Hochschule für
Technik und Wirtschaft
Dresden (FH)
University of Applied Sciences

Fachbereich Informatik/Mathematik

DIPLOMARBEIT

im Studiengang Medieninformatik

Thema:

Entwicklung einer grafischen Benutzeroberfläche für „Rich Internet Applications". Untersuchung der Leistungsfähigkeit von Flex im Vergleich zur Desktop Technologie WPF

eingereicht von	Stephan Lauterbach
Matrikelnummer	17458
eingereicht am	26.05.2010
Betreuer	Prof. Dr. Teresa Merino

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

0 Einleitung

In den letzten Jahren hat sich das Internet rasant weiter entwickelt. Die Zahl der Nutzer ist kontinuierlich gestiegen. In Abbildung 0.1 ist ersichtlich das es in den letzten 15 Jahren eine Vertausendfachung der Nutzerzahlen stattfand. Dabei ist wie die Abbildung 0.2 zeigt ein breites Altersspektrum im Netz vertreten. Durch diese riesigen und wachsenden Zielgruppen fand auch eine starke Entwicklung bei den im Internet angebotenen Diensten statt.

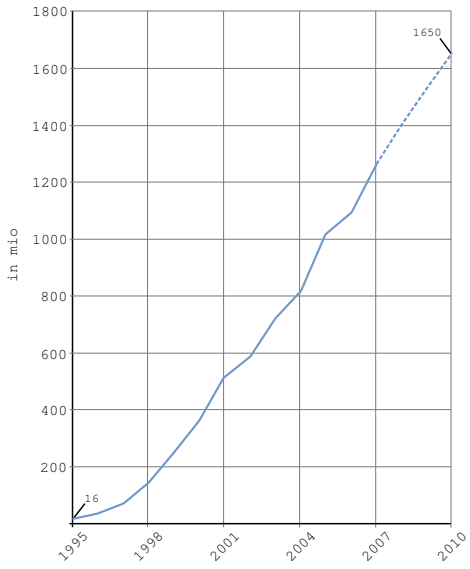

Abbildung 0.1: Internet Verbreitung
in Anlehnung an [MMG08]

Viele Services haben sich in dieser Zeit so etabliert, dass sie heute aus dem Alltagsleben schon gar nicht mehr wegzudenken sind, wie z.B. Online-Banking oder das Abrufen von Zug- /Flugreisezeiten. Es haben sich aber auch völlig neue Ideen und Konzepte etabliert. So kann man auf bei Facebook ein soziales Netzwerk aufbauen und mit über 400 Millionen aktiven Nutzern[Fac10] in Kontakt treten (mehr als Einwohner der USA). Durch die große Bedeutung findet auch gleichzeitig ein fortwährender Ausbau der Infrastruktur statt, wodurch sich die Bandbreite, die den Nutzern zur Verfügung, steht ständig erhöht [ITU09]. Dies bildet die Basis für neue technische Herangehensweisen Angebote über das Internet zu eröffnen, da mehr Daten übertragen werden können. So werden Medien wie Video und Sound immer häufiger in Internet-Anwendungen integriert. Die Videoplattform Youtube[1] z.B. ist weit bekannt und über Seiten wie

[1] www.youtube.com

Ustream[2] ist es möglich sein eigenes "Fernsehen" zu generieren und veröffentlichen. Die größeren Bandbreiten ermöglichen aber nicht nur datenintensivere Medien, sondern bilden auch die Grundlage für komplexe Anwendungen die direkt über den Browser genutzt werden können und welche man früher nur aus dem Desktopbereich kannte. Zum Beispiel bietet die Seite Vektormagic[3] leistungsstarke Grafikfunktionen oder Google Docs Textverarbeitung und Tabellenkalkulation direkt über das Internet.

Durch die breiten Benutzergruppen und die immer vielseitigeren und komplexeren Anwendungen, steigt der Bedarf nach entsprechend hochwertigen grafischen Benutzeroberflächen (engl. Graphical User Interface - GUI) um diese Anwendungen auch zugänglich zu machen. Daraus leitet sich auch das Thema dieser Arbeit ab. *"Entwicklung einer grafischen Benutzeroberfläche für 'Rich Internet Applications'. ..."* wobei eine RIA eine komplexe und reichhaltige Internet-Anwendung bezeichnet (vgl. Abschnitt Abschnitt 1.2). Um solche komplexen Webseiten zu entwickeln werden spezielle Technologien, so genannte RIA-Technologien, genutzt.

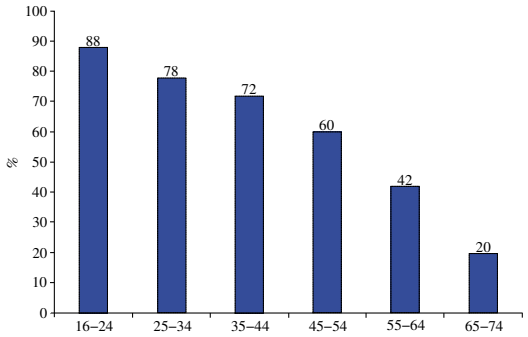

Abbildung 0.2: Internet Nutzer, Alter in Anlehnung an [ITU09]

0.1 Zielstellung

Im Rahmen einer eigenen Firma die im Gebiet der Web-Entwicklung Fuß fassen will, wird in dieser Diplomarbeit eine konkrete RIA-Technologie untersucht, um herauszufinden ob sie als zukünftige Entwicklungstechnologie für ein breites Spektrum von Projekten in Frage kommt. Dabei fiel die Wahl auf Adobe Flex, da dies eine der marktführenden Technologien im Bereich der RIA-Entwicklung ist und für die Ausrichtung der Firma gut geeignet scheint[4] Außerdem wurden bereits erste positive Erfahrungen mit dieser Technologie gesammelt was diesen Entschluss untermauerte. Für die Untersuchung der Leistungsfähigkeit von Flex gab es zwei Hauptziele. Zum einen soll untersucht werden,

[2] www.ustream.tv
[3] www.vektormagic.combietet leistungsstarke Vektorkonvertierungsoptionen
[4] Die Firma wird sich auf die Entwicklung von RIA's spezialisieren

inwieweit eine bewusst gewählte Software-Architektur (hohe Wiederverwendung und Wartbarkeit) in Flex ökonomische Vorteile für ein Software- Unternehmen mit sich bringt. Dieser Thematik widmete sich Mirko Kunath in seiner Diplomarbeit *„Entwicklung einer ‚Rich Internet Application mit Flex unter Verwendung von Architektur-Frameworks“*. Zum anderen ist der Schwerpunkt der vorliegenden Arbeit, herauszufinden, welche Funktionalitäten das Flex-Framework zur Verfügung stellt um eine leistungsstarke und moderne GUI zu entwickeln. Um dies zu ermitteln wird diese Diplomarbeit in zwei Teile aufgeteilt: einen theoretischen (Zielstellung der theoretischen Untersuchungen vgl. Abschnitt 0.1.1) und einen praktischen Teil (Zielstellung für prototypische Anwendung vgl. Abschnitt 0.1.2). Im theoretischen Teil wird das Adobe Flex-Framework untersucht. Darauf aufbauend werden im praktischen Teil die Erkenntnisse aus der Theorie in einer prototypischen RIA unter Anwendung des Flex-Frameworks angewendet und überprüft.

0.1.1 Zielstellung der theoretischen Untersuchung

Die theoretische Untersuchung hat das Ziel die von Flex zur Verfügung gestellten Funktionalitäten zur Entwicklung einer grafischen Benutzerschnittstelle zu untersuchen. Um die Ergebnisse bewerten und einordnen zu können wird eine Vergleichstechnologie genutzt und die Flex-Funktionen relativ zu dieser Technologie beurteilt. Da ein Kernmerkmal von RIA-Anwendungen die leistungsstarke GUI ist und damit eine ähnliche Qualität wie auf dem Desktop erreicht werden soll (vgl. Abschnitt Abschnitt 1.2), wird als Vergleichstechnologie die wohl leistungsstärkste Desktoptechnologie WPF[5] gewählt (Desktop-Framework für GUI-Entwicklung). Dazu wird die theoretische Untersuchung in vier für die RIA-GUI wesentliche Schwerpunkte untergliedert: Steuerelemente und Konzepte, Layout, Animation und States, Styles und Skinning. Für jeden dieser Schwerpunkte wird das Flex-Framework nach seinem gebotenen Leistungsumfang untersucht und anschließend verglichen, welche Möglichkeiten die WPF in diesem Bereich bietet. Daraus resultiert für jedes Kriterium eine abschließende Benotung, welche das Flex-Framework relativ zur WPF einordnet. Mit Hilfe der vier Einzelnoten wird ein Gesamteindruck der Leistungsfähigkeit des Flex-Frameworks im Bereich der GUI-Entwicklung relativ zur WPF dargestellt.

[5]Microsoft liefert regelmässig zwischen den offiziellen .NET-Versionen, Neuerungen über das WPF Toolkit. Dieses Toolkit stellt eine Sammlung von WPF-Features und Komponenten dar. In dieser Arbeit wird die Toolkit-Release Version vom Februar 2010 mit als Teil von WPF betrachtet.

0.1.2 Zielstellung der prototypische Anwendung

Ein Grund für viele Menschen das Internet zu nutzen, ist die Möglichkeit der Echtzeitkommunikation mit anderen Netzteilnehmern. Es gibt verschieden etablierte Programme, die dieses Bedürfnis befriedigen. Weit verbreitete Desktop-Lösungen sind z.B. der Instant-Messanger Icq, das Voice- und Videochat-Programm Skype oder das Chatprogramm Mirc. Durch RIA-Technolgien ist es nun auch möglich über eine Webanwendung in Echtzeit zu kommunizieren. So bietet z.B. die Webseite Meebo[6] eine Plattform auf der verschiedene InstantMessanger online genutzt werden können. Weitere Beispiele sind, die Webseite Mibbit[7] die eine Webvariante des Programms mirc zur Verfügung stellt oder die Webseite Chatroulette[8] welche einen Videochat zu einem zufälligen Gegenüber herstellt. Es sind also verschiedene Formen von Echtzeitkommunikation im Internet möglich und unterschiedliche Einsatzgebiete vorhanden. Aus dieser Überlegung heraus entstand die Idee mit einer Rich Internet Application verschiedene Wege der Echtzeitkommunikation zu ermöglichen. Allerdings mit dem zusätzlichen Anspruch, diese über ein wieder verwendbares Modulkonzept zu implementieren, so dass die einzelnen Teilfunktionalitäten wie z.B. die eines Instant-Messengers, später in eine andere Anwendung eingebettet werden können. Im Ergebnis sollte es leicht möglich sein, Webanwendungen zu entwickeln, die als leicht zu integrierenden Bestandteil, Echtzeitkommunikation anbieten. Dieses Projekt wird gemeinsam mit Mirko Kunath umgesetzt. Der eigene Schwerpunkt bei der Implementierung ist die grafischen Benutzeroberfläche der Anwendung, mit dem Anspruch, eine Vielzahl der von Flex gebotenen Funktionalitäten zu nutzen, um einen praktischen Eindruck der GUI-Entwicklung mit Hilfe des Flex-Frameworks zu gewinnen.

0.2 Aufbau

Nachdem dieses einleitende Kapitel einen Überblick über die Motivation und Zielstellung der Arbeit gegeben hat, werden im Kapitel Kapitel 1 grundlegende Begriffe die für das Verständnis der Arbeit wesentlich sind erklärt. Daraufhin folgt aufgeteilt auf vier Kapitel (Kapitel Kapitel 2 - Kapitel 5) die theoretische Untersuchung des Flex-Frameworks und der zugehörigen Vergleich zu den Funktionen der WPF. Wobei jedes der vier Kapitel

[6] http://www.meebo.com/
[7] http://www.mibbit.com/
[8] http://www.chatroulette.com/

sich einem Schwerpunkt der GUI-Entwicklung widmet (Steuerelemente und Konzepte, Layout, Animationen und States, Styles & Skinning). Für jeden dieser Schwerpunkte (Kapitel) wird zuerst eine Untersuchung der Flex Funktionalitäten durchgeführt. Danach folgt eine Gegenüberstellung zu dem Leistungsangebot der WPF und anschließend die Einordnung der Flex Funktionalitäten relativ zur WPF. Das Ergebnis ist jeweils eine Note am Ende des Kapitels. Daraufhin wird im KAPITEL Kapitel 6 auf die Implementierung des Prototypen eingegangen. Zuerst wird hier ein Überblick über das Projekt gegeben. Danach wird auf den praktischen Schwerpunkt, die Implementierung des Benutzerinterfaces eingegangen. Dazu wird zuerst aufgezeigt welche Erwartungen sich aus der theoretischen Untersuchung des Flex-Frameworks, an den praktischen Entwicklungsprozess, entwickelt haben. Nachfolgend wird an Hand eines Beispiels gezeigt, welche Funktionalitäten des Flex-Framworks genutzt wurden um die Benutzerschnittstelle zu entwickeln. Die implementierten Funktionalitäten werden dabei je einem der vier Schwerpunkte (Steuerelemente und Konzepte, Layout, Animationen und States, Styles und Skinning) zugeordnet um sie später mit den, durch Theorie erarbeiteten, Erwartungen vergleichen zu können. Im abschließenden KAPITEL Kapitel 7 werden die Ergebnisse der theoretischen Untersuchung und der praktischen Arbeit noch einmal zusammengefasst. Daraus wird das Ergebnis dieser Arbeit abgeleitet. Die Arbeit schließt mit einem Ausblick über zukünftige Entwicklungen im RIA-Bereich.

1 Begriffserklärungen

In diesem Kapitel werden grundlegende Begriffe, welche für das Verstädniss dieser Diplomarbeit wesentlich sind, eingeführt. Dabei wird besonders auf die Vorstellung von Adobe Flex Wert gelegt. Außerdem werden nachfolgend die Begriffe WPF und RIA erläutert.

1.1 Adobe Flex 3.0

Da Flex der Kernpunkt dieser Arbeit ist, wird nachfolgend ausführlich darauf eingangen was sich hinter der Technologie „Flex“ verbirgt. Flex ist ein Open-Source Framework zur Entwicklung von Rich Internet Applications der Firma Adobe. Man kann das für die Entwicklung von Flex-Anwendungen benötigte Flex SDK (Software Development Kit) kostenlos herunterladen[1]. Dieses SDK enthält das gesamte Flex-Framework samt umfangreicher Komponenteklassen und dem zugehörigen Compiler zum Erstellen der ausführbaren Anwendungen [Wid08]. Die aktuelle Version ist 3.0, es ist jedoch bereits eine Beta-Verion zur Nachfolgeversion 4.0 (Gumbo) erhältlich. Am Ende dieser Arbeit wird kurz auf die Neuerung dieser Version eingegangen.

1.1.1 Entstehung von Flex

Die Ursprünge von Flex liegen in der verwandten Technologie Flash. Im Jahr 1997 wurde, von Macromedia, die erste Version von Flash veröffentlicht. Es war damals ein Animationswerkzeug und kein professionelles Entwicklerwerkzeug für Enterprise-Anwendungen. Macromedia erkannte darin ein Problem. Man hatte, mit dem Flashplayer, eine Plattform geschaffen die es ermöglichte ein neues Leistungsspektrum im Netz abzudecken, allerdings fehlte es an professionellen Entwicklungswerkzeugen. Flex 1.0 war der Versuch von Macromedia dieses Entwicklungswerkzeug zu liefern um den Sprung

[1] http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3

ins Enterprise Segment zu schaffen, doch der große Erfolg blieb aus. Im Jahr 2005 wurde Macromedia dann von Adobe aufgekauft und es erschien ein Jahr später Flex 2.0. Viele Mängel aus der ersten Version wurden damit beseitigt und es entstand eine professionelle Eclipse basierende IDE. Zusätzlich entstanden auch schon einige GUI-Komponenten. Damals kostete die Serveranbindung jedoch viel Geld so das nur größere Firmen sich Entwicklungen mit Flex leisten konnten. Des Weiteren gab es in Flex 2.0 verschiedene Fehler und nur wenig Referenz-Projekte, so dass sich Flex am Markt nicht richtig durchsetzen konnte. Im Jahr 2008 erschien Flex 3.0. Mit dieser Version wurden eine kostenlose Serverkommunikation und neue Komponenten eingeführt. Außerdem erschien Adobe AIR (Adobe Integrated Runtime), eine Laufzeitumgebung mit der Flex als Desktop-Anwendung betrieben werden kann. Durch diesen Sprung der Flex-Technologie interessierten sich viele Entwickler für Flex 3.0, weshalb sich Flex am Markt etablieren konnte [Mül09].

1.1.2 Der Entwicklungsprozess

Um Flex Anwendungen zu entwickeln stehen zwei Sprachen zur Verfügung. Die deklarative Sprache MXML und die objektorientierte Skriptsprache, Actionscript 3. Für die Entwicklung der Benutzerschnittstelle stellt Flex ein umfangreiches Set an Komponenten zur Verfügung die den eigenen Bedürfnissen angepasst werden können und den Entwicklungsprozess sehr beschleunigen. Der produzierte Code wird zu einer SWF-Datei kompiliert die von dem Flash Player, im Browser ausgeführt werden kann. Von vielen Autoren, wird zum Arbeiten mit Flex, der Flexbuilder als leistungsstarke aber kostenpflichtige Entwicklungsumgebung empfohlen.

Die Entwicklungs-Sprachen

Die Kombination der Sprachen MXML und Actionscript 3 (AS3) ist eine Stärke von Flex. Die Komponenten können über die deklarative Sprache MXML erstellt und die Anwendungslogik dann über ActionScript implementiert werden [Wal09]. Im Folgenden werden MXML und AS3 näher betrachtet.

