämtlicher Code, der das Verhalten und die Logik der Oberfläche definiert, sollte also mit dem Visual Studio geschrieben werden.

XAML ist mit dem .NET Framework 3.0 erschienen. Mittlerweite ist bereits die nächste Version des Frameworks (3.5) zusammen mit dem Visual Studio 2008 veröffentlicht worden [Micbe3] . Dabei wird .NET immer erweitert und immer umfangreicher. Natürlich kann man darüber diskutieren, ob eine so schnelle Entwicklung vielleicht sogar zu schnell erfolgt. Wenn man aber die stark steigende Zahl der Möglichkeiten der Technologie von Microsoft in Betracht zieht, dann wird man feststellen, dass sich keine andere Sprache oder anderes Framework mehr für die Entwicklung von Windows Anwendungen eignet. Der Klassen Umfang des .NET Frameworks nimmt immer mehr zu [Mic071] und eröffnet den Entwicklern damit immer mehr Möglichkeiten und hilft, selbst komplexe Probleme elegant zu lösen.
XAML, als Beschreibungssprache wird benutzt, um Klassen deklarativ zu instanziieren. Bei den Klassen muss es sich dabei nicht einmal um Oberflächen Elemente handeln. XAML kann mit beliebigen, sogar eigenen Klassen arbeiten [Micbe4] . Darauf wird übrigens im Unter-Unterabschnitt 5.1.2 Eigene Objektstrukturen dieser Ausarbeitung näher eingegangen. Diese Möglichkeit gibt XAML alle Vorteile, die auch .NET besitzt.

Zu diesen Vorteilen gehört auch eine gewisse Sprachunabhängigkeit. So kann das Verhalten von XAML Oberflächen theoretisch in einer beliebigen .NET Sprache erfolgen, da allen Sprachen das gleiche Framework zugrunde liegt. Im Prinzip muss diese Sprache nur das „partial“ Schlüsselwort unterstützen, um die Code Behind Klasse mit der Klasse aus der Generated Code Datei verbinden zu können [Micbe2] . Es gibt auch eine Möglichkeit XAML zu nutzen, wenn „partial“ in der verwendeten Sprache nicht vorhanden ist, wie es zum Beispiel bei F# der Fall ist. Robert Pickering demonstriert das auf seinem Blog [Rob06] , aber auf diesen Aspekt wird im Rahmen dieser Ausarbeitung später nur teilweise näher eingegangen. Auch eigene XAML Objekt Hierarchien [Micbe4] können so sprachunabhängig deklariert werden. Eine einzige solche XAML Datei kann zum Beispiel in verschiedenen Anwendungen, die alle in unterschiedlichen .NET Sprachen implementiert sind, eingesetzt werden, ohne dass eine einzige Zeile Code angepasst werden muss. Konkret wird diese Thematik im Unterabschnitt 5.2 XAML & Code auf Seite 19 behandelt.

4.2.1 eFace

Es gibt inzwischen sogar eine XAML Variante für Java. Das eFace Presentation Framework hat eine, nach eigenen Angaben [Soy07] , kompatible Plattform zu WPF und XAML entwickelt, allerdings aus 100% Java.

> *„The core of eFace, code-named as UPF (Universal Presentation Framework), remains compatible with Microsoft's WPF (Windows Presentation Foundation). It is a 100% Java solution."* [Soy07]

Soyatec schreibt weiter, dass für die Darstellung in eFace verschiedene Renderer zur Verfügung gestellt werden, unteranderem für SWT und Swing. Dadurch kann man mit XAML jetzt auch Oberflächen für alle Betriebssystem beschreiben, für die es das Java Runtime Environment gibt. Leider muss man in diesem Fall die vielen Vorteile aufgeben, die man mit dem .NET Framework von Microsoft hätte. Zum Beispiel ist es mit eFace momentan noch nicht möglich Objekte der Oberfläche in 3D darzustellen, wie aus einem Eintrag von Yves Yang aus einem Forum auf der Soyatec Seite entnommen werden kann.

> *"The graphic animation and document presentation are two issues we will work later, probably in 23 2008. The plan can be changed if you can order this feature with some financial support."* [Soy071]

Microsofts WPF bietet in Verbindung mit XAML also noch die meisten Möglichkeiten für die Oberflächenbeschreibung.

4.3 Nachteile von XAML

Das XAML unteranderem durch das .NET Framework zu einer sehr mächtigen und umfangreichen Sprache wird, hat leider nicht nur Positives. Wie jede andere Sprache, besitzt auch die Sprache von Microsoft Schwächen.

XAML besitzt nicht nur die Vorteile von .NET, sondern zwangsläufig auch dessen Nachteile. So lässt sich für die Implementierung der Logik nur eine .NET Sprache verwenden. Bei der Verwendung von Silverlight 1.0 gibt es hier allerdings wieder eine Ausnahme, sodass auch JavaScript verwendet werden kann [Lau07] . Wenn man aber eine Oberfläche für Windows mit XAML gestalten will, ist man bei der Business Logik an .NET gebunden.

Das führt dazu, dass die entstehende Anwendung nicht wirklich plattformunabhängig ist. Man könnte mit eFace XAML auch mit Java implementieren [Soy07] , allerdings müsste man dann, wie bereits erwähnt, auf die vielen Vorteile von .NET verzichten. Wie bereits erwähnt, müsste man dann auf die Möglichkeit von 3D auf Oberflächen verzichten, da dies in Java und in eFace noch nicht unterstützt wird [Soy071] .

Wenn man aufgrund von dieser fehlenden Funktionalität in Java dann doch .NET benutzen möchte, könnte eine .NET Anwendung auch für das Mono Framework kompiliert werden. Allerdings ist es dann sehr wahrscheinlich, dass tiefergehende Änderungen am Quellcode, der für das Verhalten der Oberfläche zuständig ist, vorgenommen werden müssen. Im Rahmen des Olive Projekts, als Teil von Mono, gibt es zwar ein Presentation Framework [Monbe2] , inwiefern dieses Modell aber mit WPF vergleichbar ist, konnte im Rahmen dieser Ausarbeitung nicht festgestellt werden. Auch konnte nicht ermittelt werden, ob Olive die Möglichkeit der 3D Darstellung auf Oberflächen erlaubt. Was XAML betrifft, so gibt es einen XAML Compiler für Mono, der im Rahmen des „Summer of Code" Projekts

von Google 2005 vom Entwickler Iain McCoy entwickelt wurde [Monbe1] . Allerdings wird hier im Gegensatz zu eFace keine vollständige Kompaibilität mit dem Microsoft XAML zugesichert [Soy07] .

Ein weiterer Nachteil von XAML ist, dass für das Aussehen kein CSS, sondern eine Microsoft eigene Technologie benutzt wird, die in der MSDN Library [Micbe5] beschrieben wird. Designer, die sich mit dem W3C Standard gut auskennen, müssen folglich eine neue aber ähnliche Form der Beschreibung erlernen, wenn sie mit XAML Oberflächen gestalten wollen. Das ist nicht praktisch, da CSS bereits weit verbreitet ist und sogar zu einem W3C Standard geworden ist. Vermutlich ist die Entscheidung von Microsoft auf folgenden Aspekt zurückzuführen. Für Entwickler ist die Tatsache, dass es kein CSS gibt, dagegen kein Nachteil. Das Aussehen auf .NET Framework Klassen zurückzuführen, die sich durch .NET Code leichter zur Laufzeit manipulieren lassen können, als es mit CSS möglich gewesen wäre, ist für Entwickler selbstverständlich. An dieser Stelle ist XAML auf Entwickler zugeschnitten, Designer haben davon aber einen Nachteil.

5 XAML

Nachdem nun die Vorteile und die Nachteile von XAML als Beschreibungssprache für Oberflächen erörtert wurden, werden nun die technischen Aspekte der Markup Sprache von Microsoft behandelt. Diesbezüglich wird zunächst der Aufbau einer typischen XAML Datei analysiert.

5.1 Aufbau

Wie bereits erwähnt wurde, benutzt XAML die XML Syntax. Das hat zur Folge, dass die beschriebene Struktur eine Hierarchie darstellt, so Christian Pape in seiner Vorlesung [Pap06] .

5.1.1 Syntax

Dabei besteht eine XAML Datei aus Elementen, Attributen und Namensräumen. Eine „leere“ Datei könnte folgendermaßen aussehen.

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
    </Grid>
</Window>
```

5.1.1.1 Elemente

Eine gültige XML Datei besitzt genau ein Root Element. Dieses Root Element enthält dann weitere Elemente. Abgesehen vom Root Element, kann jedes Element nur ein übergeordnetes Vater Element besitzen. Andererseits kann es grundsätzlich selbst Vater Element für beliebig viele Unterelement sein. In XAML gibt es Einschränkungen, sodass bestimmte Elemente nur eine definierte Anzahl an Unterelementen enthalten können. Window und Button sind zum Beispiel solche Elemente, aber dazu mehr im Unter-Unterabschnitt 5.1.3 Verschachtelbarkeit.
Im obigen Beispiel besteht die XAML Datei aus einem Root Element vom Typ Window, also einem Fenster. Es gibt in WPF mehrere Möglichkeiten das Root Element zu wählen. Ein Fenster steht dabei für ein komplettes Fenster. Über ein anderes Root Element, wie zum Beispiel das Element Page, könnte eine seitenbasierte Navigationsstruktur beschreiben werden. Im Prinzip könnte das Root

Element aber auch ein einfaches Button Element sein. Ein Element entspricht in XAML immer einer WPF Klasse, sodass mit einem konkreten Tag eine Instanz dieser WPF Klasse erzeugt wird. Dabei wird der Standardkonstruktor der Klasse aufgerufen. Eine Auswahl der Konstruktor ist nicht möglich.

5.1.1.2 Attribute

Wie in XML können auch in XAML die Eigenschaften von Elementen über Attribute gesetzt werden. Dabei werden die Eigenschaften der entsprechenden WPF Klasse über öffentliche Properties gesetzt. Dabei müssen diese Eigenschaften Wertetypen darstellen. Für andere Typen muss laut Dirk Frischalowski [Unk07] ein Typkonverter bereitgestellt werden. In dieser Ausarbeitung wird dieses Thema aber nicht weiter behandelt.

5.1.1.3 Namensräume

Damit mit XAML Objekt Hierarchien beschreiben werden können, müssen die zur Verfügung stehenden Elemente bekannt gemacht werden. Dies wird wie in XML auch, über Namensräume ermöglicht. Die Namensräume werden standardmäßig im Root Element der XAML Datei definiert.

- http://schemas.microsoft.com/winfx/2006/xaml/presentation

Üblicherweise wird dieser Namensraum immer definiert, um die grundlegenden WPF Klassen zur Verfügung zu stellen.

- http://schemas.microsoft.com/winfx/2006/xaml

Dieser weitere Namensraum wird standardmäßig auch definiert und macht, wie Dirk Frischalowski in seinem Tutorial über WPF schreibt [Unk07] , den .NET Namespace System.Windows.Markup verfügbar. Dadurch lassen sich grundlegende XAML Funktionen nutzen. Wie im obigen Beispiel wird dieser Namespace meist mit dem Präfix x verbunden.

- Eigene Namensräume

Wie bereits erwähnt kann mit XAML eine beliebige Objekt Hierarchie instanziiert werden. Dafür muss allerdings ein eigener Namensraum definiert werden können. Solche eigenen Namensräume werden laut Dirk Frischalowski [Unk07] über eine spezielle Syntax beschrieben. Im unteren Beispiel wäre der eigene Namespace MyNamespace und würde aus der Assembly MyApp kommen. Ist der Namensraum in der gleichen Assembly wie die XAML Datei, kann die Assembly Information auch weggelassen werden.

```
xmlns:myN="clr-namespace:MyNamespace;assembly=MyApp"
```

5.1.2 Eigene Objektstrukturen

Mit dieser Syntax können jetzt beliebige Klassen in XAML instanziiert werden. Ob das immer so sinnvoll ist, kann allgemeingültig nicht gesagt werden. Aber die Möglichkeit besteht und das spricht für eine flexible Anwendung der Microsoft Markup Sprache. Jetzt wird auch deutlich, dass alles was in XAML gemacht werden kann, auch in .NET Code gemacht werden kann. Der Vorteil einer deklarativen Beschreibung durch die Microsoft Beschreibungssprache ist, dass auch ein entsprechendes Tool oder ein XSLT eine XAML konforme XML Datei erzeugen kann. Hier entsteht

also eine Vielzahl an neuen Möglichkeiten, die auch automatisierbar genutzt werden können. Code für eine imperative Instanziierung zu erzeugen wäre wahrscheinlich aus Sicht eines Tools wesentlich aufwendiger.

```
namespace MyBusinessLogik
{
    public class Person
    {
        private string name;
        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        public override string ToString()
        {
            return this.name;
        }
    }
}
```

Abbildung 2 Fenster aus der XAML Datei mit eigenem Objekt

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:myA="clr-namespace:MyBusinessLogik"
    Title="Window1" Height="250" Width="249">
    <Grid>
        <Button>
            <myA:Person Name="Bill">
            </myA:Person>
        </Button>
    </Grid>
</Window>
```

Im obigen Beispiel wird dem Button ein Objekt vom Typ Person zugewiesen. Da Person kein WPF Element ist, wird die ToString() Methode des Objekts aufgerufen um dessen Inhalt darzustellen. Näheres zu dieser Thematik wird im Unter-Unterabschnitt 5.1.3 Verschachtelbarkeit beschrieben.
Um noch einmal auf die Einfachheit der Instanziierung einer Klasse mit XAML hinzuweisen, wurde im folgenden Beispiel das gleiche Fenster wie im obigen Beispiel instanziiert, allerdings mit .NET Code.

```
Window window = new Window();
window.Title = "Window1";
window.Height = 250;
window.Width = 249;
Grid grid = new Grid();
window.Content = grid;
Button button = new Button();
Person person = new Person();
person.Name = "Bill";
button.Content = person;
grid.Children.Add(button);
```

Hier sieht man auf den ersten Blick keine Struktur, was für einen weiteren Vorteil von XAML spricht. Die Objekt Hierarchie bleibt im Vergleich zu gleichwertigem .NET Code relativ übersichtlich.

5.1.3 Verschachtelbarkeit

XAML benutzt die XML Syntax um eine hierarchische Struktur zu beschreiben. Dabei können genau wie in XML auch, Elemente geschachtelt werden. Das allgemeine Prinzip der Verschachtelbarkeit wird aus den folgenden Beispielen ersichtlich.
Zunächst wird ein XAML Fenster, das ein Grid Panel enthält, betrachtet. Dieses Grid Panel enthält wiederrum ein Button Element.

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
        <Button Height="100" Width="200" Name="myButton">
            Hallo Welt!
        </Button>
    </Grid>
</Window>
```

Wie bereits erwähnt kann ein Window Element nur ein einziges Unterelement enthalten. In der MSDN Library ist die Ursache dafür klar ersichtlich [Mic073] . Die System.Windows.Window Klasse erbt von der Klasse System.Windows.Controls.ContentControl.
„ContentControl“ enthält laut MSDN Library die Eigenschaft „Content“ vom Typ Objekt [Mic074] . Das bedeutet, dass alle Elemente, die von dieser Klasse erben, prinzipiell Vaterelement für ein beliebiges anderes Element sein können. Wird die Content Eigenschaft dabei auf ein Element gesetzt, dass von System.Windows.UIElement erbt, wird dieses Element auf der Oberfläche dargestellt. Wird ein anderes Objekt gesetzt, wird die ToString() Methode des Objekts aufgerufen und der daraus resultierende Text dargestellt.

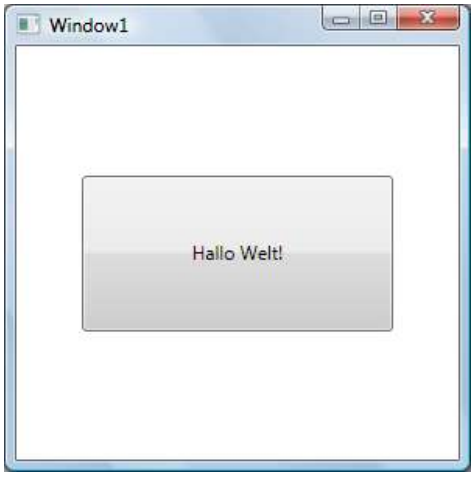


Abbildung 3 Fenster aus der einfach verschachtelten XAML Datei

Die Klassen Window und Button erben die Eigenschaft ein anderes Element zu beinhalten also von der Klasse ContentControl. Doch andere Elemente, wie zum Beispiel das Grid Panel, können mehrere Elemente enthalten. Ein Blick in die MSDN Library lüftet auch dieses Geheimnis. System.Windows.Controls.Grid erbt von System.Windows.Controls.Panel und damit laut Microsoft auch die Eigenschaft Children [Mic075] . Über diese Children Eigenschaft lässt sich eine System.Windows.Controls.UIElementCollection ansprechen, so Microsoft in der MSDN Library [Mic076] . Diese Collection ist eine Liste, der beliebig viele System.Windows.UIElement Elemente hinzugefügt werden können.
Im folgenden Beispiel wird die Verschachtelbarkeit noch einmal verdeutlicht. Dabei sei zu beachten, dass die Klassen, die eine Verschachtelbarkeit ermöglichen, auch von eigenen Klassen geerbt werden können. Auch Klassen, die mit Oberflächen gar nichts zu tun haben, wie im Unter-Unterabschnitt 5.1.2 Eigene Objektstrukturen gezeigt, können so in die Lage versetzt werden, andere Elemente zu beinhalten.

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid>
        <Button Height="100" Width="200" Name="myButton">
            <StackPanel Width="150">
                <Button Content="Klick mich..."/>
                <TextBlock>
                    Hallo Welt!
                </TextBlock>
                <TextBox/>
            </StackPanel>
        </Button>
    </Grid>
</Window>
```

In dieser beispielhaften XAML Datei besteht das Fenster nachwievor aus einem Grid Panel. Dieses Grid Panel enthält immer noch einen Button, aber dieser Button besitzt als Content Eigenschaft jetzt nicht mehr einen Text, sondern ein weiteres UIElement. Da die Content Eigenschaft nur ein einziges Child Element im Button zulässt, wird hier ein weiteres Panel gesetzt, sodass im Button mehrere UIElemente angezeigt werden können. In diesem StackPanel wird nochmal ein Button platziert. Dieser Button wird jetzt auf dem äußeren Button dargestellt. Allerdings wird hier die Content Eigenschaft des inneren Buttons nicht mehr über Element Schreibweise, sondern über Attribut Schreibweise gesetzt (siehe Unter-Unterabschnitt 5.1.4 auf dieser Seite weiter unten).

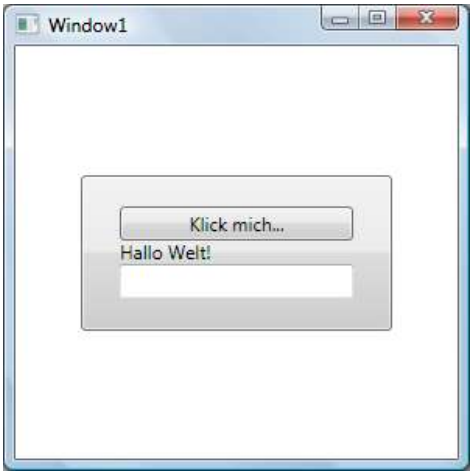


Abbildung 4 Fenster aus der komplex verschachtelten XAML Datei

Durch die Möglichkeit von WPF alles in allem zu verschachteln, kann die Oberfläche sehr umfangreich und komplex werden. Zum Beispiel enthält der Button aus dem obigen Beispiel neben einem weiteren Button einen Text und eine Text Eingabe Box. Die Möglichkeiten, die sich hier bieten, stehen in keinem Vergleich zu denen mit Windows Forms. Dabei ist die Einfachheit, eine solche hierarchische Oberfläche zu bauen, mit XAML so einfach wie noch nie geworden. Natürlich kann man es hier auch übertreiben.