- MXML - MXML ist eine XML(extensible Markup Language)-basierte Auszeichnungssprache, mit der die von Flex mitgelieferten oder selbst erstellte Komponenten deklariert werden können. Das Verfahren ähnelt HTML- basierten Webseiten, denn

ebenso wie dort mit Hilfe HTML Formulare, Textfelder, Combo-Boxen, Optionsfelder und Schaltflächen beschrieben werden, kann man in Flex Anwendungen, Komponenten, wie z.B. Listen, Buttons oder Textfelder, über MXML beschreiben. MXML stellt einen, für den Entwickler, vereinfachten Zugriff auf die Actionscript-Klassenbibliothek des Flex Frameworks dar. Intern wird der MXML-Code zu Actionscript-Code übersetzt [Wal09]. *„Leider kann an dieser Stelle keine Übersetzung des Akronyms 'MXML' geliefert werden, da diese nicht exisitiert. Oft wird behauptet, 'MXML' stehe für Macromedia-XML – das ist falsch.“ [Mül09]*

- Actionscript 3 - ActionScript ist eine leistungsfähige, objektorientierte Skriptsprache und beruht auf dem ECMA-Script-Sprachstandart[2]. ActionScript besteht aus zwei Teilen, zum einem dem eigentlichen Sprachkern der die eigentliche Sprache definiert und zum anderen aus einer Klassenbibliothek, die vom Flash Player zur Verfügung gestellt wird und die eine Brücke zu den Funktionen des Flash Players darstellt (Flash Player API). ActionScript wird für die Programmierung der eigentlichen Anwendungslogik in Flex verwendet. Um dann aus dem geschriebenen Quellcode, eine nutzbare Anwendung zu erhalten, wird dieser zu Bytecode in eine .swf-Datei (Small Web Format) kompiliert. Diese entstandene .swf-Datei kann dann vom Flash Player ausgeführt werden [Wal09].

Die Entwicklungsumgebung – „der Flexbuilder“

Der Flexbuilder ist die kostenpflichtige, offizielle Entwicklungsumgebung von Adobe zum Entwickeln von Flex-Anwendungen. Er ist eine auf Eclipse basierende IDE (Integrated Development Environment). Man kann ihn entweder als Standalone-Version beziehen oder als Plugin für eine bereits installierte Eclipse-Version. Eigentlich könnte man Flex-Anwendungen auch in einem Texteditor entwickeln doch der Flexbuilder bietet einige Features die, die Effizienz der Entwicklungsarbeit deutlich erhöhen.

- Code Completion und Fehlererkennung
- Integrierter Debugger mit Unterstützung von Breakpoints und Step-through-Code
- WYSIWYG-Editor, grafischer Designmodus mit Eigenschaftsinspektoren (What You See Is What You Get)

[2] http://www.ecma-international.org/

Eine detaillierte Feature Liste findet man auf der Adobe Webseite[3]. Die aktuelle Version (Flexbuilder 3) bietet Adobe in einer Standart- und einer Professional-Edition an. Die Standart Version kostet ca. 200 € während die Professional Version um die 600 € kostet. Die erweiterte Version bietet komplexere Testwerkzeuge und einige zusätzliche Komponenten. Diese zusätzlichen Komponenten umfassen ein erweitertes Datagrid, ein OLAPDatagrid und vor allem einige Charting-Komponenten mit denen verschiedene Formen von Diagrammen dargestellt und interaktiv bearbeitet werden können. Oft wird der Flexbuilder in der Literatur als gutes Werkzeug bezeichnet, welches sein Geld wert ist [Wal09].

Es gibt noch einige alternative Entwicklungsumgebungen für Flex. Zwei Beispiele sind z.B.: „FlashDevelop“[4] und das Visual Studio Plugin: „Amethyst“[5]. Beides sind vollwertige Entwicklungsumgebungen. FlashDevelop weißt allerdings keinen grafischen Editor auf, der für das effiziente Erstellen von Oberflächen sehr hilfreich ist. Amethyst ist aktuell in der Beta 6. Man benötigt dazu Visual Studio. Eine kostenpflichte Variante mit grafischem Editor ist ebenfalls vorhanden.

Flex Komponenten

Flex bietet eine große Anzahl vorgefertigter Komponenten an (vgl. Kapitel Kapitel 2). Dabei gibt es einen Teil der der Organisation und Anordnung von anderen Komponenten dient. Die Palette reicht hier vom einfachen Canvas welches einfach einen leeren Container darstellt, bis hin zum Akkordion-Container in dem der Nutzer zwischen verschiedenen Flächen umschalten kann die der Entwickler jeweils nach Bedarf füllen kann. Der andere Teil bietet Lösungen für oft anfallende Problemstellungen. So z.B. Buttons, Checkboxen, Colorpicker, Textfelder oder Listen. Es gibt eine recht große Auswahl und bei Bedarf kann man die Komponenten auch durch Ableitung erweitern.

1.1.3 Flash Player

Der Adobe Flash Player ist ein kostenloses Webbrowser-Plugin über den die .swf-Dateien ausgeführt werden können. Er kann auf der Adobe-Webseite[6] herunter geladen werden.

[3] http://www.adobe.com/products/flex/features/flex_builder/
[4] http://www.flashdevelop.org/wikidocs/index.php?title=Features:Interface#Features_Tour
[5] http://www.sapphiresteel.com/Amethyst-Product-Page
[6] http://www.adobe.com/

Der Player wird von den gängigen Betriebssystem und Browsern unterstützt. Es gibt unterschiedliche Statistiken zu der Verfügbarkeit des Players. Laut Adobe beträgt die Verbreitung der aktuellen Version 10 über 95% [Ado10]. Um Flex Anwendungen ausführen zu können, wird mindestens die Version 9 oder höher benötigt. Der Flash Player (ab Version 9) enthält zwei Virtual Machines: AVM1 und AVM2 (AVM – ActionScript Virtual Machine). Während die AVM1 nur noch Kompatibilitätszwecken dient, um ältere Flash Anwendungen ausführen zu können, ist die neue AVM2 das Kernstück, welches für Flex-Anwendungen benötigt wird. Sie verarbeitet den kompilierten Bytecode der Flex-Swf's und übersetzt diesen in Maschinencode [Rüt09].

1.2 Rich Internet Applications (RIA)

Um er zu klären was eine Rich Internet Application ist, soll gezeigt werden wie RIA's entstanden sind. Ursprünglich war das Internet darauf ausgerichtet einfache Dokumente und Informationen in Form von statischen HTML-Seiten zu transportieren. Diese Form der Webseite findet man heute kaum noch. Daraus entstand die Technik HTML-Seiten serverseitig zu generieren um so dynamische Inhalte zu ermöglichen. Später wurden diese Webseiten durch Content-Managementsystemen unterstützt. Die Seiteninhalte konnten auf diese Weise komfortabel variiert werden. Allerdings war die Art des Nachladens dynamischer Inhalte aus heutiger Sicht uneffizient implementiert, denn durch jede Interaktion eines Anwenders mit solch einer Webseite wurde eine Anfrage an den Server geschickt und eine komplett neue HTML Seite generiert und geladen. Eine RIA-Anwendung hingegen hat den Vorteil, dass Logik direkt auf der Seite des Benutzers (Client) ausgeführt werden kann. Durch diesen Ansatz ergeben sich völlig neue Möglichkeiten. Zum Beispiel ist es einer RIA-Anwendung auch ohne Nutzerinteraktion möglich mit Servern zu kommunizieren um so z.B. Daten zu aktualisieren. Ein weiteres wesentliches Merkmal einer RIA-Anwendung ist es das nicht immer die gesamte Seite nachgeladen werden muss sondern einzelne Teilbereiche aktualisiert werden können was ein viel breiteres Spektrum an Reaktionen auf Nutzereingaben erlaubt. Durch diese Möglichkeiten können hochinteraktive Benutzeroberflächen generiert werden, welche ein völlig neues Erleben von Webanwendungen ermöglichen. Im Hinblick auf die Nutzungsqualität (Usability) verschwimmen dadurch zunehmend die Grenzen zwischen Desktop- und Web-Bereich. Zusätzlich können in RIA-Anwendung Videos, Sound und Echtzeit-Funktionalität integriert werden. Diesen

reichhaltigen Medienangeboten in Verbindung mit der reichhaltigen Möglichkeiten zur Erstellung von Nutzeroberflächen verdankt dieser Typ der Webanwendung ihren Namen „reichhaltige Internet Anwendung“ (Rich Internet Application). Zur Entwicklung solcher Anwendung gibt es spezielle Technologien, RIA-Technologien. Drei von diesen (Adobe Flex, Microsoft Silverlight, Ajax) können dabei als marktdominierend bezeichnet werden. Für weiterführende Informationen, bietet die Diplomarbeit von Stephanie Strauß [Str08] einen ausführlichen Vergleich aktueller RIA-Technolgien.

Frau Strauß fasst RIA vereinfacht aber dennoch treffend zusammen: „Rich Internet Applications bieten das Potenzial, die Bedienbarkeit im Web erheblich zu steigern, indem sie die Vorteile von Webanwendungen mit denen herkömmlicher Desktopanwendungen verbindet“ [Str08].

1.3 Windows Presentation Foundation (WPF)

WPF (Windows Präsentation Foundation) ist ein Grafik-Framework von Microsoft. Es ist im .NET 3.0 Framework und höher enthalten und damit derzeit lediglich unter Windows einsetzbar. Die WPF wird in dieser Diplomarbeit als Vergleichsgröße zu Flex 3.0 dienen da sie ein mächtiges Werkzeug zur UI (User Interface)–Entwicklung ist. Flex wird in Kapitel Kapitel 2 bis Kapitel Kapitel 5 an dieser Technologie gemessen und bewertet.

1.3.1 Die WPF im .NET Framework

Das .NET Framework ist eine Software-Plattform von Microsoft. Es enthält eine Laufzeitumgebung und verschiedene API's (Application Programming Interface) für Softwareentwickler. Durch die Laufzeitumgebung CLR (Common Language Runtime) ist Plattformunabhängigkeit grundsätzlich möglich, Microsoft hat sich jedoch auf ihr eigenes Betriebssystem „Windows“ konzentriert. Es gibt Projekte wie z.B. Mono die es ermöglichen sollen .NET Anwendungen auf anderen Plattformen wie Linux oder Mac OS zu entwickeln. Der Entwicklungsstand ist jedoch meist hinter dem aktuellen .NET Framework zurück und eine Implementierung von WPF ist zum derzeit gar nicht abzusehen.

Ende 2006 wurde das .NET Framework 3.0 veröffentlicht. Es besteht aus den Komponenten des .NET Frameworks 2.0 und vier weiteren Hauptbestandteilen[7].

[7] Einen Überblick über die anderen Bestandteile findet man bei Frischalowski [Fri07]

- WPF (Windows Präsentation Foundation)
- WCF (Windows Communication Foundation)
- WF (Windows Workflow Foundation)
- WCS (Windows Card Space)

Diese Hauptbestandteile waren bereits in der erste Version der WPF enthalten [Fri07, Sch07]. Die Versionsnummer war allerdings nicht 1.0 sondern sie wurde der .NET Version angepasst, 3.0. Ende 2007 kam dann die Version 3.5 des .NET Frameworks auf den Markt und enthielt WPF 3.5 was die aktuellste Version darstellt. Dezeit ist bereits eine Beta für die Version 4.0 verfügbar. Mit dem Release wird noch in diesem Jahr (2010) gerechnet. Die WPF wird von den Windows-Systemen Vista und Windows 7 standardmäßig unterstützt da bei diesen Betriebssystemen das .NET Framework bereits enthalten ist. Auf Windows XP und Windows Server 2003 muss jeweils das entsprechende .NET Framework nachträglich installiert werden.

1.3.2 Merkmale der WPF

Die WPF ist ein umfangreiches Framework zum Erstellen von grafischen Benutzeroberflächen [ALMvV07]. Dabei kann die WPF als Nachfolger des Vorgängermodels Windows Forms[8] betrachtet werden. Wobei Windows Forms Komponenten weiterhin in der WPF genutzt werden können. Zusätzlich bringt WPF aber ein neues eigenes Komponenten-Set mit. Ein großer Fokus dieses Frameworks ist die Trennung der Präsentationsebene und der Geschäfts-Logik [Hub08]. Die Oberflächen können dazu mit der XML-basierten Auszeichnungssprache XAML (eXtensible Application Markup Language) definiert werden. Zur Implementierung der Programmlogik steht einem jede .NET kompatible Programmiersprache (z.B. C# oder Visual Basic) zur Verfügung. Technisch bietet das WPF-Framework viel Innovation und versucht die verschiedenen Bereiche (z.B. Entwicklung der Benutzerschnittstelle, Audio und Video, Dokumente) die für die Darstellung von Anwendungen wichtig sind zu vereinen. Die WPF dient hauptsächlich der Entwicklung von Windows-Desktop-Anwendungen jedoch gibt es die zusätzliche Option eine Web-Anwendung zu generieren (XBAP, XAML Browser Applications). Diese Art der Anwendung setzt je-

[8] Windows Forms ist eine mit .NET 1.0 eingeführte Bibliothek zur Erstellung von Benutzeroberflächen.

doch das .NET Framework auf dem ausführenden System voraus und ist damit nicht plattformunabhängig, was ein wichtiges Kriterium für eine Web-Anwendung darstellt.

2 Untersuchung und Vergleich der Steuerelemente und Konzepte

Beide Frameworks (Flex, WPF) bieten ein Set an vorgefertigten Komponenten an die den Entwicklungsprozess des GUI vereinfachen sollen. In diesem Kapitel wird der Umfang und die Leistungsfähigkeit dieser Steuerelemente untersucht. Zusätzlich wird aufgezeigt wie diese grundlegend implementiert werden können und welche Konzepte die Nutzung dieser Steuerelemente unterstützen (vgl. Abbildung 2.1). Dieses Kapitel wird mit einem Vergleich zur WPF abgeschlossen (vgl. Abschnitt 2.3).

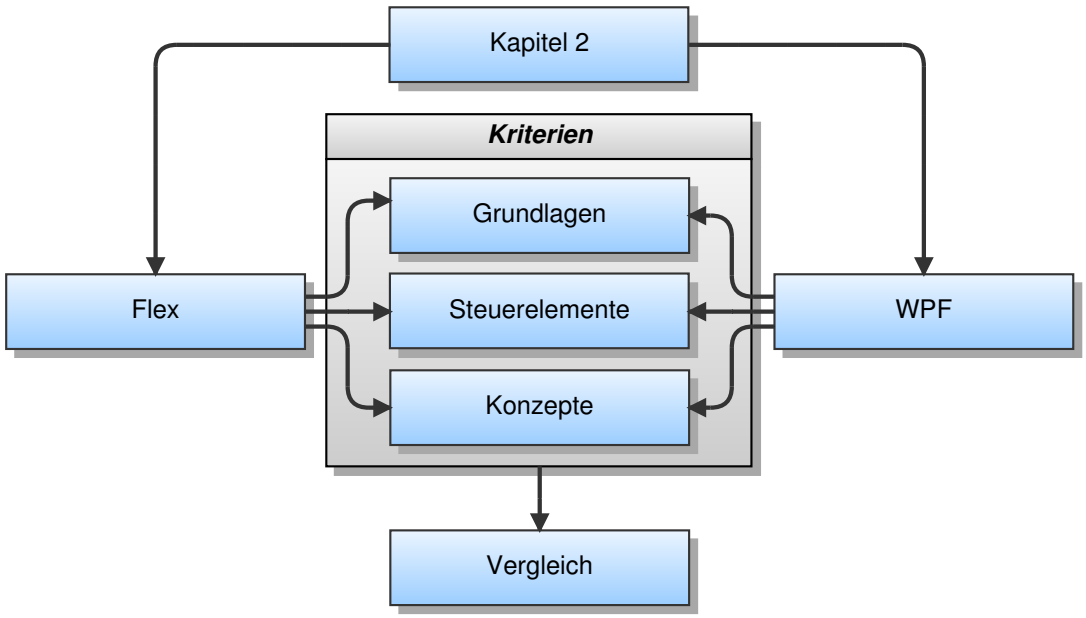

Abbildung 2.1: Ablauf und Kriterien Kapitel 2

2.1 Steuerelemente und Konzepte in Flex

Das Flex-Framework bietet eine Vielzahl von Steuerelementen, die genutzt werden können, um effizient Benutzeroberflächen zu gestalten. Hierzu gehören sowohl einfache Steuerelemente, wie Textfelder, Schaltflächen, Wertselektoren, als auch komplexe Steuerelemente, wie Listen und DataGrids. In diesem Kapitel wird aufgezeigt, welche Steuerelemente Flex bietet und es werden einige Konzepte vorgestellt, die den Entwickler bei der Arbeit mit diesen Elementen unterstützen [Wal09].

2.1.1 Grundlagen

Flex bietet als Entwicklungssprachen MXML und Actionsscript an. Dabei soll MXML den Zugriff auf bestimmte Actionsscript-Klassen erleichtern und dient hauptsächlich dazu das Layout und die statischen Komponenten einer Flex-Anwendung zu beschreiben. In der Regel ist es einfacher, übersichtlicher und schneller eine Komponente mit Hilfe von MXML anzulegen als dafür ActionScript zu verwenden. Im Laufe dieser Arbeit wird MXML genutzt wenn es möglich und sinnvoll erscheint. Es ist jedoch immer eine Option die gleiche Funktionalität auch über ActionScript zu implementieren [Wal09].

Listing 2.1 Anlegen eines Buttons mit MXML

```
<mx:Button label="Hallo" id="myButton1"/>
```

Listing 2.2 Anlegen eines Buttons mit Actionscript 3.0

```
var myButton1:Button = new Button();
myButton1.label="Hallo";
```

Listing 2.1 und Listing 2.2 zeigen wie die gleiche Schaltfläche einmal über MXML und einmal über ActionScript angelegt wird. Die Klasseneigenschaften, in diesem Fall der Button Klasse, werden im MXML über Tag-Attribute zugewiesen. Über die Attribute kann eine Vielzahl von Einstellungen vorgenommen werden. So kann man nicht nur einfache Eigenschaften wie X-,Y-Koordinaten oder Höhe und Breite einstellen. Es ist auch möglich Effekte zuzuweisen (vgl. Kapitel Kapitel 4), Stile festzulegen (vgl. Kapitel Kapitel 5) oder Reaktionen auf Events zu bestimmen (vgl. Abschnitt 2.1.4).