5.1.4 Alternative Schreibweise von Attributen

Der Inhalt eines Buttons wird über die Content Eigenschaft gesetzt. In den bisherigen Beispielen wurde diese Eigenschaft in XAML in zwei verschiedenen Arten gesetzt. Eine Möglichkeit ist es, die Attribute Syntax zu benutzen, wie es im Beispiel gezeigt wird.

```
<Button Content="Klick mich..."/>
```

Wenn aber der Bedarf besteht, UIElemente anstelle von Text als Button Content zu setzen, bekommt man schnell ein Problem. In diesem Fall würde man die Content Syntax benutzen. Dabei ist zu

beachten, dass es nicht möglich ist, die Content Eigenschaft als Attribut zu setzen und zusätzlich noch Content als Unterelement zu definieren. Im folgenden Beispiel wird die Content Syntax gezeigt.

```
<Button>
      <TextBlock>
            Klick mich...
      </TextBlock>
</Button>
```

Diese Variante reicht aus, wenn man nur eine Eigenschaft, wie hier die Content Eigenschaft, beschreiben will. Sobald man aber mehrere Attribute als Unterelemente schreiben muss, reicht dieser Ansatz nicht mehr aus. Zum Beispiel, wenn es sich bei den zuzuweisenden Konstrukten um komplexere UIElemente handelt, dann muss es eine Möglichkeit geben, mehrere Attribute als Unterelemente schreiben zu können. Diese Variante nennt sich laut Rainer Stropek Property Element Syntax [Rai07] . Angenommen der Button soll neben einem Text auch einen speziellen Hintergrund bekommen, dann können die Eigenschaften des Elements als Unterelemente geschrieben werden, wie aus dem Beispiel zu entnehmen ist. Das Ergebnis (siehe Beispiel) ist jetzt ein hübscher Button, der nicht nur einen komplexen Inhalt, sondern auch einen komplexen Hintergrund enthält.

```
<Button>
      <Button.Background>
          <LinearGradientBrush>
              <GradientStop Color="Green" Offset="1.0" />
              <GradientStop Color="Red" Offset="0.1" />
          </LinearGradientBrush>
      </Button.Background>
      <Button.Content>
            <TextBlock Foreground="White">
                  Klick mich...
            </TextBlock>
      </Button.Content>
</Button>
```

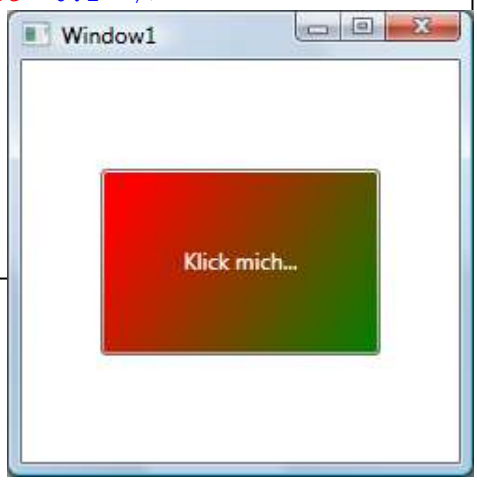


Abbildung 5 Fenster mit Button mit umfangreichen Eigenschaften

Dirk Frischalowski beschreibt diese Syntax in seinem Tutorial [Unk07] als einfache Aneinanderreihung des Elements und dem Name der Eigenschaft, mit einem Punkt getrennt. So könnten selbstverständlich alle Attribute des Button Elements definiert werden.

5.1.5 Namen

Nachdem die Elemente auf der Oberfläche angeordnet wurden, ist es meistens erforderlich diese mit Event Mechanismen zu versehen. Doch um auf die Elemente zugreifen zu können, müssen diese über einen eindeutigen Namen verfügen. Würde man die Oberfläche komplett in .NET Code schreiben, müsste man die Instanz des Fensters einer Variablen zuweisen um Eigenschaften zu setzen. In XAML ist dies nicht nötig, solange die Klasse nicht von anderen Stellen adressiert werden soll. Dirk Frischalowski schreibt in seinem Tutorial über WPF [Unk07] , dass alle Komponenten von der Klasse FrameworkElement die Eigenschaft „Name“ erben. In den obigen Beispielen wird jeweils bei einem Button diese Eigenschaft auf einen Wert gesetzt. Doch wie sieht es bei Elementen aus, die diese

Eigenschaft nicht erben, wie es zum Beispiel bei eigenen Klassen, die nichts mit Oberflächen zu tun haben der Fall ist? Oder wenn die Eigenschaft „Name“ dort zwar existiert, aber für etwas ganz anderes verwendet werden soll, wie im Beispiel im Unter-Unterabschnitt Eigene Objektstrukturen auf Seite 15? Über den Namensraum System.Windows.Markup, der in einer typischen XAML Datei dem Präfix x zugewiesen wird, lässt sich jedes Element in XAML benamen.
Ein Objekt der eigenen Klasse, das die Eigenschaft Name für einen eigenen Zweck verwendet, kann über das Attribut x:Name einem Namen zugeordnet werden. Dieser Name wäre in .NET Code der Bezeichner einer Variablen vom Typ der eigenen Klasse.

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:myA="clr-namespace:MyBusinessLogik"
    Title="Window1" Height="250" Width="249">
    <Grid>
        <Button>
            <myA:Person x:Name="myPerson" Name="Bill">
            </myA:Person>
        </Button>
    </Grid>
</Window>
```

5.2 XAML & Code

Wie bereits erwähnt, sollen Elemente auf einer Oberfläche mit Code verbunden werde, sodass die Oberfläche auch funktional wird. Dafür ist es erforderlich, alle Elemente, auf die dabei zugegriffen werden soll, mit einem eindeutigen Namen versehen zu haben. Doch wie kann auf jetzt mit .NET Code auf diese Elemente zugegriffen werden und wo kommt dieser Code überhaupt hin? Ein Blick in den Solution Explorer im Visual Studio offenbart, dass zu jeder XAML Datei auch eine Code Datei existiert. Dabei handelt es sich laut Microsofts MSDN Library um die sogenannte Code Behind Datei [Micbe2] . Im Beispiel heißt das WPF Projekt „WpfApplication“ und die XAML Datei „Window1.xaml“. Die entsprechende Code Behind Datei heißt „Window1.xaml.cs“ und wird im Solution Explorer der XAML Datei zugeordnet angezeigt.

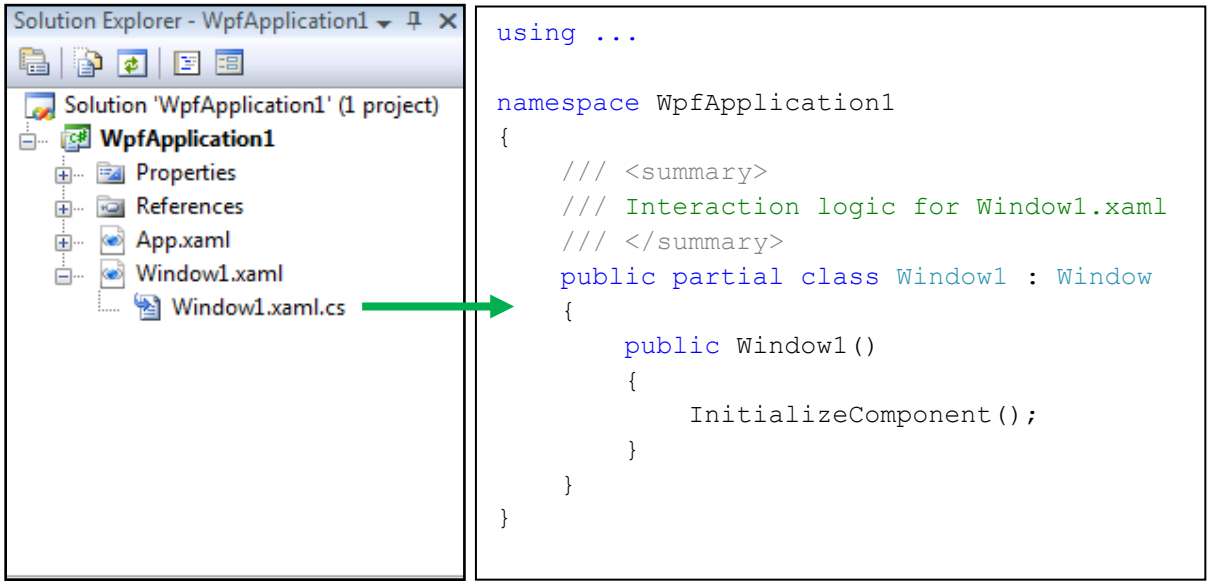


Abbildung 6 Solution Explorer Ansicht im Visual Studio 2008

In dieser Datei werden in der Regel die Event Handler implementiert, um auf Ereignisse der Oberfläche reagieren zu können. Mehr dazu aber im Unterabschnitt 5.5 Events. Im obigen Beispiel handelt es sich bei dem WPF Projekt um ein C# Projekt. Da das .NET Framework verwendet wird, ist die konkrete Sprache nicht entscheidend. Einer der vielen Vorteile von XAML ist ja die Sprachunabhängigkeit, die bereits im Unterabschnitt 4.2 Vorteile von XAML erwähnt wurde. Das .NET Framework stellt laut Microsoft eine Klassenbibliothek für viele Sprachen bereit [Micbe3] , sodass die Verwendung von XAML in jeder dieser .NET Sprachen prinzipiell möglich ist.
Um auf die Code Behind Datei zurückzukommen, sieht man in der noch ziemlich leeren Klasse sofort, dass es sich um eine „partial“ Klasse handelt. Laut Microsoft bedeutet das, dass diese Klasse auf mehrere Dateien verteilt werden kann [Mic077] . Demzufolge gibt es also noch eine weitere Datei, die auch noch .NET Code enthält. Der Aufruf der InitializeComponent Methode im Konstruktor der Code Behind Klasse bestätigt das. Doch wo wird diese Methode implementiert? Mit Visual Studio kann zu der Implementierung dieser Methode gesprungen werden.
Das Ziel dieses Sprungs stellt eine weitere .NET Code Datei dar, die nicht im Solution Explorer vom Visual Studio angezeigt wird. Offensichtlich handelt es sich hierbei um eine automatisch generierte Datei, die den Namen „Window1.g.cs“ trägt. Das „g“ steht laut Microsoft für „generated“ [Mic078] .