2.1.2 Steuerelemente in Flex

Nachdem gezeigt wurde, wie ein Steuerelement anlegt wird, soll gezeigt werden, welche Komponenten Flex für die grafische Benutzeroberfläche zur Verfügung stellt. Diese Komponenten werden in Untergruppen unterteilt und erklärt (vgl. Abbildung 2.2). Dabei wird auf die Steuelemente der Gruppen Schaltflächen, Text-Steuerelemente, Wertselektoren und Helfer nur kurz in Listenform eingegangen, da diese recht einfach und oft selbsterklärend sind. Die Elemente aus den Gruppen Medienkomponenten und Listkomponenten werden eingehender betrachtet da sie komplexere Funktionen anbieten.

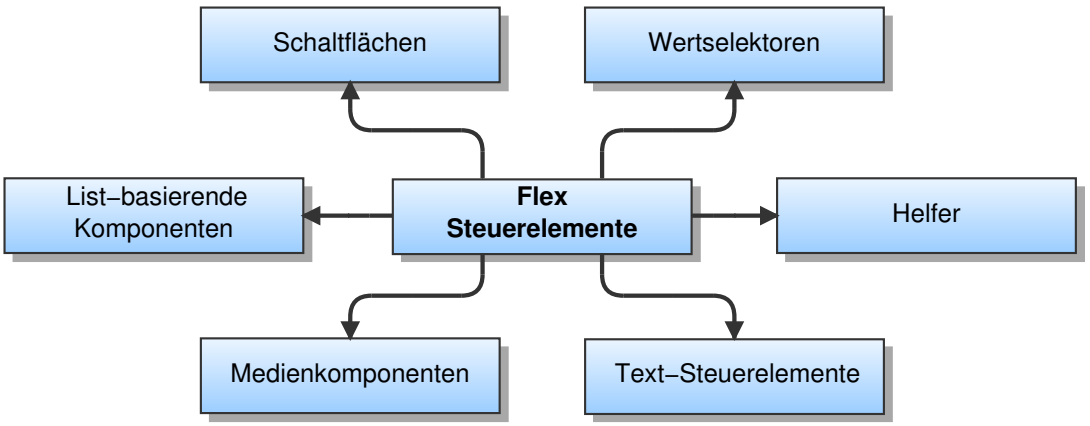

Abbildung 2.2: Flex Steuerelemente

Schaltflächen

Button	Der Button ist eine einfache rechteckige Schaltfläche, die mit Beschriftung und Icon versehen werden kann.
CheckBox	Die Checkbox ist ein Feld, das selektiert oder deselektiert werden kann.
LinkButton	Der LinkButton hat eine ähnliche Funkionalität wie der Button. Visuell wird er aber ähnlich einem Hyperlink dargestellt.
RadioButton	Der RadioButton ist ebenfalls eine selektierbare Schaltfläche. Er dient aber der Auswahl eines Wertes aus mehreren möglichen Werten.

Tabelle 2.2: Flex 3.0 Schaltflächen

Wertselektoren

ColorPicker	Der ColorPicker bietet die Option, zur Laufzeit eine Farbauswahl aus einem Sortiment von Farben zu treffen.
ComboBox	Die ComboBox ist eine Drop-down-Liste mit beliebigem Inhalt, aus der der Anwender eine Auswahl treffen kann.
DateChooser	Der DateChooser stellt einen Kalender dar, in dem man ein Datum auswählen kann.
DateField	Das DateField ist ein Eingabefeld für ein Datum. Zur Auswahl erscheint der DateChooser als Popup.
HSlider	Ein horizontaler Auswahlbalken, mit dem der Anwender, einen Wert aus einem einstellbaren Wertebereich einstellen kann.
NumericStepper	Der NumericStepper bietet die Möglichkeit, eine Zahl aus einem Zahlenpool auszuwählen.
VSlider	Der VSlider entspricht dem Hslider, nur ist dieser vertikal ausgerichtet.

Tabelle 2.4: Flex 3.0 Wertselektoren

Helfer

HRule	Eine einfache horizontale Linie.
ProgressBar	Der ProgressBar ist die visuelle Darstellung eines Fortschrittprozesses.
Spacer	Der Spacer ist ein unsichtbarer Platzhalter.
Vrule	Eine einfache vertikale Linie.

Tabelle 2.6: Flex 3.0 Helfer

Text-Steuerelemente

In der Kategorie der Text-Steuerelemente befinden sich die Komponenten für Texteingaben. Diese finden in den meisten Anwendungen gebrauch. Flex bietet zwei unterstützende Lösungen für Problemstellungen an, die im Zusammenhang mit Texteingaben häufig auftreten. Diese Lösungen sind sog. Formatter und Validatoren. Formatter bieten die Möglichkeit Texteingaben in ein bestimmtes Format zu bringen. Das kann z.B. ein korrektes Datum oder ein gültiger Geldbetrag sein. Flex bietet einige Standardformatter an,

es ist aber auch möglich, eigene zu schreiben. Die Funktionsweise ist dabei wie folgt: dem Formatter kann ein Datenobjekt übergeben werden und dieser liefert eine entsprechend formatierte Zeichenkette zurück. Validatoren hingegen sollen für eine valide Eingabe sorgen. Es ist möglich, einen eingegebenen Wert auf ein bestimmtes Muster zu prüfen. Wird eine Eingabe gemacht und entspricht nicht dem Muster, so wird das Eingabefeld rot umrandet. Wenn der Anwender nun mit der Maus über das umrandete Eingabefeld fährt, erscheint ein roter Tooltip mit einem Hinweis, weshalb die Eingabe ungültig ist. Auch hier bietet Flex bereits einige Standard-Validatoren an, es ist aber wiederum möglich, eigene zu erstellen. Standardvalidatoren sind z.B.: EmailValidator und DateValidator [BEH+09].

Label	Das Label ist ein einfaches Textfeld, das sich zur Beschriftung eignet. Nur einzeilige Textdarstellung möglich.
RichTextEditor	Der RichTextEditor ist ein kleiner Texteditor. Es kann Text eingegeben und formatiert werden. Zum Funktionsumfang gehören: Schriftart, Schriftgröße, Schriftstil, Farbe, Textausrichtung, Links.
Text	Die Text-Komponente kann einen mehrzeiligen, nicht editiebaren Text darstellen.
TextArea	Die TextArea ermöglicht die Darstellung von mehrzeiligem Text, der durch den Nutzer editierbar ist.
TextInput	Der TextInput ist ein einzeiliges Texteingabefeld.

Tabelle 2.8: Flex 3.0 Text Steuerelemente

Medienkomponenten

Image	Die Image-Komponente bietet die Möglichkeiten Grafiken zu laden und darzustellen.
SWFLoader	Der SWFLoader ist der Image-Komponente sehr ähnlich und ermöglicht ebenfalls das Laden und Anzeigen von Grafiken.
VideoDisplay	Mit dem Video-Display können .flv Filme abgespielt werden.

Tabelle 2.10: Flex 3.0 Medienkomponenten

- Image und SWFLoader - Diese beiden Komponenten sind fast identisch[1].Beide bieten die Möglichkeit, Grafiken und .swf Dateien zu laden und anzuzeigen. Geladen werden die Dateien, indem eine Zuweisung auf das *„source“* Attribut stattfindet. Dieses Attribut erwartet eine relative oder absolute URL auf die Grafik-Datei. Beim Laden der Dateien gibt es noch eine Besonderheit. Eine mögliche Einstellung ist, die Mediendatei nicht erst zur Laufzeit zu laden, sondern sie direkt mit in die Anwendung hinein zu kompilieren. Dieser Vorgang wird als „einbetten“ bezeichnet. Dadurch ist die Grafik zwar zur Laufzeit sofort verfügbar, der Anwendungsladevorgang wird aber verlängert. Vektorgrafiken im SVG-Format können sogar nur eingebettet benutzt werden[BEH+09]. Listing 2.3 zeigt das Laden einer Datei einmal normal und einmal eingebettet. Falls sehr große Grafiken geladen werden, gibt es die Möglichkeit, den Fortschritt des Ladeprozesses, mit Hilfe von Events zu überwachen und dem Anwender bei Bedarf ein Feedback zu geben.

Listing 2.3 Laden einer Grafik, normal und eingebettet

```
<mx:Image source="assets/Bild.jpg" />
<mx:Image source="@Embed('assets/Bild.jpg')" />
```

- VideoDisplay - Das Video Display ermöglicht das Abspielen von .flv Dateien. Genau wie bei den Grafik Komponenten erwartet das Attribut *„source“* dabei die URL des Videos. Das Video kann gestreamt werden wenn es sich auf einem entsprechendem RTMP-Server befindet. Alternativ findet ein progressiver Download statt[2]. Das VideoDisplay ist eine reine Anzeige-Komponente. Damit der Nutzer interaktiv in den Abspielvorgang des Videos eingreifen kann müssen dafür Steuerelemente implementiert werden. Das VideoDisplay kann dabei über die Methoden: play(), pause(), stop() angesteuert werden. Die Lautstärke ist über die *„volume“* Eigenschaft regelbar.

[1] Image liefert einige zusätzliche Eigenschaften und Ereignisse, die in Item-Renderern und Item-Editoren benötigt werden. Diese ermöglichen die Anzeige von Grafiken innerhalb von List-basierten Komponenten.

[2] Beim Streamen werden die Daten heruntergeladen die als nächstes zur Anzeige benötigt werden. Man kann also das Video bereits anschauen obwohl es noch nicht vollständig geladen wurde. Beim progressiven Download wird erst das ganze Video heruntergeladen und dann abgespielt.

Flex ermöglicht die Nutzung verschiedener externer Mediendateien. Die Medienkomponenten bieten die Möglichkeit diese Dateien (vgl. Tabelle 2.12) zu laden und zu nutzen.

Grafiken	*.gif, *.jpg, *.png, *.svg(nur embedet)
Flash	*.swf
Audio	*.mp3
Video	*.flv

Tabelle 2.12: Flex 3.0 Medienunterstützung

Für das Abspielen von Sound bietet Flex keine eigene Komponente. Man kann entweder auf die Klasse SoundEffect zugreifen (eine Beschreibung zu Effekten erfolgt im Kapitel Kapitel 4) oder man benutzt zum Abspielen Klassen der Flash API (flash.media.Sound)[Wal09].

List-basierte Komponenten

Die list-basierten Komponenten (vgl. Tabelle 2.14) spielen in Flex eine wichtige Rolle. Diese Komponenten dienen der Visualisierung und der Auswahl von Daten. Nachfolgend sollen die einzelnen Komponenten näher betrachtet werden. Anschließend wird gezeigt, wie man List-basierte Komponenten mit Daten füllen kann und wie Inhalte selektiert werden.

DataGrid	Das DataGrid ist vergleichbar mit der List-Komponente, kann jedoch mehr als eine Spalte darstellen.
HList	Die HList-Komponente ermöglicht eine horizontale aufgelistete Anzeige von Daten.
List	Die List-Komponente ermöglicht eine vertikal aufgereihte Anzeige von Daten.
TileList	Die TileList-Komponente ermöglicht die Anzeige von Daten in einer Gitter Anordnung.
Tree	Die Tree-Komponente ermöglicht die Darstellung von hierarchischen Daten in einer Baumstruktur.

Tabelle 2.14: Flex 3.0 List-Basierte Komponenten

- List, HList, TileList - Die List-Komponente ermöglicht es, eindimensionale Datenreihen darzustellen. Diese werden untereinander angezeigt. Bei Bedarf erscheinen Scroll-Balken. Die beiden anderen Listen, HList und TileList, bieten die gleiche Funktionalität an, stellen die Daten jedoch anders dar. HList ordnet die Datensätze horizontal an, während die TileList die Datensätze fortlaufend in Zeilen anordnet.

- Tree - Die Tree-Komponente ermöglicht es, Daten hierarchisch anzuzeigen. Dazu wird eine Art Ordner-Struktur dargestellt. Es ist möglich diese Ordner zu öffnen und zu schließen, wodurch jeweils die Kind Elemente ein- oder ausgeblendet werden.

- DataGrid - Die DataGrid-Komponente ist eine der komplexesten Komponenten innerhalb des Flex-Frameworks, da sie umfangreiche Funktionalität bietet. Sie dient der Darstellung von mehrdimensionalen Datenreihen in einer Tabelle. Diese Komponente ist gut geeignet, um Daten darzustellen, die aus relationalen Datenbanken bezogen wurden. Innerhalb der Komponente hat man verschiedene Möglichkeiten, die Anzeige anzupassen. Spalten können zur Laufzeit, in der Breite angepasst oder in anderer Reihenfolge angeordnet werden. Außerdem ist es möglich, die Spalten auf- oder absteigend zu sortieren.

Die Zuweisung von Daten an eine List-basierte Komponente erfolgt über das *„dataProvider“*-Attribut. Dieses Attribut steht allen List-basierten Steuerelementen zur Verfügung und erwartet ein Objekt welches das Interface mx.core.IcollectionView implementiert haben muss. In der Regel ist das ein ArrayCollection- oder ein XMLListCollection-Objekt. Diese Objekte sind einem Array nicht unähnlich. Es wäre alternativ auch möglich z.B. ein Array zu übergeben. Allerdings hat dies den Nachteil, das ein Array keine Funktionalität für Data Binding (vgl. Abschnitt 2.1.4) bietet, welches für die automatische Aktualisierung der Anzeige, bei Änderung der Daten verantwortlich ist. Die Änderungen der Daten findet also nur in dem als *„dataProvider“* übergebenem Objekt statt, wie zum Beispiel, die Sortierung, das Hinzufügen oder das Löschen von Datensätzen. Das List-basierte Steuerelement spiegelt einfach dieses Daten-Objekt wieder, indem es die Daten visualisiert. Allerdings kann man noch großen Einfluss auf diese Visualisierung der Daten nehmen. Grundlegend ist es möglich, auszuwählen welche der Daten angezeigt werden sollen und welche nicht. So genannte *„label“* Funktionen bieten außerdem die Option, ankommende Daten in einen beliebigen Ausgabestring umzuwandeln. Spezielle ItemRenderer können zusätzlich Bilder oder sogar Steuerelemente wie Checkboxen

innerhalb eines List-basierten Steuerelements darstellen [Wid08].

Innerhalb einer List-basierten Komponente kann man angezeigte Datenelemente durch einen einfachen Mausklick auswählen. Die Komponenten bietet dem Entwickler die drei Attribute, *„selectedLabel“*, *„selectedIndex“* und *„selectedItem“*, um Informationen über das gewählte Objekt zu erhalten. Dabei enthält *„selectedLabel“* den in der Komponente dargestellten String. Die Eigenschaft *„selectedIndex“* gibt den Index innerhalb des zugehörigen Daten-Objekts zurück. Und *„selectedItem“* enthält eine Referenz auf das gesamte ausgewählte Objekt. Eine zusätzliche Option bei der Auswahl eines Datenelements ergibt sich, wenn die Eigenschaft *„editable“* der Komponente auf „`true`“ gesetzt wird. In diesem Fall wird der ausgewählte Eintrag vorübergehend in einem Textfeld dargestellt, wo er direkt bearbeitet werden kann. Mit Hilfe von ItemEditoren ist es zusätzlich möglich, die Daten nicht nur über eine Texteingabe in einem Textfeld zu ändern, sondern über ein anderes zugewiesenes Steuerelement. Zum Beispiel wäre es dadurch möglich, die Stückzahl in einer Bestellübersicht, über einen NumericStepper zu editieren.

2.1.3 Zusätzliche Komponenten

Flex bietet eine breite Palette an Komponenten an. Diese können natürlich trotzdem nicht den gesamten Bedarf abdecken, der sich in den verschiedenen realen Projektentwicklungen ergibt. Es gibt mehrere Möglichkeiten in diesem Fall, zusätzliche Komponenten zu erhalten. Zum Einen liefert Adobe mit dem Flexbuilder Professional einige zusätzliche Komponenten. Diese Flexbuilderversion bietet zwei erweiterterte DataGrids und eine ganze Reihe von Charting-Komponenten an. Die Charting-Komponenten ermöglichen eine grafische Darstellung von Daten in Form von unterschiedlichen Diagrammen. Eine andere Option ist das Internet. Es gibt viele Webseiten, die verschiedene, von Nutzern entwickelte, Komponenten zur Verfügung stellen. Falls die dort erhältlichen Komponenten nicht ausreichen, müssen eigene Komponenten entwickelt werden. Da Flex Open-Source ist, ist es möglich die Funktionsweise der bestehenden Komponenten genau zu untersuchen. Dies erleichtert das Ableiten bestehender Komponenten und das Entwickeln eigener Komponenten.

2.1.4 Unterstützende Konzepte und Funktionalitäten

Flex bietet noch einige Konzepte und Funktionen die das Entwickeln einer grafischen Benutzerschnittstelle erleichtern. An dieser Stelle soll kurz auf das Event-System, das Databinding und die Unterstützung von Drag and Drop Funktionalität eingegangen werden.

- Event-System - Flex beinhaltet ein Event-System. Events sind ein wesentlicher Grundbaustein interaktiver Benutzeroberflächen. Das Event-Modell in Flex basiert auf dem Document Object Model Level 3 Event Model. Dies ist ein Standart des W3C Konsortiums. An dieser Stelle soll nur kurz darauf eingegangen werden, da eine genauere Beschreibung des Event-Systems ein eigenes Kapitel in Anspruch nehmen würde. Es gibt in Flex eine Vielzahl von Events, die von den Komponenten ausgelöst werden. Die komplette Liste der Events ist in der Flex 3 Language Reference[3] angeführt. Es ist auch möglich eigene Event-Klassen zu erstellen. Wenn ein Event ausgelöst wird ist es möglich, auf dieses Event zu lauschen, um dann entsprechend darauf zu reagieren. Es ist jedoch nicht nur möglich, auf dieses Event konkret bei der auslösenden Komponente zu lauschen, sondern es gibt auch die Option, dieses Event in den darüber liegenden Containern zu empfangen. Dazu bietet das Flex-Framework beim Suchen nach EventListnern eine Capturing- und eine Bubbling-Phase. Bei der Suche nach Listnern, die auf ein bestimmtes Event lauschen, werden in diesen Phasen auch die übergeordneten Ordner durchsucht. Dies ist wichtig für den Fall das man z.B. einen Mausklick auf ein Objekt registrieren möchte. Dieses Objekt besitzt aber noch Kind-Elemente und der Mausklick wird auf einem der Kind-Elemente ausgeführt. Man möchte diesen Mausklick aber als Event auf dem übergeordneten ursprünglichen Objekt registrieren [Wid08].

- Databinding - Databinding ist ein zentraler Bestandteil im Umgang mit Daten innerhalb einer Flex-Anwendung. Databinding bietet die Möglichkeit, die Eigenschaft eines Objekts mit der Eigenschaft eines zweiten Objekts zu verbinden. Die Eigenschaft des ersten Objekts übernimmt nun den Wert der Eigenschaft des zweiten Objekts. Ändert sich nun der Wert des zweiten Objekts wird automatisch eine Aktualisierung bei der gebundenen Eigenschaft durchgeführt. Im Ergebnis übernimmt die Eigenschaft des ersten Objekts diesen Wert, sodass beiden Werte immer

[3] http://livedocs.adobe.com/flex/3/langref/

identisch sind. Mit dieser Methode kann der Datenfluss innerhalb der Anwendung stark vereinfacht werden, da bei einer Änderung nicht alle betroffenen Objekte manuell informiert werden müssen [ALMvV07].

- Drag and Drop - Flex bietet dem Entwickler eine breite Palette an Möglichkeiten, um Drag and Drop-Operationen in die Anwendung zu implementieren. Eine sehr einfach umzusetzende Form bieten die List-basierten Steuerelemente an. Dort ist es möglich, bereits integrierte Drag and Drop Funktionalität zu aktivieren, indem man die Attribute *„dragEnabled“* und *„dropEnabled“* auf „`true`" setzt. Hierbei aktiviert *„dragEnabled“* das Herausziehen und *„dropEnabled“* das Empfangen von Objekten.Es ist dann direkt erlaubt, Objekte aus einer „Liste“, in eine andere, mit Hilfe von Drag and Drop, zu kopieren. Dabei wird das „gezogene“ Objekt dem *„dataProvider“*-Objekt des empfangenden List-basiertem Steuerelements hinzugefügt. Dieser Prozess wird dabei für den Nutzer visuell unterstützt. So zeigen kleine Symbole an, wo es erlaubt ist etwas „fallen zu lassen“ und wo nicht. Ausserdem wird das „transportierte“ Objekt an den Mauszeiger geheftet. Dies ist die einfachste Form von Drag and Drop in Flex. Es ist aber auch möglich, diese Funktionalität für nicht List-basierte Steuerelemente umzusetzen. Dies erfordert dann mehr Aufwand, z.B. muss auf die einzelnen Drag and Drop-Events gelauscht werden und entsprechende Funktionalität dann mit Hilfe des DragManagers implementiert werden, aber Flex bietet auch in diesen Fall hilfreiche Optionen an [KL08].

2.2 Steuerelemente und Konzepte in der WPF

2.2.1 Grundlagen

Die WPF bietet ähnlich wie in Flex ein Konzept zur Trennung von Code und Design. Zum Beschreiben der Komponenten bietet die WPF, die XML basierte Beschreibungssprache XAML (Extensible Application Markup Language). Zum implementieren der Programmlogik stehen der WPF die verschiedenen Programmiersprachen des .NET Frameworks zur Verfügung. Typisch sind C# und Visual Basic.

2.2.2 Steuerelemente

Auch die WPF stellt umfangreiche Komponenten zur effizienten Entwicklung einer Benutzerschnittstelle zur Verfügung. Die unterschiedlichen Typen von Steuerelementen des Flex-Frameworks finden sich im wesentlich auch in der WPF wieder. Einige, wie z.B. das DataGrid, wurden allerdings erst über das WPF Toolkit nachgereicht. Umgekehrt bietet die WPF allerdings zusätzliche Steuerelementen an. Außerdem kann man bei den Steuerelementen oft Konzepte und Funktionalitäten erkennen, die über die Leistungsfähigkeit der Flex-Komponenten hinausgehen [Hub08]. Im Folgenden soll ein Ausblick auf die Unterschiede gegeben werden.

Funktionale Unterschiede bei den Steuerelementen

Da die Steuerelemente in Flex und in der WPF völlig unterschiedlich implementiert sind und es vielfältige Abweichungen gibt soll hier an einigen repräsentativen Beispielen gezeigt werden, wo die WPF gegenüber Flex konkrete zusätzliche Komponentenfunktionalität bietet.

- Flexibler Inhalt - Während z.B. einem Button in Flex als Inhalt, nur Text und optional mit Icon zur Verfügung steht, verfolgt die WPF das Konzept des flexiblen Inhalts. So besitzt der Button in der WPF ein Attribut *„`Content`“*, dem ein beliebiges Objekt zugewiesen werden kann, welches dann als Inhalt des Buttons dargestellt wird. Andere Komponenten, die dieses Konzept ebenfalls verfolgen sind z.B. Expander, ToolTip oder ComboBox.

- Textdarstellung - Beide Frameworks bieten einen RichTextEditor an. Dieser bietet einige Standardformatierungen wie Schriftart auswählen, kursive Darstellung etc.. Die Variante der WPF, bietet hier zusätzlich eine implementierte Undo- und Redo-Funktionalität an. Ausserdem ist die Darstellung der Inhalte viel flexibler. Es ist mit Hilfe der WPF-Variante möglich, ein Objekt der Klasse FlowDocument darzustellen. Die FlowDocument-Klasse dient der Anzeige umfangreicher und komplexer Texte. Diese Art von Dokument bietet viele Gestaltungsmöglichkeiten, wie z.B. die Anzeige von Listen und Tabellen. Des Weiteren gibt es so genannte BlockUIContainer Elemente innerhalb des FlowDocuments, mit deren Hilfe Steuerelemente innerhalb des Textes angezeigt werden können. Dies können z.B. Bilder, Videos oder Buttons sein. Unterstützend wird zusätzlich die Klasse AnnotationService

angeboten, die es erlaubt, Textpassagen hervorzuheben oder Notizen innerhalb des Textes anzubringen.

- Beschriftung - In beiden Frameworks gibt es zur Beschriftung ein Steuerelement welches sich Label nennt. Dieses kann in Flex nur einfachen Text darstellen. In der WPF unterstützt auch diese Komponente das Konzept des flexiblen Inhalts und ist entsprechend beliebig zu füllen. Zusätzlich ist es auf einfache Weise möglich, statt einem normalen Text einen *„AccessText“* darzustellen. In diesem Text kann ein Buchstabe mit einem „_“ markiert werden. Drückt der User nun zur Laufzeit die Alt-Taste so erscheint der markierte Buchstabe unterstrichen. Wird zusätzliche der markierte Buchstabe gedrückt so erhält ein zugehöriges Element automatisch den Fokus. Dies ist eine sehr komfortable Funktion, wenn man Steuerung über Tastatureingabe implementieren möchte.

- Video - Während in Flex zum Abspielen von Videos das VideoDisplay zur Verfügung steht besitzt die WPF mehrere Optionen zur Wiedergabe. Das Steuerelement zum Abspielen von Videos heißt MediaElement. Dieses Steuerelement besitzt eine einfache Ausgabe in einem rechteckigen Bereich und lässt sich in zwei Modi ausführen. Einen Modus zur normalen Ansteuerung (Independent-Modus) und einen Modus zum Synchronisieren mit Animationen (Clock-Modus). Die unterstützten Videoformate sind vielfältiger als in Flex. Möglich sind z.B.: .wmv, .avi, .mpeg. Allerdings wird das .flv Format nicht unterstützt. Durch das flexible Inhaltsmodell ist es möglich, dieses Videosteuerelement anderen Steuerelemente hinzuzufügen und das Video dort abzuspielen. Es wäre dadurch z.B. möglich, ein laufendes Video auf einem Button anzuzeigen. Alternativ dazu ist es möglich innerhalb der WPF die Klasse MediaPlayer zu nutzen. Diese bietet keine eigene Ausgabe. Stattdessen ist es möglich die Videoausgabe manuell auf eine visuelle Oberfläche zu zeichnen. Das kann z.B. das *„Foreground“* Attribut einer TextBox sein womit das Video auf dem angezeigten Text dargestellt werden würde. Eine weitere Möglichkeit ist die Nutzung der Option directShow, was vielfältige Möglichkeiten zur Videowiedergabe und Ansteuerung bietet [Mic07]. Diese Option ist allerdings sehr komplex und soll hier nur der Vollständigkeit halber erwähnt werden, wird aber nicht weiter untersucht.

Zusätzliche WPF Steuerelemente

InkCanvas	Dieses Steuerelement ist zur Eingabe mit einer Maus oder einem Stift gedacht. Der Nutzer kann direkt in das InkCanvas hinein zeichnen. Es gibt die Option das Gezeichnete zu speichern und zu bearbeiten. Als Besonderheit ist es möglich, Gesten zu erkennen oder mit der Zusatzklasse InkAnalyzer zu versuchen, Handschrift zu analysieren und in digitalen Text zu übersetzen.
Toolbar und ToolBarTray	Das Toolbar-Element entspricht der unter Windows typischen Werkzeugleiste. Es ist möglich, mehrere, nach Funktionen zusammengefasste, Werkzeugleisten zu erstellen und diese dann in ein ToolBarTray Element einzufügen. Der Benutzer hat dann die Möglichkeit, selbst über die Anordnung und Größe der einzelnen Werkzeugleisten innerhalb des ToolBarTray zu bestimmen. Diese Funktionalität ist z.B. aus Microsoft Word bekannt.
Frame	Das Frame-Steuerelement bietet die Möglichkeit, HTML Seiten anzuzeigen. Bei der Darstellung wird auf die Funktionalitäten des Internet Explorers zugegriffen.
Document-ViewerBase	DocumentViewerBase ist kein Steuerelement sondern eine Basisklasse von der einige Komponenten erben, wie z.B. FlowDocumentReader oder DocumentViewer. Diese Steuerelemente dienen zur Anzeige von Texten. Einerseits von Texten der FlowDocument Klasse[4] andererseits von Dokumenten des .xps Formats (XML Paper Specification). Das XPS Format ist Teil der WPF, diese Dokumente haben ein festes Layout und ähneln dem .pdf Format (Portable Document Format).
AutoCompleteBox (WPF Toolkit)	Stellt eine Texteingabefeld zur Verfügung. Bei einer Eingabe wird innerhalb einer angegebenen Datensammlung nach Übereinstimmungen gesucht, diese werden angezeigt und können selektiert werden. Die Funktionalität ist bekannt von verschiedenen Suchfeldern auf Webseiten. Sie ist unter anderem zu finden bei der Google Suchmaschine [Wil08].

Tabelle 2.16: WPF zusätzliche Steuerelemente

Die WPF bietet einige Steuerelemente zu denen es im Flex-Framework keine Entsprechung gibt. Stellenweise ermöglichen diese nur wenige Funktionalitäten. Zum Beispiel bietet die

WPF mehrere Button-Typen an, die sich nur minimal unterscheiden. An dieser Stelle soll eine Übersicht, mit Hilfe der Tabelle 2.16, über die zusätzlichen Steuerelemente gegeben werden, die im Verlauf dieser Arbeit, als wesentlich angesehen wurden.

2.2.3 Unterstützende Konzepte und Funktionalitäten

Die WPF hat für viele der vorgestellten Flex-Konzepte ebenfalls Lösungen implementiert. So ist das Event-System ähnlich und es gibt Lösungen für Databinding, Drag and Drop und Validatoren. Ein Konzept welches Flex nicht zur Verfügung stellt, welches aber zur Entwicklung einer Benutzerschnittstelle sehr hilfreich sein kann sind die Commands der WPF.

Commands erlauben eine zusätzliche Trennung zwischen der Präsentationsschicht (die durch XAML erstellte Oberfläche) und der Anwendungslogik. Commands kapseln dabei Funktionalität, die von verschiedenen Stellen der Benutzeroberfläche angestoßen werden kann. Ein Beispiel dafür ist, dass man innerhalb eines Texteditors, die „kopieren"-Funktion durch einen Menüeintrag und durch eine Schaltfläche innerhalb einer Werkzeugleiste zugänglich machen möchte. Den Steuerelementen kann dabei über das Attribut „*Command*" ein entsprechendes Command-Objekt[5] übergeben werden. Es soll an dieser Stelle nicht weiter auf die konkrete Umsetzung eingegangen werden da die Erläuterung zu umfangreich wäre. WPF bietet vorgefertigte Commands und mit zugehörige Funktionalität an. Wenn z.B. eine TextBox das Ziel des vordefinierten Commands, ApplicationCommands.Copy ist, würde direkt die aktuelle Auswahl der TextBox in die Zwischenablage kopiert werden [Ede08, Hub08].

2.3 Vergleich

Beide Frameworks bieten ein Grundkonzept mit mehreren Sprachen an, einer XML-basierten Beschreibungssprache zur Beschreibung der Komponenten und einer (oder mehrere) objektorientierten Programmiersprache(n) zum implementieren der Anwendungslogik. Dies bildet eine starke Basis und wird im Rahmen dieser Untersuchung als wesentlicher Punkt gesehen. Zusätzlich bieten beide ein breites Set an Steuerelementen und unterstützenden Konzepten, wobei auffällt, dass die WPF mehr Steuerelemente zur Verfügung stellt. Der Mehrwert dieser Steuerelemente hängt sehr stark von dem Bedarf

[5] ein Object das das Interface ICommand implementiert hat.

in dem jeweils speziellen Funktions-Bereich ab. Die komplexe Textdarstellung, Gestenerkennung, Schriftanalyse und die Möglichkeit, Html Seiten innerhalb eines Projekts darzustellen, sind im Bedarfsfall mit Flex schwer oder gar nicht manuell zu implementieren[6]. Insgesamt werden sie als leichter Mehrwert relativ zur Gesamtfunktionalität gewertet.

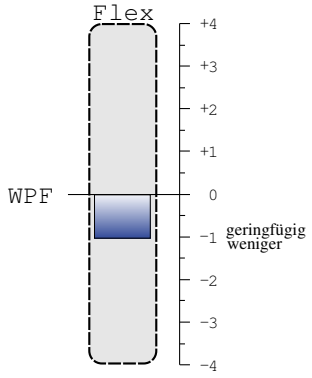

Abbildung 2.3: Vergleich Steuerelemente

Die von der WPF angebotenen Steuerelemente bieten, wie beschrieben, in einigen Fällen komfortable oder wichtige Zusatzfunktionen. Vor allem das Konzept des flexiblen Inhalts bietet in der Darstellung, einige interessante Möglichkeiten z.B. das Anzeigen von Bildern (oder auf Wunsch auch Videos) innerhalb der ComboBox-Komponente.

Bewertung: Die gemeinsame Basis der Frameworks in diesem Bereich überschneidet sich stark und wird als sehr umfangreich beurteilt. Insgesamt bietet die WPF im Bereich der Steuerelemente und Konzepte einen breiteren Umfang an Funktionalität. Deshalb wird Flex hier mit -1 (geringfügig weniger Leistungsumfang, vgl. Abbildung 2.3) relativ zur WPF beurteilt.

[6] auf externe Ressourcen wie einen Browser zuzugreifen, was für die Anzeige der Html-Seite wesentlich ist, ist in Flex nicht möglich, da eine Flex-Anwendung eingeschränkte Zugriffsrechte auf das System hat.

3 Untersuchung und Vergleich der Layout-Optionen

In diesem Kapitel wird die Möglichkeit untersucht Steuerelemente auf dem Bildschirm zu organisieren. Dazu stellen beide Frameworks eine Vielzahl von Containern-Komponenten zur Verfügung. Diese werden in diesem Kapitel untersucht und anschließend verglichen (vgl. Abbildung 3.1). Da Flex im Zentrum dieser Diplomarbeit steht wird einleitend auf Grundlagen des Flex-Layouts eingegangen.

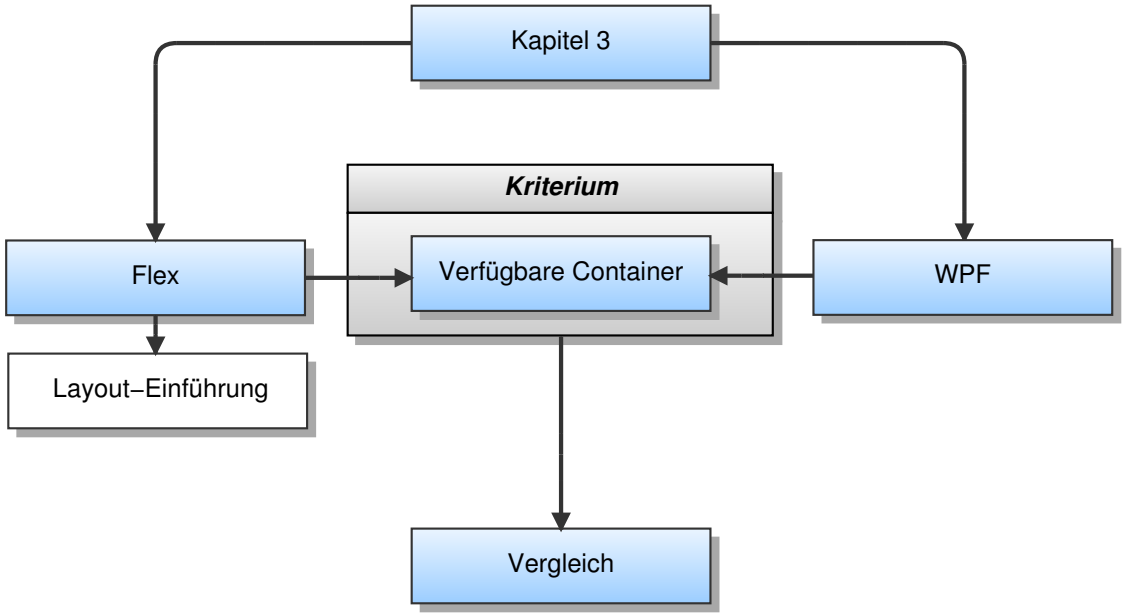

Abbildung 3.1: Ablauf und Kriterien Kapitel 3

3.1 Das Layout in Flex

3.1.1 Das Container-Konzept

Nachdem im vorherigen Kapitel gezeigt wurde welche Steuerelemente Flex zur Verfügung stellt, soll nun betrachtet werden wie man diese Elemente auf dem Bildschirm anordnen kann. Dazu bietet Flex ein Container Konzept an. Ein Container ist ein Behälter, in dem man die eigentlichen Steuerelemente einfügen kann. Sie dienen einer komfortablen Anordnung der Komponenten innerhalb der Flex-Anwendung. Ein Container-Steuerelement ist durch einen rechteckigen Bereich innerhalb einer Flex-Anwendung definiert, der im Flash Player angezeigt werden kann. Ihm werden Steuerelemente oder auch weitere Container hinzugefügt. Diese hinzugefügten Elemente nennt man Kind Elemente. Sie werden nach unterschiedlichen Regeln (je nach Art des Containers) positioniert und dimensioniert. Flex bietet dazu ein breites Spektrum von Containern mit verschiedenen Layouteinstellungen an, welche in diesem Kapitel, mit ihren dazugehörigen Konzepten, vorgestellt werden.