```
#pragma checksum "..\..\Window1.xaml" "{406ea660-64cf-4c82-b6f0-
42d48172a799}" "AB4CDA123E53A53945B84FC6C78C2C0E"
//------------------------------------------------------------------------
--------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.1378
//
//     Changes to this file may cause incorrect behavior and will be
lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------
--------
...
    public partial class Window1 : System.Windows.Window,
      System.Windows.Markup.IComponentConnector {

        internal System.Windows.Controls.Button myButton;

        private bool _contentLoaded;
            ...
```

In dieser automatisch erstellten Klasse (siehe Code Auszug), wird für jedes Element aus der XAML Datei, dem ein Name zugewiesen wurde, ein Attribut erstellt. Im obigen Beispiel wurde übrigens die XAML Datei aus dem Unter-Unterabschnitt Verschachtelbarkeit von Seite 16 verwendet. Dort wurde ein Button mit dem Namen „myButton“ versehen, der hier jetzt als Bezeichner verwendet wird. Außerdem erbt diese Klasse von System.Windows.Window, da das Root Element der XAML Datei ein Window Element ist. Auf die restlichen Bestandteile dieser automatisch generierten Datei, die im

obigen Ausschnitt nicht zusehen sind, wird im Unterabschnitt 5.4 Verarbeitung zur Laufzeit näher eingegangen.
Es existiert also die Generated Code Datei, über die die XAML Elemente bekannt gemacht werden, und es existiert die Code Behind Datei um das Verhalten der Oberfläche zu programmieren. Diese beiden Dateien beschreiben die Klasse, dessen Instanz zur Laufzeit ein Fenster erzeugen wird. Es gibt .NET Sprachen, die das Schlüsselwort „partial“ nicht kennen, wie zum Beispiel F# oder J#. Das bedeutet, dass die Fenster Klasse aus dem obigen Beispiel nicht in eine Generated Code und eine Code Behind Datei aufgeteilt werden kann. Allerdings bedeutet das nicht, dass mit diesen Sprachen keine XAML Oberfläche instanziiert werden kann. Robert Pickering zeigt das an einem Beispiel in F# [Rob06] , auf das in dieser Ausarbeitung nicht näher eigegangen werden kann. Im Unter-Unterabschnitt 5.5.4 Implementierung von Event Handlern wird aber erläutert, wie Events ohne eine Code Behind Datei implementiert werden können.

5.3 Kompilierung

Wie bereits mehrfach erwähnt, wird XAML im Gegensatz zu vielen anderen Markup Sprachen für Oberflächen nicht interpretiert, sondern kompiliert. Allerdings wird die XAML Datei nicht direkt in .NET Code kompiliert, sondern in eine binäre Application Markup Datei. Rainer Stropek hat in seinem Vortrag auf der .NET Entwickler Konferenz DEVcamp07 den Kompilierungsprozess von XAML in WPF, wie im unteren Bild gezeigt, vorgestellt [Rai07] .

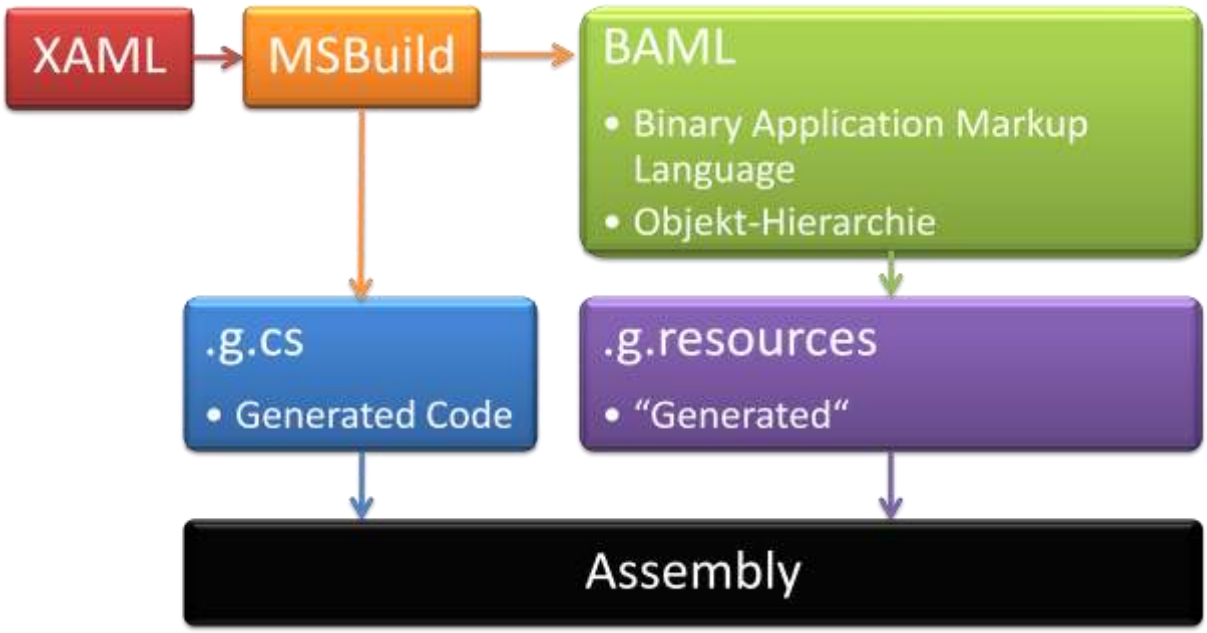


Abbildung 7 Kompilierungsprozess von XAML

Demnach wird XAML über die Microsoft Build Plattform, die auch in Visual Studio integriert ist, zum Einen in eine binäre Zwischendarstellung kompiliert. Ursprünglich gab es einen eigenen XAML Kompiler (xamlc.exe), der aber laut Ashish Shetty durch die MSBuild Engine ersetzt wurde [Ash06] . Aus dieser binären Darstellung, der BAML Datei, wird zunächst eine Ressource generiert, die genau wie die XAML Datei die komplette Hierarchie der beschriebenen Objekte abbildet. Diese Struktur geht beim Kompilieren also nicht verloren. Microsoft beschreibt diesen Vorgang in der MSDN Library [Mic078] . Die XAML Datei wird dabei geparst, validiert und in seine Bestandteile zerlegt, bevor sie in binärem Format weiter verarbeitet werden kann. Dadurch kann das Laden der mit XAML

beschriebenen Objekt Struktur zur Laufzeit wesentlich schneller erfolgen, als XAML im ASCII XML Format direkt zu laden, so Microsoft [Mic078] .
Beim Kompilieren speichert der MSBuild Kompiler die erzeugte BAML Datei temporär im Projektverzeichnis ab. Im Nachfolgenden Beispiel wurde eine XAML Datei kompiliert und die erzeugte BAML Datei in einem Editor geöffnet. Wie erwartet ist diese Datei nun fast nur noch maschinell lesbar, obwohl es sich um eine sehr einfache Objekt Hierarchie handelt. Hier wird lediglich ein Fenster mit einem Grid Panel und einem Button beschrieben.

```
<Window x:Class="WpfApplication2.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="300" Width="300">
    <Grid>
        <Button>
            Hallo Welt
        </Button>
    </Grid>
</Window>
```

MSBuild

```
....M.S.B.A.M.L...`...`...`...ÿÿÿÿ......WpfApplication2.......WpfApplica
tion2.Window1....5.........\..XPresentationFramework, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35.R..NWindowsBase,
Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35.W..SPresentationCore, Version=3.0.0.0,
Culture=neutral,
PublicKeyToken=31bf3856ad364e35.D.9http://schemas.microsoft.com/winfx/20
06/xaml/presentation........5.........6.x,http://schemas.microsoft.com/w
infx/2006/xaml......5.........$.Ñÿ.300¤þ5.........$.Çÿ.300¤þ6.....òÿ5.....
.....ÿ..+ÿ5.........Éÿ..òÿ6.......Hallo
Welt.5.........5.........5.........
```

Abbildung 8 Window1.baml - Kompilierte XAML Datei

Das einzige, was noch lesbar ist, ist der Button Text, sowie die Pfade der Namensräume und gewisse Assembly Informationen.
Um wieder zurück zum Prozess des Kompilierens zu kommen, so wird die XAML Datei nicht nur in eine BAML Datei kompiliert. Zusätzlich wird die bereits vielfach erwähnte Generated Code Datei erstellt. Wie bereits im Unterabschnitt 5.2 XAML & Code erwähnt, sind in dieser generierten sprachspezifischen Code Datei für alle benamten Elemente aus der XAML Datei Attribute vorhanden, über die auf die Instanzen dieser Elemente zugegriffen werden kann.

5.4 Verarbeitung zur Laufzeit

Der Aufbau der Generated Code Datei, die beim Übersetzen der XAML Datei erstellt wird, wurde im Unterabschnitt 5.2 XAML & Code kurz beschrieben. Im Unterabschnitt 5.3 Kompilierung wurde dann gezeigt, wie die Markup Sprache von Microsoft übersetzt wird. Dabei wurde die BAML Datei als Ressource in die erzeugte Assembly eingelagert. Der Entwickler möchte aber mit diesen Ressourcen nichts zu tun haben, sondern mit Objekten und Attributen arbeiten. Dafür enthält die automatisch generierte .NET Code Datei entsprechende Repräsentation der XAML Elemente. Es stellt sich jetzt die Frage, wie die BAML Datei zur Laufzeit geladen wird und wie die Attribute der Generated Code Datei mit den XAML Elementen referenziert werden können. Um diese Verarbeitung verstehen zu können,

wird im Folgenden der Aufbau der automatisch generierten Code Datei, die bereits im Unterabschnitt XAML & Code auf Seite 19 kurz beschrieben wurde, genauer analysiert.
Wie bereits erwähnt, erbt die Klasse aus dieser Datei von der .NET Klasse, die über das Root Element in der XAML Datei bestimmt wurde. Somit kann die Klasse zum Beispiel als Fenster weiter verwendet werden. Zusätzlich wird das Interface System.Windows.Markup.IComponentConnector implementiert, wie dem folgenden Ausschnitt aus der Generated Code Datei entnommen werden kann.