Durch das Verschachteln der verschiedenen Container entsteht eine hierarchische Baumstruktur. Die Wurzel dieses Baumes, das Ursprungselement aller anderen Container, ist der Application-Container. Im Application-Container ist es möglich, über das Attribut, *„layout“* zu beeinflussen, wie die Kind Elemente angeordnet werden. Zur Verfügung stehen die Optionen „`absolut`“, „`vertical`“ und „`horizontal`“. Diese Grundkonzepten der Anordnung lassen sich auch in den meisten anderen Containern wiederfinden.

3.1.2 Grundkonzepte der Anordnung

Es gibt zwei Grundformen der Anordnung. „Absolut“ ermöglicht direktes Anordnen der Komponenten und „Automatisch“ ordnet die Elemente entsprechend vordefinierter Regeln des Containers an, wahlweise in horizontalen oder vertikalen Reihen. Nachfolgend werden diese beiden Konzepte noch etwas genauer erläutert.

Absolutes Layout

Bei Containern, die sich des absoluten Layouts bedienen, werden die Kind Elemente über ihre Attribute fest definiert. Man kann Höhe, Breite und die x,y –Koordinaten direkt festlegen. Die Größenangaben können auch prozentuale Angaben enthalten. Bei

Veränderung der Containergröße bleiben die Elemente auf ihrem Platz. Dadurch kann es passieren, dass Elemente aus dem Blickfeld verschwinden oder sich große Leerräume bilden. Es gibt für diese Form des Layouts noch das unterstützende Konzept der Constrains. Mit Constrains kann man Kind Elemente relativ zu einer Containerseite anordnen. Dieses Konzept wird am Ende des Kapitels, unter „Constrains“ (vgl. Abschnitt 3.1.4), ausführlicher erklärt.

Automatisches Layout

Bei dieser Form des Layouts wird die Position der Kind Elemente durch den Container bestimmt. Es gibt die grundlegenden Formen „vertikal“ und „horizontal“. Bei einem horizontalem Layout werden die Elemente von links nach rechts in einer Reihe angeordnet, bei vertikalem Layout von oben nach unten. Die Position ist dann abhängig von der Größe der eingefügten Elemente und den konfigurierbaren Abständen zwischen den Elementen. Da für eine komplexe Oberfläche meist keine einfache Reihe von Elementen ausreicht, müssen hier die verschiedenen automatischen Container entsprechend kombiniert (verschachtelt) werden.

Hinweis zum Arbeiten mit Containern im Flexbuilder

Wenn man im Designmodus des Flexbuilders mit Containern arbeitet, kann man etwas durcheinander kommen. Oftmals liegen Container ohne äußeres Erscheinungsbild direkt übereinander oder viele Container sind ineinander verschachtelt. Flexbuilder bietet deshalb die hilfreiche Option, die Containerstruktur anzuzeigen. Die Schaltfläche dafür befindet sich direkt neben den Schaltflächen zum Wechseln zwischen Code- und Design-Modus.

3.1.3 Verfügbare Container

Hier wird eine Übersicht über die verfügbaren Container gegeben und deren Funktion erläutert. Eine visuelle Darstellung findet man bei Balderson et al. [BEH+09] oder bei der „Tour de Flex” von Adobe[1]. Zur besseren Übersicht werden die Container an dieser Stelle in zwei Unterklassen unterteilt: Layout-Container und Navigations-Container.

[1] http://www.adobe.com/devnet/flex/tourdeflex/web/

Layout-Container

Die Hauptaufgabe dieser Container ist, wie bereits beschrieben, die Position und Größe ihrer Kind Elemente zu definieren und zu beeinflussen.

- Canvas (Layout absolut) - Ein sehr einfacher Basiscontainer mit absolutem Layout. Er ist, sofern nicht anders konfiguriert, nicht zu sehen.
- Box, VBox, HBox (Layout automatisch) - Die Box ist ein einfacher Basiscontainer mit automatischem Layout. Bei der „normalen“ Box kann man die Layoutausrichtung über das Attribut *„direction“* setzen. Die VBox ist eine Box mit voreingestelltem vertikalem Layout und die HBox entsprechend die horizontale Variante. Die Boxen verfügen, sofern nicht anders konfiguriert, über einen Rahmen.
- DividedBox, VDividedBox, HDividedBox (Layout automatisch) - Diese Container sind ähnlich wie die Box-Container. Mit dem Unterschied, dass zwischen den einzelnen aufgereihten Kind Elementen eine kleine Schaltfläche angezeigt wird. Mit dieser Schaltfläche kann man den zur Verfügung gestellten Platz für die angrenzenden Kind Elemente verändern. Bei der HDividedBox kann man die Breite anpassen, bei der VDividedBox entsprechend die Höhe.
- Panel, TitleWindow (Layout einstellbar) - Das Panel ist ein Container mit einem deutlichem, breitem Rahmen, in dem ein Titel angezeigt werden kann. Unter dem Titel befindet sich eine weiße Region in der die Kind Elemente eingefügt werden können. Das TitleWindow ist mit dem Panel identisch. Man kann aber zusätzlich einen „close“ Button einschalten der dann als Kreuz am rechten oberen Rand erscheint und bietet so sich als Basis für Pop Up Fenster an.
- Grid (Layout speziell) - Das Grid ähnelt vom Konzept her dem von HTML Tabellen. Man kann Spalten anlegen (GridRow) und diese dann mit speziellen Containern füllen (GridItems). Die untereinander liegenden GridItems werden dabei wie eine HTML Spalte behandelt. Das heißt, dass sich die Breite aller GridItems in einer Spalte nach dem breitesten Element in dieser Spalte richtet. Für Reihen gilt dasselbe, allerdings bezogen auf die Höhe der Elemente. Der eigentliche Inhalt wird innerhalb der GridItems angeordnet. Das Grid selbst ist nicht sichtbar.
- Tile (Layouttyp horizontal aber mehrzeilig) - Der Tile Container erzeugt ebenfalls eine Art Tabellenstruktur. Diese ist aber automatisiert. Das heißt, die Kind Elemente

können nacheinander eingefügt und dann automatisch angeordnet werden. Dabei hat ein Cluster für ein Kind Element, das Format des größten Kind Elements im Tile Container. Der Tile Container fühlt die Elemente von oben rechts nach unten links und füllt dabei immer zuerst die Reihen. Die Aufteilung ist also auch abhängig von der Größe des Tile Containers selber. Wenn dieser z.B. sehr breit ist, wird er für die Darstellung des Inhalts wahrscheinlich weniger Reihen brauchen, da auf einer Reihe, dann mehr Elemente untergebracht werden können. Der Tile Container selbst ist standardmäßig nicht sichtbar.

- Form (Layouttyp vertikal) - Dieser Container dient dazu, schnell und einfach Eingabemasken zu erstellen. Dazu bietet das Formheading Element die Möglichkeit, eine Überschrift darzustellen oder einzelne Eingabeabschnitte zu unterteilen. Mit Form Items kann man dann im Format von: „Beschriftung mit einem folgenden Eingabefeld“ die eigentlichen Eingaben erstellen. Das entstehende Formular ist von der Darstellungsart von HTML Seiten bekannt.

- ApplicationControlBar, Controlbar (Layouttyp horizontal) - Der ApplicationControlBar ist ein Balken mit recht auffälliger Optik. Er ist darauf ausgelegt, wie eine Art Navigationsleiste in einer Anwendung zu funktionieren. So gibt es ein eigenes Attribut: “dock“, mit welchem man den Balken am oberen Bildschirmrand „andocken“ kann. Man kann beliebige Kind Elemente einfügen. Es bietet sich jedoch an, Menüs, wichtige Schaltflächen oder Texteingaben dort zu platzieren, welche vielleicht eine zentrale Rolle in der Anwendung spielen. Der Controlbar hat die gleiche Funktionalität nur das er selbst keine optische Darstellung hat.

Navigations Container

Diese Container haben zusätzlich die Aufgabe, auf Benutzersteuerung zu reagieren. Meist werden Teile der Kind Elemente in einer Art Gruppe zusammengefasst. Der Benutzer sieht dann je nach Ansteuerung nur jeweils eine Gruppe der Kind Elemente. Ein Beispiel dafür ist ein Menü bei dem immer nur die Unterpunkte des aktuell angewählten Menüpunktes angezeigt werden.

- Accordion - In den Accordion Container kann man nur weitere Container hinzufügen. Dabei ist immer nur einer dieser Container (mit seinem Inhalt) sichtbar. Jeder dieser Kind Container besitzt ein Attribut „`label`“. Aus diesen Bezeichnungen

der Kind Container, erstellt das Accordion anklickbare Flächen, die untereinander angeordnet sind. Beim Anklicken einer solchen Fläche zeigt der Accordion Container den entsprechenden Kind Container samt Inhalt an.

- Menubar - Der Menubar kreiert ein Menü mit horizontaler Ausrichtung. Man kann direkt im MXML die Menüstruktur erstellen. Diese kann auch Untermenüs beinhalten. Die im MXML angelegten einzelnen Elemente heißen MenuItems. Man kann diesen den „*radio*“ Status geben dann erhalten sie im Menu einen Marker. Dieser stellt dar, ob ein Menupunkt an oder abgewählt ist.

- Viewstack - Der Viewstack ist ein wichtiges Basis Element. Bildlich kann man sich ein Buch vorstellen. Es ist möglich verschiedene Seiten (Container) mit jeweils eigenem Inhalt (Kind Elemente) anzulegen. Zwischen diesen Seiten kann man dann zu Laufzeit umschalten. Es ergibt sich also eine Sammlung verschiedener Steuerelement-Kompositionen, von denen jeweils nur eine angezeigt wird, zwischen denen man aber wechseln kann. Zwischen den einzelnen Containern kann man mit Hilfe des Attributs „*selectedIndex*“ umschalten. Der Viewstack selbst ist nicht sichtbar und dient nur der Anzeige seiner Container-Elemente. Die Viewstack-Komponente wird als Basis für die folgenden Elemente benutzt (Tabbar, Buttonbar, Tooglebuttonbar und Linkbar).

- Tabbar, Buttonbar, Tooglebuttonbar, Linkbar - All diese Elemente sind dazu gemacht, um mit einem Viewstack zu interagieren. Es sind optische Möglichkeiten, eine Viewstack umzuschalten und so den zugehörigen Inhalt anzuzeigen. Der Tabbar erstellt z.B. eine Reiteranzeige, der Togglebuttonbar eine Anzeige mit Knöpfen bei der immer der aktuell gewählte Knopf gedrückt erscheint. Die Verknüpfung ist recht einfach. Man kann einem dieser Elemente über das Attribut „*DataProvider*“ einen Viewstack zuweisen. Der Bar erkennt dann die Anzahl der Elemente des Viewstacks und ließt dessen „*label*“ Attribut aus. Es wird dann automatisch die richtige Anzahl von Bedienflächen mit richtiger Beschriftung erstellt. Der erstellte Bar kann dann ohne weiteren Code direkt genutzt werden und kann den Viewstack umschalten.

3.1.4 Constrains

Constrains sind ein Konzept, um auch in Containern mit absolutem Layout auf Größenänderungen reagieren zu können, sodass z.B. Kind Elemente nicht aus dem sichtbaren Bereich geraten, wenn der Container zu klein wird. Die Position eines Kind Elements wird im absoluten Layout normalerweise über die Attribute „*x*“ und „*y*“ bestimmt. Wenn Constrains genutzt werden, greift man statt dessen auf die Attribute „*left*“, „*right*“, „*top*“, „*bottom*“ zu. Man kann damit das Element an eine Kante des Containers binden und zwar mit einer einstellbaren Entfernung. Bei Skalierung des Containers positioniert sich das Kind Element auf die entsprechende Entfernung zur Container-Kante. Es ist auch möglich ein Kind Element an mehrere Kanten zu binden. Dadurch kann man auch eine dynamische Größe des Kind Elements erreichen. Wenn man z.B. die rechte Seite eines Elements an die rechte Container Kante bindet und die linke Seite entsprechend an die linke Kante, so wird das Kind Element seine Breite immer der Breite des Containers anpassen.

Zusätzlich ist ein fortgeschrittenes Constrains Konzept verfügbar. Damit kann ein Container mit unsichtbaren Constrain Spalten und Reihen durchzogen werden. An den entstehenden Reihen bzw. Spalten können Kind Elemente, ebenfalls über die genannten Attribute, angehängtz werden. Das funktioniert auch mit prozentuellen Angaben. Damit kann z.B. erreicht werden, dass sich ein Kindelement immer genau an der Mitte des Eltern Containers ausrichtet. Für weiterführende Informationen findet man bei Balderson et al. [ALMvV07].

3.2 Das Layout in der WPF

3.2.1 Layout-Container in der WPF

Die WPF bietet ebenfalls ein Containerkonzept an. Dabei liegt der Schwerpunkt auf automatischem Layout, sprich auf Containern, diePosition und Größe ihrer Kind-Elemente selbst verwalten. Für das absolute Layout bietet die WPF ebenfalls einen Canvas Container. Dieser bietet jedoch ein weniger komplexes Constrain-Konzept als die Flex-Variante. Die WPF stellt genau wie Flex mehrere Komponenten mit automatischem Layout zur Verfügung, das Repertoire ist dabei recht ähnlich. Die WPF hat allerdings keine Entsprechung für die DividedBoxen. Es ist jedoch möglich, diese Funktionalität über die Grid

Variante der WPF zu implementieren. Dieses Grid soll hier als wesentlicher Unterschied zu Flex vorgestellt werden, da es einige interessante zusätzliche Möglichkeiten bietet.

Funktionale Unterschiede bei Layout-Containern

Das Grid der WPF ähnelt in der Grundfunktionalität dem Grid in Flex. Es ist also möglich, eine Tabelle mit einer vorgegebenen Anzahl von Zeilen und Spalten anzulegen. Während in Flex die Möglichkeiten, auf die Größe der einzelnen Tabellen-Elemente einzuwirken, begrenzt ist. Bietet die WPF hier mehrere Optionen an. Grundlegend ist es möglich ein Tabellen-Element über mehrere Zeilen oder spalten zu strecken. Dazu stehen die Attribute *„RowSpan“* und *„ColumnSpan“* zur Verfügung.

Zusätzlich besteht die Möglichkeit, in die Tabelle Objekte vom Typ GridSplitter einzufügen. Diese ermöglichen dann zur Laufzeit die Beeinflussung der Breite oder Höhe der Zellen. Des Weiteren können Zellengrößen aneinander gebunden werden, sodass bei Größenänderung einer Zelle über GridSplitter alle gebundenen Zellen ebenfalls ihre Größe entsprechend anpassen. Diese Funktionalität wird über das Attribut *„SharedSizeGroup“* realisiert, dem ein Gruppenname übergeben werden kann. Alle Elemente mit dem gleichen Gruppennamen reagieren dann gemeinsam auf Veränderung.

3.2.2 Navigations-Container in der WPF

Die WPF bietet fast identische Navigationscontainer an wie Flex. Es gibt eine Entsprechung zu dem Akkordion, dem Menü, und dem Viewstack mit seinen verschiedenen Steuerkomponenten. Die Akkordion-Komponente wurde allerdings erst über das WPF Toolkit nachgereicht. Zusätzlich bietet die WPF noch ein Kontextmenü und eine Expander-Komponente an.

Funktionale Unterschiede der Navigations-Containern

Das Menü der WPF bietet genau wie das im Abschnitt 2.2.2 erwähnte Label komfortable Hotkey-Unterstützung an. Als Gegenstück zum Viewstack in Flex bietet die WPF das TabControl an. Dieses bietet die Möglichkeit, zwischen verschiedenen TabItems umzuschalten. Die Anzeige der Tabs kann man dabei, anders als in Flex, wahlweise oben, unten, links oder rechts anordnen. Allerdings bietet Flex unterschiedliche Darstellungsformen der Tabs (Tabbar, Buttonbar, Linkbar, Togglebuttonbar). Zusätzlich können diese in

Flex beliebig positioniert werden, während sie in der WPF Variante immer direkt neben dem eigentlichen Inhalt positioniert werden.

Zusätzliche WPF Navigations-Container

Als kleines zusätzliches Element der WPF soll hier der Expander erwähnt werden. Dieses Steuerelement bietet eine Beschriftung und beliebigen Inhalt an. Der Inhalt kann dabei über eine kleine Schaltfläche ein- oder ausgeblendet werden. Des weiteren bietet die WPF ein Kontextmenü an. Das Kontextmenü ermöglicht die Anzeige eines Menüs innerhalb der Anwendung. Üblicherweise wird die Anzeige durch einen Rechtsklick auf ein Element ausgelöst und wird dann relativ zur Position des Mauszeigers angezeigt. Dabei kann das Menü beliebigen Inhalt anzeigen. In Flex ist die rechte Maustaste bereits durch das Flash-Kontextmenü belegt.

3.3 Vergleich

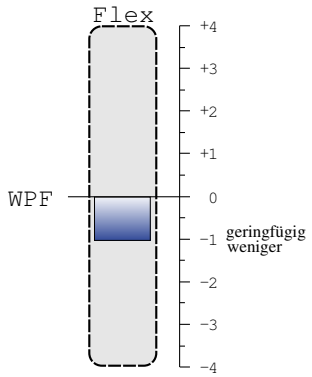

Abbildung 3.2: Vergleich Layout

Flex und die WPF bieten bei den Layout-Komponenten jeweils kleine funktionale Unterschiede. Insgesamt wird die Funktionalität und die sich daraus ergebenden Möglichkeiten in diesem Bereich als recht gleichwertig eingestuft. Die Grid-Komponente der WPF bietet die Möglichkeit, sehr spezielle Anordnungen von Kind Elementen zu realisieren und auch zur Laufzeit auf deren Darstellung Einfluss zu nehmen. Sie sticht damit durch ihre Komplexität hervor, was als deutlicher Mehrwert empfunden wird. Zusätzlich ist das Kontextmenü ein bekanntes und verbreitetes Konzept dessen Fehlen in Flex-Anwendungen störend wirkt.

Bewertung:Insgesamt ist der angebotene Funktionsumfang in diesem Bereich recht ähnlich. Die WPF bietet durch das Grid und das Kontextmenü allerdings Layoutoptionen zu denen Flex keine Entsprechung bietet. Deshalb wird Flex in diesem Bereich mit -1 (geringfügig weniger Leistungsumfang) gegenüber WPF beurteilt.

4 Untersuchung und Vergleich der Möglichkeiten zur Animation von Objekten

Dieses Kapitel untersucht und bewertet die Animationsmöglichkeiten der beiden Frameworks (Flex und WPF). Zusätzlich wird auf das komplexe State-Konzept (bei dem Animationen eine große Rolle spielen) eingegangen. Dieses wird aber in dem abschließenden Vergleich nicht mit einbezogen da für die WPF-Implementierung zu wenig Quellen zur Verfügung standen (vgl. Abbildung 4.1)

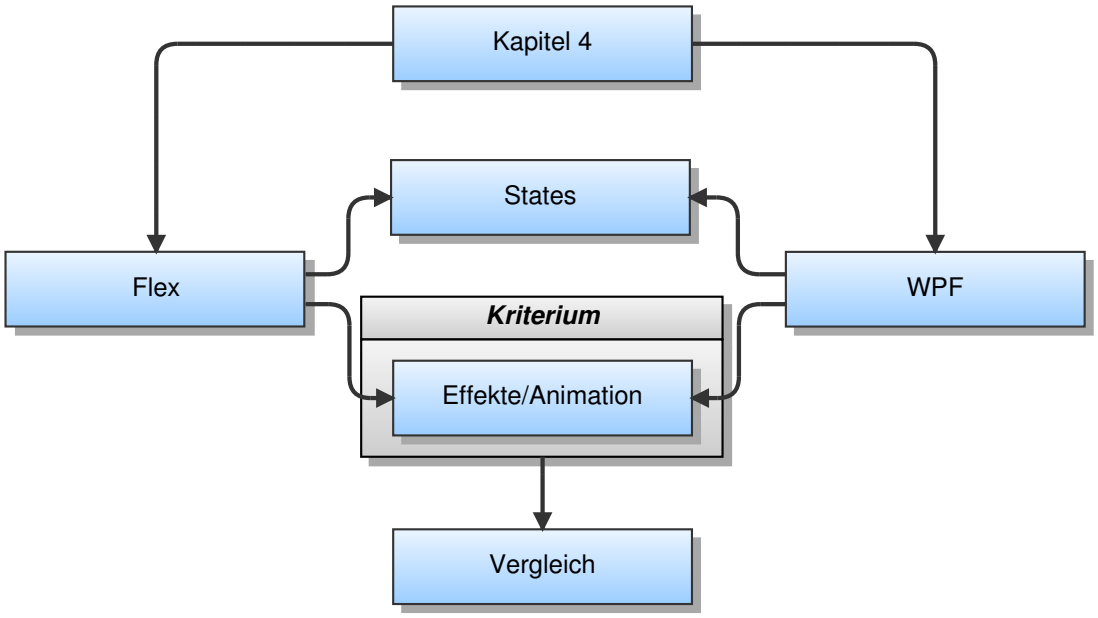

Abbildung 4.1: Ablauf und Kriterien Kapitel 4

4.1 Animationen in Flex

Animationen realisiert man in Flex mit Hilfe von Effekten. Im Grunde steht ein Effekt bereits für eine konkrete Animation. Flex bietet eine Reihe vorgefertigter Effekte wie z.B. Fade (aus/einblenden) oder Rotate (Drehung). Um sie der eigenen Problemlösung anzupassen, kann man sie über Parameter wie z.B. die Effektdauer genau konfigurieren. Es ist auch möglich, sie miteinander zu kombinieren oder bei Bedarf eigene Effekte erstellen. In diesem Kapitel werden die verschiedenen Möglichkeiten des Arbeitens mit Effekten beleuchtet. Anschließend wird das States-Konzept vorgestellt. Dieses Konzept dient nicht direkt dazu Animationen zu entwickeln, aber es beschäftigt sich mit der viusellen Veränderung der Bedienoberfläche. Diese Veränderungen können animiert werden, wobei wiederum auf die Effekte zugegriffen wird. Ein Effekt kann einfach über MXML (damit natürlich auch über Actionsscript) angelegt werden (vgl. Listing 4.1). Das Listing 4.1

Listing 4.1 Fade Effekt, MXML

```
<mx:Fade id="myFade" target="{myUIC}" duration="50"
        alphaFrom="0" alphaTo="0.5" />
```

erstellt einen Fade Effekt. Dieser kann intern über seine Id „`myFade`“ angesprochen werden. Wird der Effekt gestartet, wird dem Ziel des Effekts ein Alpha-Wert von 0 zugewiesen, der sich dann innerhalb der Dauer von 50 ms, zu einem Alpha-Wert von 50 % entwickelt. Das Ziel eines Effekts kann über das Attribut „*`target`*“ zugewiesen werden. Gültig ist jedes Objekt des Typs: UIContainer.

4.1.1 Standarteffekte des Flex-Frameworks

Die folgenden Tabelle (vgl. Tabelle 4.2) listet die Effekte des Flex-Frameworks auf und beschreibt kurz deren Funktion.

Blur	Weichzeichner, der das Ziel verschwimmen lässt
Dissolve	Erzeugt ein farbiges Viereck über der Komponente das sich ausblendet
Fade	Ein- und Ausblenden über die Animation des „*alpha*“-Werts
Glow	Erzeugt einen Schein um die Komponente
Iris	Skalierende Maske die den Blick auf das Ziel freigibt oder es verbirgt
Move	Erzeugt eine Bewegung
Resize	Animiert die Größe einer Komponente
Rotate	Das Ziel wird rotiert
SoundEffect	Spielt eine kurze Audiosequenz ein
WipeLeft, WipeRight, WipeUp, WipeDown	Erstellt eine Maske die, die Komponente aus einer Richtung ein- oder ausblendet.
Zoom	Animiert die X- und Y-Skalierung
AnimateProperty	dieser Effekt hat einen besonderen Status mit ihm kann man eine beliebigen numerische Eigenschaft einer Komponente animieren.

Tabelle 4.2: Flex 3.0 Standard Effekte

(in Anlehnung an Petra Waldminghaus [Wal09])

4.1.2 Effekte konfigurieren

Effekte können, wie in dem obigen Beispiel gezeigt, über Attribute konfiguriert werden. Jeder Effekt hat dabei Standartattribute wie z.B. „*duration*“, „*startDelay*“ oder „*repeatCount*“. Über diese Attribute kann beeinflusst werden, wie lange der Effekt aktiv sein soll, wie lange der Start des Effekts verzögert werden soll oder ob der Effekt eventuell mehrfach ausgeführt werden soll. Die einzelnen konkreten Effekte bringen aber auch jeweils ihre eigenen speziellen Attribute mit. So z.B. der Fade Effekt den „*alphaFrom*“ und den „*alphaTo*“ Wert (Start Alpha Wert und Ziel Alpha Wert). Oder der Move Effekt den „*xFrom*“ und „*xTo*“ Wert (Start X Koordinate und Ziel X Koordinate der

Bewegung). Über die Gesamtheit dieser Attribute ist es möglich die Effekte den eigenen Bedingungen anzupassen. Das Attribut, „*easingFunction*“, soll im Folgenden etwas genauer betrachtet werden, da es eine besondere Funktionalität ermöglicht, um die Bewegungen von Objekten zu beeinflussen.

Easing

Die meisten Standardeffekte bieten das Attribut „*easingFunction*“. Diesem Attribut kann eine spezielle Easing-Funktion übergeben werden. Das mx.effects.easing package bietet bereits einige dieser Funktionen an. Easing bewirkt, das die Interpolation zwischen einem Start und dem Zielwert nicht linear geschieht. Es wird eine mathematische Funktion genutzt um Veränderung in der durch Interpolation hervorgerufenen Bewegung zu erreichen. Ein Beispiel: Eine Komponente soll eine Bewegung, von links nach rechts, über den Bildschirm vollziehen. Dabei soll sie sich nicht mit gleich bleibender Geschwindigkeit bewegen, sondern sich am Anfang schnell bewegen und dann langsamer werden je näher sie ihrem Zielort kommt. Eine Easing-Funktion bietet die Möglichkeit, einer Animation noch einen gewissen Akzent zu verleihen oder sie weicher starten oder ausklingen zu lassen. Auf der Webseite von James Ward[1] werden die Bewegungen, die die Easing-Funktionen des Flex-Frameworks bieten visuell dargestellt.

4.1.3 Effekte auslösen

Nachdem nun geklärt wurde, was ein Effekt ist und wie er angelegt wird soll nun geklärt werden, wie ein Effekt gestartet wird. Dazu gibt es drei unterschiedliche Möglichkeiten: Auslösen über Trigger, manuelles Steuern und starten bei Zustandsübergängen mit Hilfe von Transitions. In diesem Abschnitt werden diese drei Möglichkeiten genauer erläutert.

Auslösen über Trigger

Die Objekte vom Typ UIComponent, denen man Effekte zuweisen kann, bieten alle eine Reihe von Standart-Triggern an. Ein Trigger steht dem Anwender als Attribut einer Komponente zur Verfügung z.B. „*mouseDownEffect*“, ihm kann ein Effekt übergeben werden. Intern ist der Trigger mit einem Event verknüpft. Er startet selbstständig den ihm zugewiesenen Effekt und zwar dann wenn das ihm zugehörige Event ausgelöst

[1] http://www.jamesward.com/easingFunctionFun/easingFunctionFun.html

wird. Die Tabelle 4.4 zeigt eine Auflistung der Trigger, die Flex anbietet und eine kurze Beschreibung, wann diese Trigger ausgelöst werden.

addedEffect	Die Komponente wurde einem Container hinzugefügt
creationCompleteEffect	Die Komponente wurde erzeugt
focusInEffect	Die Komponente hat den Fokus erhalten
focusOutEffect	Die Komponente hat den Fokus verloren
hideEffect	Die Komponente wird unsichtbar
mouseDownEffect	Die Maustaste wurde über der Komponente gedrückt
mouseUpEffect	Die Maustaste wurde über der Komponente losgelassen
moveEffect	Die Komponente verändert ihre Position
removedEffect	Die Komponente wurde aus einem Container entfernt
resizeEffect	Die Größe der Komponente wurde verändert
rollOutEffect	Der Mauszeiger verlässt den Bereich der Komponente
rollOverEffect	Der Mauszeiger tritt in den Bereich der Komponente ein
showEffect	Die Komponente wird sichtbar

Tabelle 4.4: Flex 3.0 Effekt Trigger

(in Anlehnung an Petra Waldminghaus [Wal09])

In Listing 4.2 sieht man, wie ein Effekt angelegt und anschließend dem Trigger eines Steuerelements zugewiesen wird. Sobald der Nutzer nun mit der Maus über das Panel

Listing 4.2 Zuweisen eines Glühen Effekts an einen Trigger

```
//es wird ein Glühen Effekt erzeugt:
<mx:Glow id="myGlow" />

//dieser wird einem Panel auf den rollOverEffect Trigger zugewiesen:
<mx:Panel rollOverEffect="{myGlow}" />
```

fährt, wird der rollOverEffect-Trigger ausgelöst und der Glow-Effekt gestartet.

Effekte können Triggern auch über Cascading Style Sheets zugewiesen werden. So ist es z.B. möglich, allen Buttons innerhalb einer Applikation einen bestimmten Effekt zuzuweisen, der ausgelöst werden soll, wenn der Button gedrückt wird. Mit dieser Arbeitsweise ist es möglich, viel Code zu sparen und dadurch eine größere Übersicht zu erhalten. Trigger sind einfach benutzbar, übersichtlich und erfordern weniger Code

als z.B. eine Implementierung mit der Hilfe von Event-Listenern benötigen würde. Sie sind damit bestens für viele Standardsituationen geeignet. Ergibt sich ein individuelleres Szenario bei dem mehr Kontrolle benötigt wird, müssen die Effekte auf eine andere Weise angesteuert werden.

Manuelles Steuern der Effekte

Alternativ zu Triggern ist es möglich, Effekte in ActionScript mit Hilfe ihrer `play()` Methode zu starten. Dazu muss vorher festlegt werden, welches Ziel der Effekt hat. Bei der Trigger Variante, wurde das dadurch bestimmt, dass der Effekt direkt an den Trigger einer Komponente übergeben wurde. Bei der Ansteuerung über ActionScript, muss beim Anlegen des Effekts über sein Attribut *„target“* ein Ziel festlegt werden (vgl. Listing 4.3). Alternativ ist es auch möglich, einem Effekt über das Attribut *„targets“* mehrere Ziele zuzuweisen. Sobald nun die `play()` Methode des Effekts aufrufen wird, wird der Effekt auf allen Ziel-Komponenten ausgeführt. Zusätzlich zur der Methode, `play()`, bietet

Listing 4.3 Zuweisen eines Glühen über die *„target“*-Eigenschaft

```
//anlegen eines Buttons mit der Id: "myButton".
<mx:Button id="myButton" />
//anlegen eines Glühen Effekts mit dem Ziel: "myButton".
<mx:Glow target="{myButton}"/>
```

Flex noch weitere Optionen zur Ansteuerung. Mit den Effekt-Methoden `stop()`, `pause()`, `resume()`, `reverse()` kann man jetzt individuell in den Ablauf eines Effekts eingreifen. Mit `stop()` kann man einen Effekt stoppen. Mit `pause()` und `resume()` kann man den Effekt anhalten und später an der Stelle fortsetzen. Mit `reverse()` ist es möglich die Abspielrichtung des Effekts zu ändern. Der Effekt wird dann rückwärts abgespielt. Diese Methode wird sofort ausgelöst, wenn der Effekt gerade abgespielt wird. Dann ändert er seine Richtung und läuft zu seinem Ursprungszustand zurück. Ansonsten wird der Effekt so konfiguriert, dass er das nächste Mal, beim Aufruf der `play()`-Methode, invertiert abläuft.

Effekte bei Zustandsübergängen (Transitions)

Die dritte Möglichkeit, Effekte auszulösen ist die Variante über Transitions. Es ist in Flex möglich, innerhalb einer Komponente visuelle Zustände zu definieren, die sogenannten States. Der Übergang zwischen zwei Zuständen kann mit Hilfe von Transisition animiert werden. Diese Technik wird im Abschnitt 4.1.6 noch genauer betrachtet.

4.1.4 Effekte kombinieren – Composite-Effects

Bisher wurde erläutert, wie einzelne Effekte genutzt werden. Zusätzlich besteht aber die Option, Effekte zu kombinieren. Dazu stellt das Flex-Framework die Klassen Sequence und Parallel zur Verfügung. Beide sind vom CompositeEffect abgeleitet, welcher wiederum von Effect abgeleitet ist. Es ist also möglich, sie überall dort zu übergeben, wo ein normaler Effekt erwartet wird. Man kann sie wie normale Effekte betrachten. Die Klasse Sequence ermöglicht es, mehrere Effekte nacheinander abzuspielen. Über MXML können der Sequenz mehrere Effekte hinzugefügt werden. Beim Starten der Sequenz wird der erste Effekt abgespielt. Wenn dieser fertig ist wird der nächste Effekt gestartet usw., bis die Effektliste vollständig durchlaufen ist. Die Klasse Parallel ermöglicht es, ebenfalls mehrere Effekte hinzuzufügen diese werden dann allerdings alle gleichzeitig abgearbeitet. Ein Beispiel: Kombiniert man einen Move und einen Fade Effekt miteinander als

- Sequenz, so wird sich das Objekt erst bewegen und nachdem es seine Zielposition erreicht hat, wird es eingeblendet.
- Parallel, so wird sich das Objekt bewegen und sich während der Bewegung einblenden.

Es ist möglich, Sequenzen und Parallele miteinander zu verschachteln und so recht komplexe Effekte-Kombinationen zu erstellen.

4.1.5 Eigene Effekte erstellen

Mit den Standardeffekten in Kombination mit den Composite-Effekten stellt das Flex-Framework viele Möglichkeiten zur Verfügung, um Animationen in einer Anwendung zu realisieren. Trotzdem werden damit natürlich nicht alle Entwicklungs-Szenarien abgedeckt. Für den Fall, dass die Standardeffekte nicht ausreichen, bietet das Flex-Framework die Möglichkeit, sich eigene Effekte zu erstellen. Diese Effekte können dann genauso benutzt

werden wie die Standarteffekte. Um eigene Effekte zu erstellen ist es allerdings nötig, die Implementierung der Effekte innerhalb des Flex-Frameworks besser zu verstehen und wird deshalb an dieser Stelle nicht weiter beleuchtet.

4.1.6 Das State-Konzept in Flex

Um zu klären was States eigentlich sind, soll einführend an Hand eines abstrakten Beispiels gezeigt werden, welche Funktionalität States ermöglichen. Angenommen, in einem Login Bereich befinden sich zwei Eingabefelder und ein Login Button. Diese Steuerelemente sind untereinander angeordnet. Es soll nun erreicht werden, dass diese Elemente sich zur Laufzeit nebeneinander anordnen (vgl. Abbildung 4.2). In diesem Fall können States eingesetzt werden. Bildlich betrachtet ist nun folgendes möglich. Es wird ein Foto vom Originalzustand der Elemente gemacht, in dem sie untereinander angeordnet sind. Dann werden sie zu der Position verschoben, die sie nebeneinander einnehmen sollen und es wird erneut ein Foto gemacht. Später kann man dann zur Laufzeit sagen: „Elemente! Ordnet euch an, wie auf dem ersten Foto!“ oder eben: „Elemente! Ordnet euch an, wie auf dem zweiten Foto!“ Die Fotos übernehmen in diesem bildlichen Beispiel die Aufgabe von States.