```
public partial class Window1 : System.Windows.Window,
        System.Windows.Markup.IComponentConnector
```

Dadurch kann diese Klasse von außen allgemeingültig behandelt werden, egal ob es sich um ein Fenster, eine Seite oder einen anderen Typ handelt, der als Root Element in XAML in Frage kommt. Aus der MSDN Library lässt sich entnehmen, dass das Interface IComponentConnector zwei Methoden enthält [Mic079] , die von der ableitenden Klasse implementiert werden müssen.
Die Methode InitializeComponent wird von der Generated Code Klasse implizit implementiert. Dadurch kann die Methode bei allen Objekten dieser Klasse aufgerufen werden, was im Konstruktor der Klasse aus der Code Behind Datei auch praktiziert wird (siehe Seite 19). In der Methode, die im unteren Code Auszug gezeigt wird, wird ein Uniform Resource Identifier erstellt, der einen relativen Verweis auf die projektinterne XAML Datei enthält, aus der diese generierte Code Datei erstellt wurde. Über die Methode System.Windows.Application.LoadComponent wird dann die über das URI Objekt spezifizierte XAML Datei zur Laufzeit in eine Zielkomponente geladen. Im folgenden Auszug wird die Zielkomponente, das konkrete Fenster selbst, per Referenz übergeben. Laut Microsoft kann die Methode auch eine neue Zielkomponente erzeugen, die dann zurückgegeben wird, wie in der MSDN Library beschrieben [MSD07] .

```
public void InitializeComponent() {
    if (_contentLoaded) {
        return;
    }
    _contentLoaded = true;
    System.Uri resourceLocater =
      new System.Uri("/WpfApplication1;component/window1.xaml",
      System.UriKind.Relative);

    System.Windows.Application.LoadComponent(this, resourceLocater);
}
```

Bei der XAML Datei, die über den URI der LoadComponent Methode bekannt gemacht wird, handelt es sich in diesem Fall um die als Ressource in die Assembly eingelagerte BAML Datei. Die projektinterne XAML Datei existiert zur Laufzeit ja nicht mehr. Microsoft schreibt in der MSDN Library, dass mittels dem URI auch eine externe XAML Datei geladen werden kann, die nicht zuvor in eine BAML Datei kompiliert [MSD07] wurde. Allerdings ist das Laden einer bereits als binäre Ressource vorliegender Datei wesentlich schneller.
Sollte der XAML Inhalt nicht aus einer Datei kommen, so kann er auch aus einem Stream geladen werden. Im .NET Namensraum System.Windows.Markup existiert dafür die Klasse XamlReader, die laut Microsoft eine Load Methode [Mic0710] anbietet. Mit dieser Methode kann genau wie mit der LoadComponent Methode ein XAML formatierter Inhalt geladen werden. Es muss sich also nicht

immer um Dateien handeln, in denen XAML Inhalt zur Verfügung gestellt wird. Im gleichen Namensraum existiert übrigens auch der XamlWriter, mit dessen Hilfe Oberflächenelemente serialisiert werden können [Mic0711] .
Wenn ein XAML Inhalt über einen Stream geladen wird, kann auf benamte Komponenten nicht so einfach zugegriffen werden. Der XamlReader liefert aus dem Stream ein Objekt zurück. In diesem Objekt müssen konkrete Elemente erst gesucht werden, um sie verwenden zu können. Wird eine XAML Datei aber über die LoadComponent Methode der Klasse Application geladen, kann ein Mechanismus genutzt werden, der auch beim Kompilieren der XAML Datei durch MSBuild berücksichtigt wird.
Wie bereits erwähnt, implementiert die Klasse aus der Generated Code Datei die Schnittstelle System.Windows.Markup.IComponentConnector. Im Gegensatz zur InitializeComponent Methode implementiert die Klasse die Connect Methode des Interfaces explizit. Das bedeutet, dass diese Methode nur aufgerufen werden kann, wenn das Fenster als IComponentConnector verwendet wird, oder dessen Referenz entsprechend gecastet wird.

```
void System.Windows.Markup.IComponentConnector.Connect(
      int connectionId, object target) {
    switch (connectionId) {
          case 1:
          this.myButton = ((System.Windows.Controls.Button)(target));
          return;
    }
    this._contentLoaded = true;
}
```

Indem die Referenz auf das Fenster Objekt der LoadComponent Methode übergeben wird, kann diese die Connect Methode aufrufen. Dabei ist zu beachten, dass jedem Attribut in der Klasse des Fenster Objekts beim Kompilieren eine eindeutige Connection ID zugewiesen wurde. Wenn die Connect Methode dann zur Laufzeit aufgerufen wird, wird eine Connection ID, sowie eine Referenz auf eine konkrete Instanz eines XAML Elements übergeben. Die Connect Methode weißt diese Referenz dann abhängig von der Connection ID einem Attribut zu, sodass auf die Instanz auch von der Code Behind Klasse aus zugegriffen werden kann. Dabei werden noch eventuell vorhandene EventHandler zugewiesen (siehe 5.5.4 Implementierung von Event Handlern).
Durch diese Variante können XAML Elemente sehr flexibel geladen und mit EventHandlern versehen werden. Im Gegensatz zum Laden per XamlReader brauch das Ergebnis nicht mehr selbst durchsucht zu werden, sondern das Ergebnis bindet sich über dir Connect Methode automatisch an die in der Generated Code Datei angegebenen Attribute.

5.5 Events

Es wurden bereits mehrfach EventHandler angesprochen, die Elementen in XAML zugewiesen werden. In diesem Kapitel werden die Event Mechanismen von WPF erläutert, sowie an einem Beispiel demonstriert. Im Unter-Unterabschnitt 5.1.3 Verschachtelbarkeit wurde gezeigt, dass sich mit XAML Elemente nahezu beliebig verschachteln lassen. Zum Beispiel können Buttons in Buttons geschachtelt werden. Es liegt also nahe, dass in dieser mächtigen Technologie Möglichkeiten bestehen müssen, um auf Ereignisse angemessen reagieren zu können. Zum Beispiel muss ein Button, der einen anderen Button enthält es mitkriegen, dass der innere Button angeklickt wurde. In der Windows Presentation Foundation stellt Microsoft dafür drei unterschiedliche Arten an Events

zur Verfügung. In der MSDN Library werden diese Events von Microsoft beschrieben [Mic0712] . Dabei handelt es sich um direkte, Bubbling und Tunneling Events.

5.5.1 Direktes Event

Das direkte Event ist bereits aus Windows Forms Anwendungen bekannt. Bei einem konkreten Steuerelement kann ein EventHandler definiert werden, der nur dann aufgerufen wird, wenn dieses Element ein Ereignis auslöst. Das Click Event des Buttons ist ein solches direktes Event. Nur der Button, der wirklich angeklickt wurde, bekommt damit die Gelegenheit das Ereignis zu behandeln, so Dirk Frischalowski in seinem WPF Tutorial [Dir071] .

5.5.2 Bubbling Event

Anders ist das beim Bubbling Event. Dieses Event gibt nicht nur dem Element, das auch wirklich das Ereignis ausgelöst hat, die Möglichkeit das Ereignis zu behandeln. Vielmehr werden hier die EventHandler aller Elemente auf dem Weg vom Ereignis auslösenden Element bis zum Root Element aufgerufen. Dirk Frischalowski beschreibt dieses Verhalten in seinem Tutorial über Ereignisbehandlung [Dir071] .

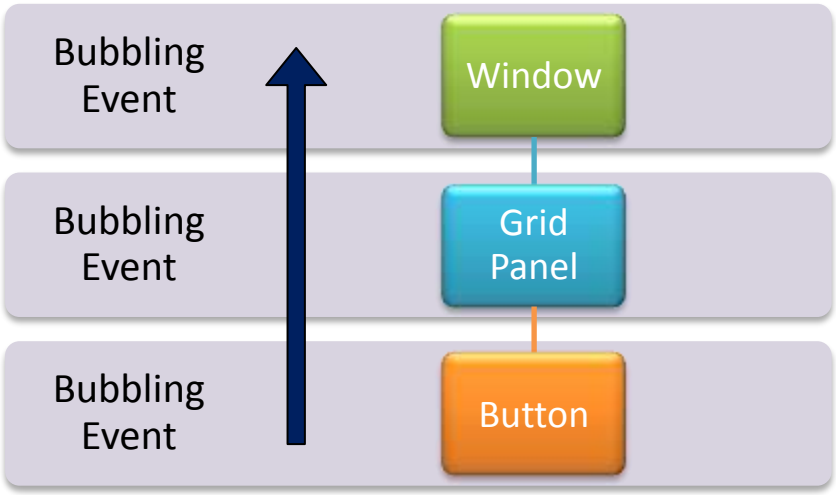


Abbildung 9 Bubbling Event im Element Baum

Wenn zum Beispiel ein Ereignis ausgelöst werden soll, wenn der Benutzer in ein Grid Panel klickt, so soll dieses Ereignis auch ausgelöst und behandelt werden können, wenn der Benutzer auf einen Button klickt, der sich in diesem Grid Panel befindet. Zusätzlich soll der Button das Ereignis aber auf eine andere Art und Weise behandeln können. In diesem Fall reicht ein direktes Event nicht mehr aus und es sollte ein Bubbling Event verwendet werden.

5.5.3 Tunneling Event

Ein sehr ähnliches, aber dennoch sehr unterschiedliches Event, ist das Tunneling Event. Angenommen, die Oberfläche besteht aus einem Button. Dieser Button enthält eine TextBox, sodass ein Benutzer etwas eingeben kann.

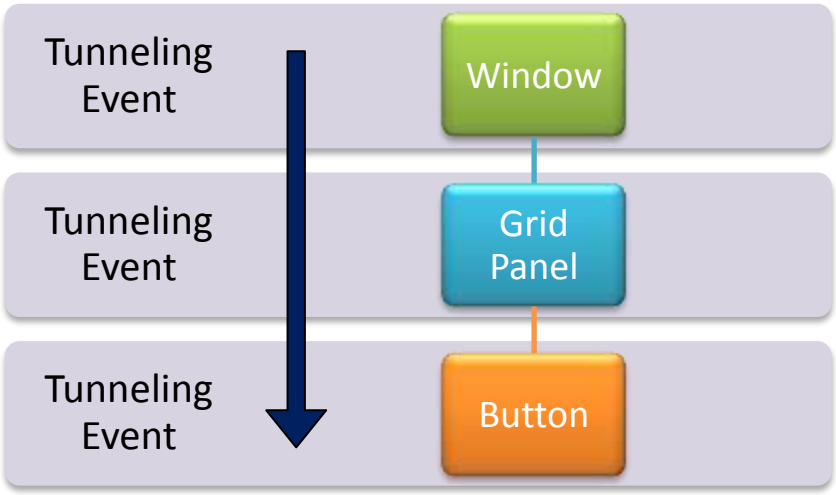


Abbildung 10 Tunneling Event im Element Baum

Wenn dieser Benutzer den Focus der Anwendung auf die TextBox gesetzt hat und die Enter Taste auf der Tastatur betätigt hat, soll der Button das mitkriegen. Allerdings soll er das mitkriegen, bevor das Ereignis die TextBox selbst erreicht, um beispielsweise

irgendwelche Prüfungen durchführen zu können. In diesem Fall reicht das Bubbling Event auch nicht mehr aus, da bei dessen Verwendung der Button erst nach der TextBox die Möglichkeit bekommen würde auf das Ereignis zu reagieren. Für diesen Fall gibt es das Tunneling Event. Laut Dirk Frischalowski startet das Ereignis beim Root Element der Hierarchie und ruft dort den entsprechenden EventHandler auf, bevor es sich bis zum Event auslösenden Element nach unten hangelt [Dir071] . Tunneling Events werden also vor den Bubbling Events ausgelöst und geben dem Entwickler optimale Möglichkeiten sämtliche Ereignisse behandeln zu können.
In der Regel heißen die Tunneling Events genauso wie die BubblingEvents, allerdings mit dem Präfix „Preview“. Dabei sei zu bedenken, dass nicht alle Events ein zugehöriges Bubbling oder Tunneling Event besitzen. Das Click Event zum Beispiel existiert nur als direktes Event. Über die MSDN Library kann aber ermittelt werden, zu welchen Element Klassen welche Events gehören [Mic0712] .

5.5.4 Implementierung von Event Handlern

Nachdem die einzelnen Mechanismen eingeführt wurden, stellt sich nun die Frage, wie diese Events behandelt werden können. Die Implementierung der bereits mehrfach erwähnten EventHandler kann auf zwei Arten erfolgen.
Zum Einen kann die EventHandler Methode direkt in XAML eingebettet werden. Dabei wird das Code Tag aus dem System.Windows.Markup Namespace verwendet, wie es Dirk Frischalowski in seinem WPF Tutorial beschreibt [Dir071] . Anschließend kann beliebiger .NET Code geschrieben werden. Da es sich dabei nicht um gültiges XML handelt, darf dieser Code nicht geparst werden, was laut Christian Pape durch die Verwendung des typischen XML Konstrukts „<![CDATA[ ]]>“ erreicht werden kann [Pap06] . Dieser Code wird beim Kompilieren in die Generated Code Datei verschoben.