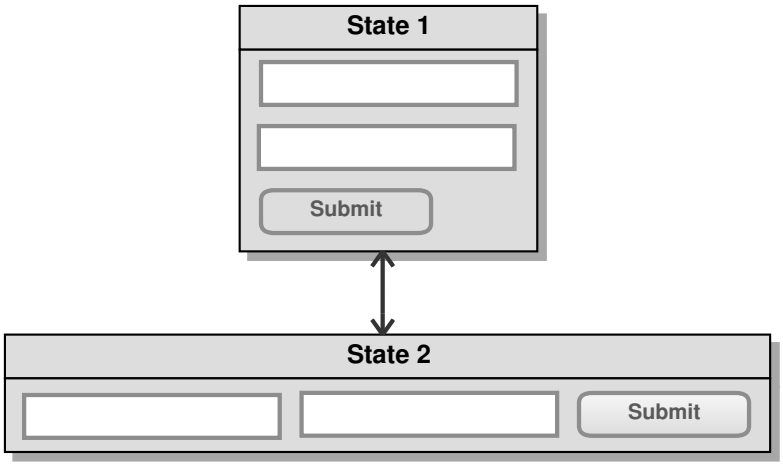

Abbildung 4.2: States-Beispiel

Ein State kann also verschiedene Zustände und Anordnungen von Steuerelementen innerhalb von Komponenten speichern und sie bei Bedarf wieder herstellen. Man kann für jede UIComponent States definieren und zwischen diesen States über die Eigenschaft „currentState“ umschalten.

Dabei hat eine Anwendung immer wenigstens einen State den „Base State“. Dieser enthält die grundlegende Anordnung, jeder Anwendung oder Komponente. Von diesem Base State ausgehend können neue States erstellt werden. Dabei speichert ein State nicht wirklich die ganzen Einstellungen aller einzelnen Steuerelemente, sondern er speichert nur die Werte ab, die einen Unterschied zwischen „Ursprungszustand“ und eigenem Zustand aufweisen. Das bedeutet, dass jeder neu erstellte State einen Ursprungszustand braucht auf den er sich bezieht.

4.1.7 States erzeugen

Der Flexbuilder bietet eine recht komfortable grafische Möglichkeit mit States zu arbeiten. Man kann per Knopfdruck neue States generieren und dann in der grafischen Ansicht der Anwendung die Steuerelemente so anordnen, wie sie in dem entsprechenden Zustand benötigt werden. Dabei wird MXML Code generiert, der natürlich auch manuell erstellt werden kann. Listing 4.4 zeigt, wie man einen State mit dem Namen „`myFirstState`“ anlegt. Das hier gezeigte Beispiel zeigt nur die grundlegende Syntax, und bringt noch

Listing 4.4 Anlegen eines States in Flex 3.0 ohne Veränderungen

```
<mx:states>
        <mx:State name="myFirstState">

        </mx:State>
<mx:states>
```

keine Veränderungen mit sich. Ein State hat außer dem Namen noch zwei wichtige Eigenschaften. Erstens die Eigenschaft *„`basedOn`“*, sie definiert den Ursprungszustand des States. Wird kein hier kein Wert übergeben, so wird automatisch der Base State als Ursprungszustand genutzt. Die zweite Eigenschaft ist *„`overrides`“*. Dies ist ein Array welches die einzelnen Zustandsänderungen vom *„`basedOn`“* State zu dem eigenen State enthält. Diese Änderungen kann man hinzufügen, indem man Override-Elemente zwischen den öffnenden und schließenden <mx:State> Tag einfügt. Im Listing 4.5 ändern sich zwischen dem Zustand „`Base State`“ und dem Zustand „`myFirstState`“ zwei Eigenschaften. Das ist die X- und die Y-Koordinate der Schaltfläche „`loginButton`“. Diese Änderungen wurden in das overrides-Array eingetragen. Es soll nun gezeigt werden, welche Änderungen in das overrides-Array eingetragen werden können, also welche Möglichkeiten der Veränderung zwischen States das Flex Framework anbietet.

Listing 4.5 Anlegen von States in Flex 3.0 mit Veränderungen

```
<mx:states>
        <mx:State name="myFirstState">
                <mx:SetProperty target="{loginButton}"
                        name="x" value="10" />
                <mx:SetProperty target="{loginButton}"
                        name="y" value="20" />
        </mx:State>
<mx:states>
```

4.1.8 Veränderungen zwischen States

Die folgende Tabelle (vgl. Tabelle 4.6) zeigt, die möglichen Elemente des overrides-Arrays. Sie werden im Anschluß an die Tabelle, im Detail erläutert.

<mx:SetProperty>	Ändert eine Eigenschaft einer Komponente
<mx:SetStyle>	Ändert die Stil-Eigenschaft einer Komponente
<mx:AddChild>	Fügt dem Zustand eine neue Komponente hinzu
<mx:RemoveChild>	Entfernt eine Komponente aus dem Ursprungszustand
<mx:SetEventHandler>	Registriert einen neuen Eventhandler

Tabelle 4.6: override-Array Elemente

(in Anlehnung an Petra Waldminghaus [Wal09])

Veränderungen von Eigenschaften und Stilen

Die Eigenschaften eines Steuerelements kann man über den Tag <mx:SetProperty> ändern. Dabei gibt es die drei wesentliche Attribute *„target“*, *„name“* und *„value“*. Mit *„target“* legt man die zu beeinflussende Komponente fest. Mit *„name“* kann man ein beliebiges numerisches Attribut der Komponente angeben, welches verändert werden soll. Die Eigenschaft *„value“* erwartet den neuen Wert, des über *„name“* definierten Attributs. Der <mx:SetStyle> Tag funktioniert grundlegend genauso, bezieht sich aber nicht auf Attribute der Ziel-Komponente, sondern auf eine Style Eigenschaft. Die UI-Components besitzen eine Reihe von optischen Einstellungsmöglichkeiten. Um diese zu konfigurieren benötigt man die Funktion: *setStyle(„styleName", „value")*. Mit

dem <mx:SetStyle> Tag ist es direkt möglich, ein Style Attribut zusetzen.

Steuerelemente hinzufügen oder entfernen

Mit dem Tag <mx:AddChild> lassen sich neue Objekte hinzufügen. Welches Steuerelement hinzugefügt werden soll kann mit dem Attribut *„target“* festlegt werden. Über die Attribute *„targetFactory“* und *„creationPolicy“* ist einstellbar, wann die Instanz des hinzugefügten Objekts erstellt werden soll. Über die Eigenschaft *„position“* kann man angeben auf welcher Darstellungsebene innerhalb des Containers das neue Objekt angezeigt wird. Dabei gibt es die Möglichkeit *„firstChild“* oder *„lastChild“*, was dem obersten oder untersten Element im Container entspricht. Alternativ kann man *„before“* oder *„after“* nutzen. Damit kann man angeben, dass sich das neue Objekt direkt vor oder hinter einem anderen Objekt, anordnen soll. Das Objekt an dem die Ausrichtung in diesem Fall stattfindet, kann über das Attribut *„relativeTo“* festgelegt werden. Mit dem Tag <mx:RemoveChild> lassen sich Objekte entfernen. Dieser Tag verfügt ebenfalls über das Attribut *„target“* mit dem festgelegt wird, welches Objekt entfernt werden soll. Das Objekt ist dem System in diesem Fall nicht mehr bekannt. Es darf also nicht mehr darauf zugegriffen werden, da sonst ein Fehlerauswurf provoziert wird.

Evenhandler hinzufügen oder entfernen

Diese Funktion bietet sich an, wenn Steuerelemente in einem anderen State kurzzeitig anders genutzt werden sollen. Ein Beispiel: In einen Loginbereich gibt es die Möglichkeit, Loginname und Passwort einzugeben. Nun möchte man diese Elemente auch gleich für eine Registrierung nutzen. Dazu wird ein weiteres Eingabefeld über <mx: AddChild> hinzugefügt. Es dient zum Wiederholen des Passworts beim Registrieren. Die Beschriftung des Login-Buttons wird zusätzlich in *„registrieren“* geändert. Wird dieser Button jetzt gedrückt, soll natürlich eine andere Funktionalität ausgelöst werden als beim Loginvorgang. Um dies zu erreichen kann jetzt der Eventhandler geändert werden. Die Möglichkeit soll hier nachfolgend nur kurz, mit zwei Sätzen, beschrieben werden. Der <mx:SetEventHandler> Tag verfügt über vier wichtige Eigenschaften. Die Eigenschaft *„target“* bestimmt das Objekt für welches das Event registriert werden soll, *„name“* erwartet den Namen des Events und die *„handlerFunction“* oder der *„handler“* legen fest, was bei Eintreffen des Events geschehen soll.

4.1.9 Zustandsübergänge animieren

Es ist möglich, die Übergänge zwischen den Zuständen zu animieren. Hierzu wird wieder auf die im Abschnitt 4.1.1 eingeführten Effekte zugegriffen. Die Möglichkeit, die Effekte mit den Zustandsübergängen zu verknüpfen, nennt man: „Transition“. Um Transitions zu nutzen, wird ein transitions-Array innerhalb der Komponente, in der die States definiert sind, gefüllt. Es stellt die Sammlung aller Animationsdefinitionen zwischen den Zuständen dar. Ein einzelnes Objekt dieses Arrays nennt sich Transition. Eine Transition definiert, welche Animationen ausgeführt werden sollen, wenn die Komponente ihren Zustand wechselt. Das Transition Objekt besitzt drei wichtige Attribute. Das *„effect“* Attribut erwartet einen Effekt. Dies kann auch ein Composite-Effekt sein. Außerdem gibt es das *„fromState“* und das *„toState“* Attribut, diese definieren bei welchem State-Übergang der Effekt ausgelöst wird. Listing 4.6 zeigt dieses Vorgehen. Dieses Beispiel setzt voraus,

Listing 4.6 Transition mit Flex 3.0

```
<mx:transitions>
        <mx:Transition effekt="{myEffect}"
                fromState="firstState" toState="secondState"/>
</mx:transitions>

<mx:Fade id="myEffect" target="{loginButton}"
        alphaFrom="0" alphaTo="1"/>
```

dass man zwei States definiert hat. Einen *„firstState“* und einen *„secondState“*. In beiden ist das Steuerelement „`loginButton`“ vorhanden. Dieser Button wird beim Übergang vom firstState zum secondState eingeblendet. Es gibt hier noch einige weiterführende Konzepte wie z.B. die Festlegung der Reihenfolge der einzelnen Stateveränderungen. Diese werden an dieser Stelle jedoch aus Platzgründen, nicht weiter ausgeführt. Eine ausführliche Beschreibung findet man bei Balderson et al. [BEH+09]

4.2 Animationen in der WPF

Um Animationen zu realisieren stellt die WPF zahlreiche Klassen zur Verfügung. Diese befinden sich im Namespace System.Windows.Media.Animation. Eine wichtige Basis für Animationen ist die Interpolation von Werten zwischen einem Start und einem Zielwert. Anders als in Flex, wird von der WPF, nicht nur Zwischenwertberechnung von

nummerischen Werten angeboten, sondern es ist zusätzlich auch Interpolation bei anderen Datenobjekten wie z.B. Color- oder Vector3D-Objekten möglich[2]. Es können also z.B. auch Farbeigenschaften animiert werden. Bei der Implementierung wird ein Container für eine Reihe von Animationen zur Verfügung gestellt, das Storyboard. Für dieses wird ein zu animierendes Objekt festgelegt. Dem Storyboard können dann verschiedene Animationen hinzugefügt werden, die jeweils eine Eigenschaft des Animationsobjekts beeinflussen. Zum Beispiel wäre es möglich, einen Button als Animationsobjekt festzulegen und dann einzelne Animationen zu erstellen, die jeweils die X- und Y-Koordinate verändern. Das Storyboard selbst kann vom Entwickler über Steuerklassen beeinflusst werden. Mögliche Funktionalitäten sind z.B. `PauseStoryboard` (ermöglicht das Anhalten des Storyboards) oder `SeekStoryBoard` (überspringt eine bestimme Zeit in einem Storyboard). Gestartet wird das Storyboard über die Methode `Begin()`. In XAML kann alternativ zum Aufruf dieser Methode das Storyboard in einen <BeginStorybard> Tag gekapselt werden. Das Ganze kann, ähnlich wie in Flex, direkt in XAML Triggern zugeordnet werden. Die Funktionalität, eine Animation zu startet, wenn z.B. ein Button gedrückt wird, kann also direkt in XAML umgesetzt werden. Es gibt drei verschiedene Animationstypen, die WPF dem Entwickler zur Verfügung stellt und die innerhalb der Storyboards genutzt werden können. Diese soll kurz vorgestellt werden.

4.2.1 Animationstypen in der WPF

Die drei Typen von Animationen die, von der WPF angeboten werden sind Basis-Animation, Pfadanimation und Keyframe-Animation. Diese werden in diesem Abschnitt einzeln erläutert. Alle drei Typen haben gemeinsame Eigenschaften über die der zeitliche Verlauf der Animation gesteuert werden kann. Zum Beispiel `Duration` (Animationsdauer), `AccerlerationRatio` (ermöglicht ein Startbeschleunigung der Bewegung), `DecelerationRatio` (ermöglicht ein Abbremsen am Ende der Animation) oder `RepeatBehavior` (ermöglicht Wiederholungen der Animation).

Basis-Animation

Die Basis-Animation kann auf viele der animierbaren Datentypen angewandt werden. Das Grundkonzept der Basis-Animation sieht vor, dass ein Start- und ein Ziel-Wert

[2]Bei Huber[Hub08] – findet man eine vollständige Tabelle der Datentypen

angegeben wird und die Animation dann zwischen diesen beiden Werten abläuft. Zur Verfügung stehen die Attribute „*from*“ (definiert Startwert), „*to*“ (definiert Zielwert) und „*by*“ (definiert eine Spanne). Der Start- und Ziel-Wert kann also z.B. auch durch die Angabe eines Startwerts und einer angegebenen Veränderung, definiert werden.

Pfad-Animation

Die Pfad-Animation sieht vor, dass das Animationsobjekt sich an einem Bewegungspfad ausrichtet. Dazu kann dem Attribut „*PathGeometry*“ ein Pfad übergeben werden (Objekt vom Typ PathGeometry). Animiert werden können mit dieser Form nur die Typen Double, Point und Matrix. Es ist möglich, zusätzlich zu der X-,Y-Koordinate der aktuellen Pfadposition, auch den Winkel der Tangente zu erhalten. Dadurch bietet sich die Option, ein Objekt das sich an dem Pfad entlang bewegt, auch immer relativ zum Pfad auszurichten (zu drehen). So kann z.B. bei einer Animation an der Innenseite eines Kreies entlang, das animierte Objekt immer zur Mitte des Kreises ausgerichtet sein.

Keyframe-Animation

Die Keyframe-Animation arbeitet mit einer Folge von konkreten Werten (Schlüsselwerte), die eine Animation durchlaufen soll. Sie kann mit allen animierbaren Datentypen durchgeführt werden. Bei einer solchen Animation werden alle Schlüsselwerte mit einem Zeitpunkt verknüpft und nacheinander angesteuert. Es gibt jedoch drei unterschiedliche Arten, auf die die interpolierten Werte zwischen diesen Schlüsselwerten berechnet werden. Diese werden kurz erläutert.

- Diskrete Keyframe-Animation - Bei dieser Form werden nur die Schlüsselwerte verwendet, es gibt keine Interpolation. Ein Schlüsselwert bleibt solange als aktueller Animationswert bestehen, bis der Zeitpunkt des nächsten Schlüsselwerts erreicht wird. Sobald dieser erreicht ist entspricht der aktuelle Animationswert diesem Schlüsselwert. Für manche Datentypen kommt nur diese Form der Animation in Frage. Zum Beispiel für den String-Datentyp. Objekte von diesem Typ, enthalten Texte, zwischen deren Schlüsselwerten nicht interpoliert, sondern nur umgeschaltet werden kann.

- Lineare Keyframe-Animation - Bei dieser Form der Animation wird eine lineare Interpolation zwischen den Schlüsselwerten durchgeführt. Wenn man sich die

Schlüsselwerte diese Animation als Punkte um Raum vorstellt, würden die Werte der linearen Interpolation eine Gerade zwischen den Schlüsselwerten ergeben.

- Spline Keyframe-Animation - In diesem Fall ist es möglich, für die Schlüsselwerte zusätzlich zu dem Zeitpunkt und dem eigentlichen Wert zwei Keyspline-Werte anzugeben. Diese fließen in die Interpolationberechnung mit ein. Wenn man sich die Schlüsselwerte wieder als Punkte im Raum vorstellt, würde die Interpolation in diesem Fall eine Spline-Kurve, anhand der angegebenen Keyspline-Werte, zwischen den Punkten ergeben. Wie im Fall des String-Datentyps gezeigt, kann nicht jeder Datentyp mit jeder Interpolationmethode genutzt werden. Eine Übersicht über die Datentypen und die mögliche Interpolation findet man bei Marquardt [Mar07].