```
<Button
PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown_1"/>
<x:Code>
    <![CDATA[
    private void Button_PreviewMouseLeftButtonDown_1(object sender,
    MouseButtonEventArgs e)
    {
        MessageBox.Show("Hallo");
        e.Handled = true;
    }
    ]]>
</x:Code>
```

Diese Art der Implementierung ist allerdings sehr unschön, da sie der Philosophie von XAML, Logik und Aussehen einer Oberfläche konsequent zu trennen, widerspricht. Im obigen Beispiel wurde ein Handler für das Tunneling Event PreviewMouseLeftButtonDown geschrieben, der eine MessageBox anzeigt, wenn auf den Button geklickt wird. Der EventHandler wird dafür über das entsprechende Attribut dem Button Element hinzugefügt.
Ein interessanter Aspekt ist dabei die Methoden Signatur des Handlers. Neben dem Element, das das Ereignis ausgelöst hat, wird ein MouseButtonEventArgs Objekt übergeben. Über dieses Objekt kann das Event direkt beeinflusst werden. Wie im obigen Beispiel kann das Ereignis auch als behandelt markiert werden. Dafür wird beim EventArgs Objekt die Eigenschaft „Handled“ auf true gesetzt. Das hat zur Folge, dass das Event nicht weiter getunnelt oder gebubbelt wird. Auf diese Weise kann ein Ereignis vorzeitig abgefangen werden, sodass nachfolgende EventHandler nicht mehr aufgerufen werden.

Eine andere Art der Ereignisbehandlung ist das implementieren der Handler in der Code Behind Datei. Die Zuweisung wird in XAML dabei wieder über das entsprechende Attribut vorgenommen. In der Generated Code Datei wird die EventHandler Methode der Instanz des entsprechenden XAML Elements zugewiesen. Das erfolgt an der Stelle, an der die Instanz auch mit dem zugehörigen Attribut des Objekts verbunden wird, also der Connect Methode (siehe Code Ausschnitt).

```
switch (connectionId)
{
      case 1:
      this.myButton = ((System.Windows.Controls.Button)(target));
      this.myButton.Click +=
            new System.Windows.RoutedEventHandler(this.Button_Click);
      return;
}
```

5.5.5 Beispiel

Im Folgenden werden die Event Mechanismen von WPF noch einmal an einem Beispiel demonstriert. Dafür wird eine Oberfläche erstellt, die einen Button enthält. Dieser Button enthält ein StackPanel, welches wiederrum ein Bild und einen TextBlock enthält.

Zusätzlich wird dem Button, dem StackPanel und dem Bild jeweils ein EventHandler für das Bubbling Event MouseLeftButtonDown zugewiesen. Auch für das entsprechende Tunneling Event wird jeweils eine EventHandler Methode zugewiesen. Zusätzlich wird auch das direkte Click Event beim Button registriert.

Ziel dieser Demonstration ist es ein Gefühl für die Reihenfolge der Ereignisse zu vermitteln. Dafür werden in der Code Behind Datei sämtliche Handler Methoden implementiert, sodass diese jeweils ihre Identität, sowie das Event das gerade behandelt wird, ausgeben.

Der untere Ausschnitt aus der entsprechenden XAML Datei zeigt die Zuweisung der konkreten Ereignisbehandlungs Methoden.

Abbildung 11 Event Demo / Der große Bill

```
<Button
MouseLeftButtonDown="Button_MouseLeftButtonDown"
PreviewMouseLeftButtonDown="Button_PreviewMouseLeftButtonDown"
Click="Button_Click">
      <StackPanel
MouseLeftButtonDown="StackPanel_MouseLeftButtonDown"
PreviewMouseLeftButtonDown="StackPanel_PreviewMouseLeftButtonDown">
            <Image Width="130" Source="D:\temp\billg1.jpg"
MouseLeftButtonDown="Image_MouseLeftButtonDown"
PreviewMouseLeftButtonDown="Image_PreviewMouseLeftButtonDown"/>
            <TextBlock FontSize="15">Der große Bill!</TextBlock>
      </StackPanel>
</Button>
```

Der untere Ausschnitt aus der Code Behind Datei zeigt die Implementierung der EventHandler des Buttons. Die Methoden für das StackPanel und das Bild sehen ähnlich aus.

```
private void Button_MouseLeftButtonDown(object sender,
MouseButtonEventArgs e)
{
    Debug.WriteLine("MouseLeftButtonDown von Button");
}

private void Button_PreviewMouseLeftButtonDown(object sender,
MouseButtonEventArgs e)
{
    Debug.WriteLine("PreviewMouseLeftButtonDown von Button");
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    Debug.WriteLine("Click von Button");
}
```

Nachdem die Anwendung kompiliert und ausgeführt wurde, ergibt sich nach einem Klick auf das Bild im Button folgender Output.

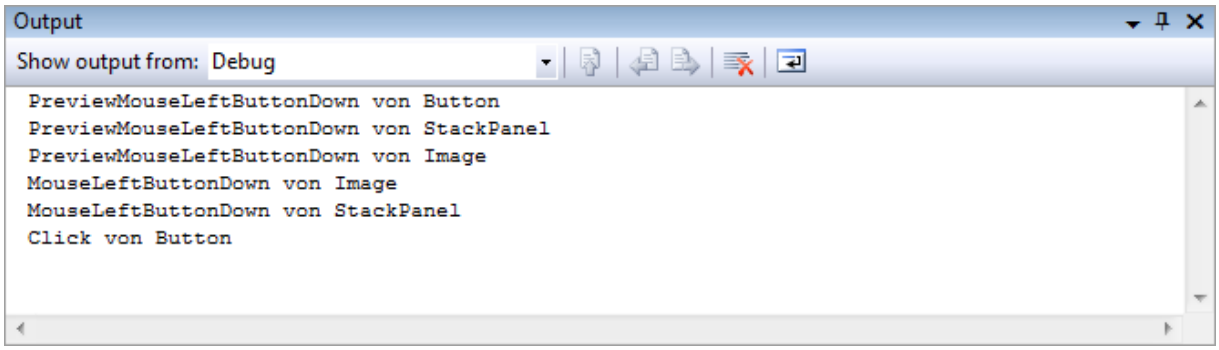


Abbildung 12 Event Demo / Output

Zunächst wird das Tunneling Event des Buttons behandelt. Der Button stellt das zentrale Element im Element Baum der XAML Datei dar. Danach tunnelt das Event zum StackPanel und wird dort vom entsprechenden EventHandler behandelt. Anschließend kommt das Tunneling Event zum Bild. Dort ist der Tunneling Prozess des Events beendet und das Ereignis bubbelt zurück in Richtung Root Element. Nachdem das Bubbling Event vom StackPanel behandelt wurde, wird der EventHandler des Click Events für den Button aufgerufen. Das MouseLeftButtonDown Event vom Button wird nicht aufgerufen. Dies hat den Hintergrund, dass das Click Event, welches ja ein direktes Event ist, auf dem MouseLeftButtonDown Event basiert und es laut Dirk Frischalowski als „Handled“ markiert.

> *“Dieses Ereignis […] basiert auf den Ereignissen MouseLeftButtonDown und MouseLeftButtonUp bzw. wird es in diesem Ereignissen ausgelöst. […] Da es […] diese eigentlich ersetzen soll, wird es als behandelt markiert. ”* [Dir071]

Dank diesen mächtigen Event Mechanismen lassen sich selbst Ereignisse in komplexesten WPF Oberflächen effektiv behandeln.

6 Fazit

Die Oberflächenbeschreibungssprache XAML von Microsoft ist eine sehr vielseitige Sprache. Sie benutzt die XML Syntax um Objekt Hierarchien deklarativ zu initialisieren. Doch wird die Microsoft Sprache den in Abschnitt 3 Anforderungen ermittelten Anforderungen auch wirklich gerecht?

Für das Gestalten einer Oberfläche können Elemente, die zuvor imperativ initialisiert werden mussten, einfach verschachtelt werden, sodass eine logische Baumstruktur entsteht. In dieser Ausarbeitung wurde im Unter-Unterabschnitt 5.1.3 Verschachtelbarkeit gezeigt, wie einfach es ist, eine komplexe Oberfläche zu gestalten. In der Windows Presentation Foundation Api vom .NET Framework kann beispielsweise der Inhalt eines Button Elements von einem beliebigen Objekt sein. In der XML Syntax kann durch die Verschachtelung von Elementen dieser Inhalt sehr komfortabel beschrieben werden. XAML ist also ohne viel Aufwand zu erlernen. Es müssen lediglich die WPF Api und verschiedene Möglichkeiten der Ereignisbehandlung erlernt werden.

Da das Beschreiben von komplexen Oberflächen auf eine sehr einfache und intuitive Art erfolgen kann, lassen sich bei der Verwendung von XAML sehr schnell verblüffende Ergebnisse erzielen. Die Verschachtelbarkeit von Elementen ermöglicht eine einfache Anordnung auf der Oberfläche, die in .NET Code bei komplexeren Anwendungen nur wesentlich aufwendiger erreicht werden kann. Es lässt sich also festhalten, dass das deklarative Gestalten von Oberflächen mit XAML im Vergleich zur imperativen Gestaltung mit .NET Code zu deutlich schnelleren Ergebnissen führt.

Das ist auch darauf zurückzuführen, dass das Beschreiben einer Oberfläche mit .NET Code in einen sehr unübersichtlichen Code resultiert. Im Unter-Unterabschnitt 5.1.2 Eigene Objektstrukturen wurde gezeigt, wie ein Fenster in XAML implementiert werden kann und wie die gleiche Implementierung in imperativem Code aussehen würde. Während das Fenster in XAML klar strukturiert und übersichtlich beschrieben werden konnte, war der .NET Code nur sehr schwer lesbar. Gerade bei umfangreichen Oberflächen würde XAML zu jeder Zeit für eine gute Übersicht sorgen und das nachträgliche Verändern erheblich vereinfachen. Bei Windows Forms Anwendungen war es ja so, dass die Oberfläche meist ausschließlich im Visual Studio Designer gestaltet wurde und der erzeugte Code ja nicht angefasst werden sollte. Eine nachträgliche Änderung an einer komplexen Oberfläche war mit Windows Forms ein sehr aufwändiger Prozess, da man meistens doch den erzeugten Code manipulieren musste. Mit der Verwendung von XAML bleibt das nachträgliche Verändern transparent, sodass der Gestalter zu keiner Zeit am automatisch generierten Code etwas anpassen muss. Diese Wartbarkeit verdankt XAML seiner sehr übersichtlichen Strukturierung.

XAML ist sehr leicht zu benutzen. Dabei gehen die Möglichkeiten der Sprache weit über die Beschreibung von Oberflächen hinaus. Der Grundgedanke von XAML ist die deklarative Instanziierung von Objekten einer Struktur. Im Falle von WPF und Silverlight wird dieser Mechanismus dafür benutzt eine Oberfläche zu erzeugen. In der Windows Workflow Foundation wird XAML benutzt um einen Workflow zu beschreiben. Doch auch eigene Strukturen können mit XAML beschrieben und Instanziiert werden, wie im Unter-Unterabschnitt 5.1.2 Eigene Objektstrukturen gezeigt wurde. Dadurch ist es möglich sämtliche Klassen zu verwenden, die das .NET Framework zu bieten hat oder die selbst implementiert wurden. Die Möglichkeiten von XAML sind aus dieser Perspektive betrachtet, nahezu unbegrenzt.

Um die vielen Möglichkeiten von XAML auch nutzen zu können, sind geeignete Werkzeuge erforderlich. Im Prinzip kann die Microsoft Sprache auch mit jedem beliebigen Texteditor geschrieben werden, aber es existieren natürlich wesentlich besser geeignete Editoren. Ein Tool liefert Microsoft beim .NET Framework SDK schon mit, das XAMLPad. Wenn aber der Bedarf besteht, eine sehr anspruchsvolle Oberfläche zu entwickeln, dann gibt es ein mächtigeres Werkzeug für Designer. Wie in Unter-Unterabschnitt 4.1.6 Vorteile von XAML bereits erwähnt, haben Designer mit Microsoft Expression Studio ein umfangreiches Programm zur Hand, mit dem sich Oberflächen gestalten lassen. Entwickler dagegen können das Microsoft Visual Studio 2008 verwenden. Beide Tools unterstützen die XAML Implementierung, setzen allerdings unterschiedliche Schwerpunkte. Zusätzlich gibt es immer mehr Werkzeuge von anderen Herstellern, sodass man sagen kann, dass für die Microsoft Sprache viele geeignete Programme verfügbar sind.

Wie bereits mehrfach erwähnt sind es eher die Designer, die eine Oberfläche entwerfen. Die Entwickler implementieren dann das Verhalten zu dieser Oberfläche. Für beide Seiten gibt es entsprechende Tools, allerdings müssen diese Werkzeuge auch die Zusammenarbeit der beiden Gruppen unterstützen können. Eine erfolgreiche Zusammenarbeit von Designern und Entwicklern ist meistens ausschlaggebend dafür, dass ein Programm bei den Nutzern gut ankommt. Aus diesem Grund muss XAML so flexibel sein, dass es von Designern zur Gestaltung genutzt werden kann und von den Entwicklern, um Events zu implementieren. Im Unter-Unterabschnitt 5.5.4 Implementierung von Event Handlern wurde die Möglichkeit gezeigt, Elemente einer XAML Oberfläche mit Logik zu verbinden. Designer können also die Oberfläche nach ihren Kriterien gestalten und brauchen die XAML Datei anschließend nur an den Entwickler weiterzugeben. Dieser kann dann die erforderlichen Methoden zur Ereignisbehandlung implementieren, ohne das der Designer um das Aussehen seiner Oberfläche besorgt sein muss. Mit der Verwendung von Windows Forms war das nicht so einfach. Dort haben Designer die Oberfläche meist nur in einem Zeichentool „gestaltet“ und den Entwickler dann mit der Umsetzung beauftragt. Natürlich hat das Ergebnis nicht den Erwartungen des Designers entsprochen, da die Entwickler mit den Möglichkeiten der Oberflächengestaltung nicht so vertraut sind. Mit XAML werden die Zusammenarbeit und die Integration von Entwicklern und Designern in einen Workflow also wesentlich verbessert.

Zuletzt bleibt noch die Frage, ob durch XAML eine konsequente Trennung zwischen Aussehen und Logik einer Oberfläche erreicht werden kann. Im Unterabschnitt 5.2 XAML & Code wurde gezeigt, dass es zu jeder XAML Datei eine Code Behind Datei gibt. In dieser Datei können sämtliche EventHandler implementiert werden, die das Verhalten einer Oberfläche definieren sollen. Das konkrete Aussehen wird in der XAML Datei selber beschrieben und kann von einem Tool problemlos verändert werden, ohne dass das Verhalten der Oberfläche beeinflusst wird. Somit wird Logik und Aussehen nicht nur konsequent getrennt, sondern auch in unterschiedlichen Sprachen vorgenommen.

XAML ist also eine flexible und mächtige Beschreibungssprache, die allen Anforderungen aus dem Abschnitt 3 Anforderungen gerecht wird. Im Gegensatz zu vielen anderen Sprachen, die das gleiche Ziel haben, ist bei XAML alles vorhanden, was eine umfangreiche und leicht erlernbare Markup Sprache können muss. Das Gestalten einer Oberfläche ist ein komplexer Prozess, an dem Designer und Entwickler beteiligt sind. Mit XAML kann dieser Vorgang aber optimal durchgeführt und eine erfolgreiche Zusammenarbeit gewährleistet werden.

7 Literatur

[Tho06] Baekdal, Thomas. 2006. The Battle Between Usability and User-Experience. *The Battle Between Usability and User-Experience.* [Online] 19. Juni 2006. [Zitat vom: 13. November 2007.] http://www.baekdal.com/articles/usability/usabilty-vs-user-experience-battle/.

[Marbe1] Clifton, Marc. Frequently Asked Questions. *Frequently Asked Questions.* [Online] [Zitat vom: 10. November 2007.] http://www.myxaml.com/support/faq.aspx.

[Marbe] —. MyXaml Home. *MyXaml Home.* [Online] [Zitat vom: 10. November 2007.] http://www.myxaml.com/.

[Chr04] Coenraets, Christophe. 2004. An overview of MXML. *An overview of MXML.* [Online] 29. März 2004. [Zitat vom: 10. November 2007.] http://www.adobe.com/devnet/flex/articles/paradigm.html.

[And07] Dittes, Andreas. 2007. Microsoft Xtopia - User Experience - the next big thing? *Microsoft Xtopia - User Experience - the next big thing?* [Online] 11. Oktober 2007. [Zitat vom: 11. November 2007.] http://dittes.info/blog/2007/10/11/microsoft-xtopia-user-experience-the-next-big-thing/.

[Unk07] Frischalowski, Dirk. 2007. WPF Tutorial - Einführung in XAML. *WPF Tutorial - Einführung in XAML.* [Online] 2007. [Zitat vom: 4. Dezember 2007.] http://www.gowinfx.de/WPF%20Tutorial/kap3.html.

[Dir071] —. 2007. WPF Tutorial - Ereignisbehandlung. *WPF Tutorial - Ereignisbehandlung.* [Online] 2007. [Zitat vom: 8. Dezember 2007.] http://www.gowinfx.de/WPF%20Tutorial/kap4.html.

[Klabe] Klaene, Michael. Ajax with the ZK Framework. *Ajax with the ZK Framework.* [Online] [Zitat vom: 10. November 2007.] http://www.developer.com/design/article.php/3610476.

[Micbe3] Microsoft. 2007. .NET Framework. *.NET Framework.* [Online] 2007. [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/de-de/netframework/default.aspx.

[Mic078] —. 2007. Building a Windows Presentation Foundation Application. *Building a Windows Presentation Foundation Application.* [Online] 2007. [Zitat vom: 7. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/aa970678.aspx.

[Micbe2] —. 2007. Code-Behind and XAML. *Code-Behind and XAML.* [Online] 2007. [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/en-us/library/aa970568.aspx.

[Mic07] —. 2007. Expression Studio - Neue Werkzeuge für besseres Design. *Expression Studio - Neue Werkzeuge für besseres Design.* [Online] 2. Februar 2007. [Zitat vom: 10. November 2007.] http://www.microsoft.com/germany/expression/expression-studio/overview.aspx.

[Micbe1] —. 2007. Get an early look at Visual Studio 2008. *Get an early look at Visual Studio 2008.* [Online] 2007. [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/de-de/vstudio/aa700831.aspx.

[Mic071] —. 2007. Microsoft .NET Framework 3.5 Beta 2. *Microsoft .NET Framework 3.5 Beta 2.* [Online] 24. Juli 2007. [Zitat vom: 10. November 2007.] http://www.microsoft.com/downloads/details.aspx?familyid=d2f74873-c796-4e60-91c8-f0ef809b09ee&displaylang=en.

[MSD07] —. 2007. MSDN - Application.LoadComponent Method. *MSDN - Application.LoadComponent Method.* [Online] 2007. [Zitat vom: 8. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/system.windows.application.loadcomponent.aspx.

[Mic074] —. 2007. MSDN - ContentControl Content Model Overview. *MSDN - ContentControl Content Model Overview.* [Online] 2007. [Zitat vom: 5. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/ms742554.aspx.

[Mic075] —. 2007. MSDN - Grid Class. *MSDN - Grid Class.* [Online] 2007. [Zitat vom: 5. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/system.windows.controls.grid.aspx.
[Mic079] —. 2007. MSDN - IComponentConnector-Methoden. *MSDN - IComponentConnector-Methoden.* [Online] 2007. [Zitat vom: 8. Dezember 2007.] http://msdn2.microsoft.com/de-de/library/system.windows.markup.icomponentconnector_methods.aspx.
[Micbe5] —. MSDN - Inline Styles and Templates. *MSDN - Inline Styles and Templates.* [Online] [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/en-us/library/ms788725.aspx.
[Mic076] —. 2007. MSDN - Panel.Children Property. *MSDN - Panel.Children Property.* [Online] 2007. [Zitat vom: 5. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/system.windows.controls.panel.children.aspx.
[Mic073] —. 2007. MSDN - Window Class. *MSDN - Window Class.* [Online] 2007. [Zitat vom: 5. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/system.windows.window.aspx.
[Mic0710] —. 2007. MSDN - XamlReader Class. *MSDN - XamlReader Class.* [Online] 2007. [Zitat vom: 8. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/system.windows.markup.xamlreader.load.aspx.
[Mic0711] —. 2007. MSDN - XamlWriter Class. *MSDN - XamlWriter Class.* [Online] 2007. [Zitat vom: 8. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/system.windows.markup.xamlwriter.aspx.
[Mic077] —. 2007. Partial Class Definitions. *Partial Class Definitions.* [Online] 2007. [Zitat vom: 6. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/wa80x488(vs.80).aspx.
[Mic0712] —. 2007. Routed Events Overview. *Routed Events Overview.* [Online] 2007. [Zitat vom: 8. Dezember 2007.] http://msdn2.microsoft.com/en-us/library/ms742806.aspx#routing.
[Mic072] —. 2007. Windows Vista User Experience Guidelines. *Windows Vista User Experience Guidelines.* [Online] 10. Oktober 2007. [Zitat vom: 11. November 2007.] http://msdn2.microsoft.com/en-us/library/aa511258.aspx.
[Micbe] —. 2007. XAML. *XAML.* [Online] 2007. [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/en-us/library/ms747122.aspx.
[Micbe4] —. 2007. XAML and Custom Classes. *XAML and Custom Classes.* [Online] 2007. [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/en-us/library/ms753379.aspx.
[Monbe] Mono Project. Moonlight. *Moonlight.* [Online] [Zitat vom: 10. November 2007.] http://www.mono-project.com/Moonlight.
[Monbe2] —. Olive - Mono. *Olive - Mono.* [Online] [Zitat vom: 10. November 2007.] http://www.mono-project.com/Olive.
[Monbe1] —. Summer2005. *Summer2005.* [Online] [Zitat vom: 10. November 2007.] http://www.mono-project.com/Summer2005.
[Lau07] Moroney, Laurence. 2007. Getting Started with Silverlight. *Getting Started with Silverlight.* [Online] 30. April 2007. [Zitat vom: 10. November 2007.] http://msdn2.microsoft.com/en-us/library/bb404300.aspx.
[Moz07] Mozilla Projekt. 2007. Mozilla Developer Center. *Mozilla Developer Center.* [Online] 30. August 2007. [Zitat vom: 10. November 2007.] http://developer.mozilla.org/de/docs/XUL.
[Uch07] Ogbuji, Uche. 2007. Browser extensions using XUL. *Browser extensions using XUL.* [Online] 2. Oktober 2007. [Zitat vom: 10. November 2007.] http://www.ibm.com/developerworks/web/library/wa-xul1/.
[Ope06] OpenLaszlo Administrator. 2006. Introducing OpenLaszlo. *Introducing OpenLaszlo.* [Online] 5. September 2006. [Zitat vom: 10. November 2007.] http://www.openlaszlo.net/index.php?option=com_content&task=view&id=12&Itemid=32.

[Pap06] Pape, Christian. 2006. Vorlesung XML - Grundlagen. *Vorlesung XML - Grundlagen.* [Online] 20. März 2006. [Zitat vom: 8. Dezember 2007.] http://www.home.hs-karlsruhe.de/~pach0003/xml/01_XML_Basics.pdf.
[Rob06] Pickering, Robert. 2006. F# meets WPF 3D. *F# meets WPF 3D.* [Online] 23. Juni 2006. [Zitat vom: 10. November 2007.] http://www.strangelights.com/blog/archive/2006/06/23/1309.aspx.
[Pot07] Potix Corporation. 2007. Getting Started with ZK. *Getting Started with ZK.* [Online] keine Angabe. Juli 2007. [Zitat vom: 10. November 2007.] http://www.zkoss.org/doc/tutorial.dsp.
[Dir07] Primbs, Dirk. 2007. 3D Tag Cloud II - geht das nicht anders besser? *3D Tag Cloud II - geht das nicht anders besser?* [Online] 16. Oktober 2007. [Zitat vom: 10. November 2007.] http://blogs.msdn.com/dirkpr/archive/2007/10/16/3d-tag-cloud-ii-geht-das-nicht-anders-besser.aspx.
[Ash06] Shetty, Ashish. 2006. Has xamlc.exe been removed? *Has xamlc.exe been removed?* [Online] 27. März 2006. [Zitat vom: 8. Dezember 2007.] http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=315405&SiteID=1.
[Soy07] Soyatec. eFace - XAML/WPF for Java. *eFace - XAML/WPF for Java.* [Online] [Zitat vom: 21. November 2007.] http://www.soyatec.com/eface/.
[Rai07] Stropek, Rainer. 2007. DEVcamp07 - .NET Entwickler Konferenz. *DEVcamp07 - .NET Entwickler Konferenz.* [Online] 18. Oktober 2007. [Zitat vom: 7. Dezember 2007.] http://www.devcamp.at/content/documents/DEVcamp07_Wunderwaffe_XAML.pdf.
[Swi07] SwingML Project. SwingML. *SwingML.* [Online] [Zitat vom: 17. November 2007.] http://swingml.sourceforge.net/index.html.
[Swi072] —. SwingML - Custom Event Handler. *SwingML - Custom Event Handler.* [Online] [Zitat vom: 17. November 2007.] http://swingml.sourceforge.net/files2/CustomEventHandlers.pdf.
[Swi071] —. SwingML - Tutorial. *SwingML - Tutorial.* [Online] [Zitat vom: 17. November 2007.] http://swingml.sourceforge.net/files2/Tutorial.pdf.
[TEIbe] TEIA AG. XML-basierte Beschreibung von Benutzungsschnittstellen. *XML-basierte Beschreibung von Benutzungsschnittstellen.* [Online] [Zitat vom: 10. November 2007.] http://www.teialehrbuch.de/Kostenlose-Kurse/AJAX/20847-XML-basierte-Beschreibung-von-Benutzungsschnittstellen.html.
[XHT07] XHTML2 Working Group. 2007. XHTML2 Working Group Home Page. *XHTML2 Working Group Home Page.* [Online] 23. Oktober 2007. [Zitat vom: 10. November 2007.] http://www.w3.org/MarkUp/.
[Soy071] Yang, Yves. 2007. eFace Feature discussion - 3D graphics. *eFace Feature discussion - 3D graphics.* [Online] 16. November 2007. [Zitat vom: 23. November 2007.] http://www.soyatec.com/forum/viewtopic.php?t=708.