4.2.2 Visual State Manager in der WPF

Die WPF bietet ein Konzept, das dem State-Konzept von Flex ähnelt. Es nennt sich Visual State Manager und wird über das WPF Toolkit bereit gestellt. Laut der Webseite WPF-CodePlex[3] wurde dieses Konzept in der Toolkit-Version vom Juni 2009 veröffentlicht. Vorher gab es bereits Preview-Versionen. Durch den recht nah zurückliegenden Release-Termin, konnte in der Literatur keine Beschreibung zu diesem Konzept gefunden werden. Es wird an dieser Stelle ein Ausblick auf die Funktionalität, auf Grund der Informationen der Webseite CodePlex, geboten. In dem Visual State Manager ist es wie in Flex möglich, States zu erzeugen. Die States werden aber zusätzlich in StateGroups gekapselt. Eine StateGroup beinhaltet dabei mehrere sich gegenseitig ausschließende States. Eine Komponente kann dabei mehrere StateGroups beinhalten und somit auch mehrere Zustände annehmen, immer jeweils einen innerhalb einer Gruppierung. Die States repräsentieren dabei, ebenfalls wie in Flex, einen konkreten Zustand der Komponente. Die Zustände unterscheiden sich durch die Anordnung und Darstellung der Elemente innerhalb der Komponente. Über die `GoToState`-Methode kann zwischen diesen Zuständen umgeschaltet werden. Zusätzlich kann für einen State ein Storyboard angegeben werden, das eine Animation enthält, die abgespielt wird, wenn sich die Komponente in dem entsprechenden Zustand befindet. Dies würde z.B. das Blinken eines Steuerelements ermöglichen. Eine weitere Option ist das Erzeugen von Transitions, ähnlich wie in Flex. Mit Hilfe der Transition kann den Übergängen zwischen konkreten Zuständen, Animatio-

[3] CodePlex Ist das Portal zur Beschreibung des WPF Toolkits von Microsoft

nen zugewiesen werden. Man definiert also die visuelle Veränderung zwischen zwei oder mehreren Zuständen.

4.3 Vergleich

Beide Frameworks bieten zur Animation ein Konzept an, in dem Objekteigenschaften über die Zeit verändert werden können. Bei Flex beschränkt sich dies auf numerische Werte, während die WPF auch in der Lage ist, zwischen komplexeren Datentypen zu interpolieren. Flex bietet dem Entwickler einige Standardanimationen an, die nicht ohne weiteres durch die Manipulation einer einfachen Objekteigenschaft erreicht werden können, z.B. der Iris- oder der Blur-Effekt. Die WPF liefert nichts Vergleichbares. Zusätzlich bietet das Flex-Framework über die CompositeEffects die komfortable Möglichkeit, Animation zu kombinieren. In WPF ist dies innerhalb das Storyboards ebenfalls möglich.

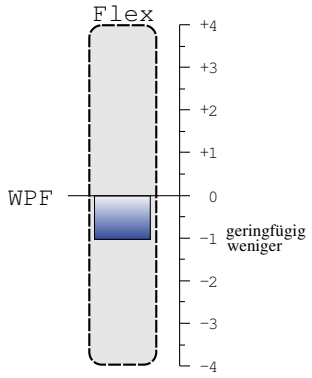

Abbildung 4.3: Vergleich Animation

Allerdings müssen Animationsfolgen durch die Anpassung der Startzeiten manuell erstellt werden. Auch bei den „Animationsakzenten“ bietet Flex mit den easing-Funktionen eine Möglichkeit, schnell aus einer umfangreichen vorgefertigten Optionspalette zu wählen. Die WPF ermöglicht eine Manipulation des Zeitverlaufs entweder über die Attribute AccerlerationRatio bzw. DecelerationRatio womit eine Animation auf einfache Weise beschleunigt bzw. abgebremst werden kann. Für eine komplexeren Einfluss müsste auf eine Keyframe-Animation mit Spline-Kurven zurückgriffen werden. Bei der Berechnung des eigentlichen Animationsverlaufs bietet die WPF jedoch ein viel breiteres Spektrum an Möglichkeiten. Flex liefert hier einen Funktionsumfang der mit dem der von WPF angebotenen Basis-Animation zu vergleichen ist. Die zusätzlichen Optionen der Pfad-Animation bzw. der Keyframe-Animation, mit ihren verschiedenen Arten der Interpolation, bieten die Möglichkeit, deutlich komplexere Bewegungsabläufe zu implementieren.

Die beiden Konzepte der Frameworks, zur Implementierung von States wirken ähnlich. Eine Bewertung wird innerhalb dieser Untersuchung aber nicht vorgenommen da die Quellen zum Visual State Manager der WPF unzureichend sind.

Bewertung: Flex ermöglicht mit den vorgefertigten Effekten und den Easing-Funktionen eine schnelle Implementierung von einem breiten Spektrum von Animationen. Die WPF wiederum bietet wesentlich komplexere Möglichkeiten, Animationen zu entwickeln. Die von Flex gebotene Funktionalität lässt sich damit ebenfalls umsetzen. Da das Vorhandensein sehr komplexer Animationen im Rahmen der Entwicklung einer Grafischen Benutzerschnittstelle als eher selten eingeschätzt wird, wird Flex insgesamt mit einer -1 (geringfügig weniger Leistungsumfang, vgl. 4.3) gegenüber WPF im Bereich der Animationen bewertet.

5 Untersuchung und Vergleich der visuellen Anpassbarkeit der Steuerelemente

In diesem Kapitel wird untersucht welche Möglichkeiten zur Darstellung der Steuerelemente von den Frameworks zur Verfügung gestellt werden. Beide Frameworks bieten eine Technik über Stil-Eigenschaften visuelle Anpassungen vorzunehmen. Um grundlegendere Veränderungen vorzunehmen gibt es unterschiedliche Herangehensweisen (Skinning und Templates). Zusätzlich ist es in beiden Frameworks mögliche die visuellen Anpassungen zusammenzufassen und damit besser zu organisieren. Die Gesamtheit der Optionen zur visuellen Veränderbarkeit werden in einem abschließenden Vergleich aneinander gemessen (vgl. Abbildung 5.1).

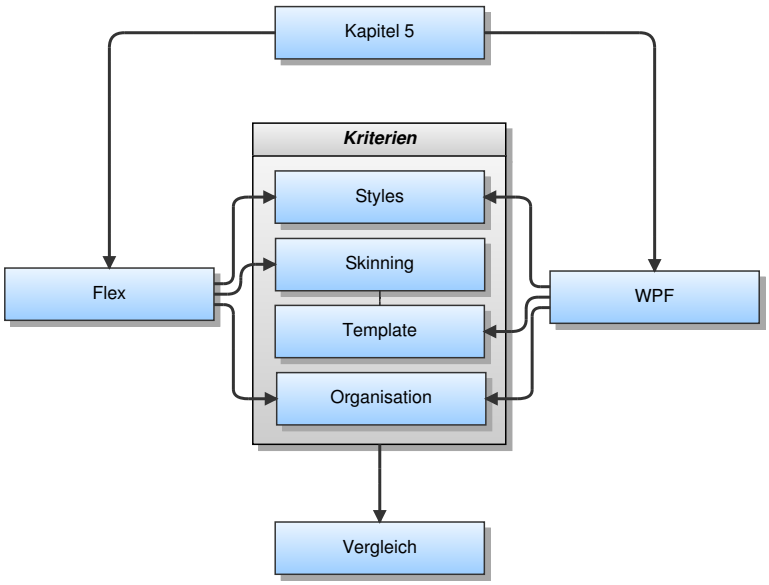

Abbildung 5.1: Ablauf und Kriterien Kapitel 5

5.1 Stile und Skins in Flex

In Flex gibt es zwei grundlegende Möglichkeiten, die Darstellung der Komponenten anzupassen: Stile und Skins. Unter Stilen versteht man Eigenschaften der Steuerelemente in Flex, die eine optische Veränderung ermöglichen. Das können zum Beispiel Rundungen, Schriftgröße, Abstände oder Farbeinstellungen sein. Mit der Hilfe von Skins ist es möglich, die Darstellung durch das zuweisen von Grafiken selbst zu definieren. So kann man zum Beispiel die Anzeige eines normalerweise eckigen Buttons durch eine runde Darstellung ersetzen. In diesem Kapitel sollen zuerst die Möglichkeiten der Stile beleuchtet werden und danach die des Component Skinnings.

5.1.1 Stile

Jedes Flex Steuerelement bietet verschiedene Stil Attribute an. Diese sind sehr vielfältig und es ist möglich eine große Anzahl von Einstellungen vorzunehmen. Um den Umfang an einem Beispiel zu verdeutlichen, wird hier kurz auf die Button-Komponente eingegangen. Die Flex Api[1] gibt an, dass diese Komponente über 51 Stil Eigenschaften verfügt. So z.B.: Farbeinstellungen für verschiedene Zustände, komplexe Alpha und Highlight Einstellungsmöglichkeiten, verschiedene Font Konfigurationen und Abstandsdefinitionen. Diese Einstellungen kann man direkt beim Erstellen der Komponente innerhalb des MXML-Tag vornehmen (inline). Ein Beispiel anhand einer Button-Schaltfläche zeigt Listing 5.1. Im Beispiel des Listings 5.1 wird den Rundungen der Ecken ein Wert von

Listing 5.1 Flex 3.0 „inline" Stile

```
<mx:Button cornerRadius="15" fontStyle="italic" label="Login"/>
```

„15" zugewiesen, was einer großen Abrundung entspricht. Außerdem ist die Beschriftung des Buttons „Login" und diese Schrift wird kursiv dargestellt.

Der Nachteil dieser Einstellungsoption ist, dass man die Konfiguration für jede Instanz einer Komponente einzeln vornehmen muss. Selbst wenn eigentlich eine Vielzahl von Elementen visuell identisch dargestellt werden soll. Eine Alternative bietet da die Nutzung einer CSS-Datei (Cascading Style Sheets).

[1] http://livedocs.adobe.com/flex/3/langref/index.html

Stile in CSS

Wenn man Stile mehrfach in einer Anwendung nutzen will, bietet es sich an eine CSS-Datei anzulegen. Dies ist eine einfache Textdatei mit der Endung „.css“. Wurde eine CSS-Datei erstellt, kann sie mit dem Tag <mx:Style> in die Applikation eingebunden werden. Alternativ besteht auch die Möglichkeit, die CSS Datei als Compiler Argument zu übergeben. Innerhalb dieser Datei können eigene Stilezuweisungen definiert werden. Dazu wird eine Stileigenschaft und ein dazugehöriger Wert angegeben. Doch während bei der inline Definition klar ist, auf welche Komponente sich die Stil Einstellung bezieht, muss dies in der CSS-Datei explizit festgelegt werden. Dafür gibt es Selektoren. Ein Selektor bestimmt, für welche Komponente eine CSS-Stil-Einstellung gültig ist. Es gibt drei unterschiedliche Arten von Selektoren: Typ-Selektoren, Klassen-Selektoren sowie den global-Selektor diese werden im Folgenden genauer untersucht.

- Typ-Selektor - Der Typ-Selektor bezieht sich auf den Namen eines Komponententyps, zum Beispiel „Button“. Alle Stil Einstellungen innerhalb dieses Selektors würden sich nun auf alle Buttons im Gültigkeitsbereich[2] der CSS-Datei beziehen. Das Listing 5.2 zeigt ein Beispiel für einen Typ-Selektor. Mit diesen Einstellungen würden alle Button-Steuerelemente der Anwendung, stark abgerundete Ecken erhalten und eine weiße Randfarbe.

Listing 5.2 Flex 3.0 Typ-Selektor

```
Button{
        cornerRadius: 15;
        borderColor: #FFFFFF;
}
```

- Klassen-Selektor - Der Klassen-Selektor bietet sich an, wenn man innerhalb der Anwendung mehrere Untertypen eines Steuerelements benötigt. Zum Beispiel für einen rot- und einen grün-umrandeten Button. Ein Klassen Selektor ist dadurch gekennzeichnet das sein Name mit einem „.“ beginnt. Er bezieht sich nicht auf einen bestimmten Typ, sondern kann einer beliebigen Komponente zugewiesen werden. Die Zuweisung dieser Stildefinition zu einer Komponente erfolgt über das

[2] dies ist in diesem Fall immer die Anwendung. CSS-Dateien können auch innerhalb von Komponenten definiert werden. Allerdings ist dann der Typ Selektor nicht gültig.

Attribut „*styleName*“. Der Name des Selectors nach dem „.“, dient dabei als Id und kann dementsprechend frei gewählt werden. Zugewiese Stileigenschaften, die eine Zielkomponente nicht besitzt, werden ignoriert. Im Listing 5.3 werden zwei Typen von Buttonschaltflächen angelegt. Innerhalb der Anwendung könnte man jetzt einem Button über das Attribut „styleName“ einen der beiden Typen zuordnen. Das Listing 5.4 zeigt den Code um z.B. einen der rot umrandeten, eckigen Buttons anzulegen.

Listing 5.3 Flex 3.0 Klassen-Selektor

```
.redBorderButton{
        cornerRadius: 0;
        borderColor: #FF0000;
}
.greenBorderButton{
        cornerRadius: 15;
        borderColor: #00FF00;
}
```

Listing 5.4 Flex 3.0 Zuweisung eines Styles

```
<mx:Button styleName="redBorderButton" />
```

- Global-Selektor - Der global Selektor ist die dritte Möglichkeit, eine Stil Definition zuzuweisen. Er wirkt wie der Name bereits aussagt global. Das bedeutet, dass dieser Selektor auf alle Komponenten wirkt. Nach Listing 5.5 würde jede Komponente die das Attribut „*color*“ besitzt für diese diese Stileigenschaft den Wert rot annehmen.

Listing 5.5 Flex 3.0 Global-Selektor

```
global{
        color: #FF0000;
}
```

Die unterschiedlichen Definitionen können leicht ineinander greifen. Es ist z.B. vorstellbar, dass man sich innerhalb einer Anwendung für weiße Rahmen entscheidet (die Rahmenfarbe wird global auf weiß gesetzt). Im Ausnahmefall möchte man aber gerne, dass Komponenten etwas mit einem roten Rahmen betonen (es wird ein Klassenselektor für einen roten

Rahmen erstellt). Es stellt sich die Frage welche der Einstellungen ist nun gültig. In diesem Fall greift eine festgelegte Reihenfolge in der die Stile angewandt werden. Es gibt folgende Hierarchie (absteigend, vgl. Tabelle 5.2).

1.	setStyle() zur Laufzeit
2.	Stile auf Instanzebene (Inline-Stile)
3.	Klassen-Selektoren
4.	Typ-Selektoren
5.	Parent Styles (vererbbare Styles)
6.	global-Selektor

Tabelle 5.2: Flex 3.0 Style Hierarchie

Zwei Punkte in der Tabelle 5.2 wurden bisher nicht erwähnt. Die Methode `setStyle()` wird zur Laufzeit aufgerufen und ermöglicht es, den aktuellen Stil zur Laufzeit zu ändern. Sie hat die höchste Priorität, falls verschiedene Stildefinitionen auf ein Element wirken. Des Weiteren gibt es in Flex für die Fonteinstellungen ein Vererbungsprinzip. Es besagt, dass Styles, die sich auf Fonts beziehen, auch auf die Kind Elemente eines Containers wirken. [Wid08]

5.1.2 Component-Skinning

Mit Stiles kann man eine Vielzahl visueller Anpassungen vornehmen. Trotzdem ist es möglich, dass diese nicht ausreichen um individuelle Vorstellungen umzusetzen. In diesem Fall kann man das Konzept des Component-Skinning nutzen. Bei diesem Vorgang wird das Erscheinungsbild einer Komponente grundlegend verändert, indem die grafischen Elemente ausgetauscht werden. Das Component Skinning ist dabei eigentlich ein Unterpunkt von Styles, da die Zuweisung der Grafiken ebenfalls über Stil-Eigenschafen stattfindet.

Welche Elemente nutzen Skins

Jedes Flex Steuerlement nutzt Skins. So lange vom Entwickler keine andere Zuweisung erfolgt sind das die Default Skins. Diese Default Skins sind über das Programmtic Skinning realisiert. Programmtic Skinning ist ein Verfahren, um Skins darzustellen und wird im nächsten Abschnitt genauer erläutert. Bei Komponenten, die verschiedene Zustände besitzen, wird für jeden Zustand eine eigene Skin-Eigenschaft angeboten. Es ist also

möglich, jeden dieser Zustände unabhängig von einander zu gestalten. Bei einem Button z.B. kann man dem Zustand „gedrückt“ über die Eigenschaft downSkin und dem Zustand „nicht gedrückt“ über upSkin ein grafisches Element zuweisen.

Möglichkeiten Skins zu erstellen

Es gibt drei unterschiedliche Formate, in dem ein Skin Element vorliegen kann. Diese werden nachfolgend kurz erläutert.

1. Pixel Grafik – in Form von *.jpg , *.png etc.
2. Vektor Grafik – in Form von *.swf.
3. Programmatic Skinning - ActionScript Klassen mit Zeichenanweisungen.

Bei Pixel- und Vektor-Grafiken wird dem entsprechenden Skin Attribut einfach das neue Grafik Element übergeben. Dabei ist zu beachten, dass Pixel-Grafiken als Skin Elemente einen Nachteil haben. Sollte die Grafik eine andere Abmessung haben als die zugewiesene Komponente, wird Flex die Grafik verzerren, um sie der Größe der Komponente anzupassen. Flex bietet allerdings ein Verfahren um dem entgegen zu wirken. Es heißt Scale9. Mit dem Scale9-Verfahren ist es möglich, die Grafik in Teile zu zerlegen, die bei Bedarf einzeln oder gar nicht skaliert werden. Dies hat zur Folge, dass bei einer Größenänderung keine Verzerrungseffekte auftreten oder diese weniger stark ausgeprägt sind.

Alternativ zu dem direkten Zuweisen von Grafiken kann man Skins über Programatic Skinning realisieren. In diesem Fall wird das Aussehen der Komponente über eine Actionscript-Klasse bestimmt. In dieser Klasse wird der Skin direkt über die Zeichenfunktionen von Flash kreiert. Es gibt eine vorbereitete Basis Klasse um einen Programmatic Skin zu erstellen. Sie heißt ProgrammaticSkin. Die Klasse beinhaltet die Methode `updateDisplayList()`, die überschrieben werden kann, um die eigene Visualisierung umzusetzen. Innerhalb dieser Methode ist es nun möglich, über die verschiedenen Zeichenfunktionen den Skin zu gestalten. Weiter Informationen findet man bei Waldminghaus [Wal09]. Der Vorteil der programmatischen Skins ist, dass man zur Laufzeit auf Veränderung reagieren kann. Zum Beispiel, wenn die Komponente z.B. skaliert werden soll oder sich der State ändern soll.

5.1.3 Organisieren von Stiles und Skins

Wurden für ein Projekt verschiedene visuelle Anpassungen vorgenommen, hat man die Möglichkeit, diese zusammenzufassen. Dazu gibt es die Möglichkeit der Themes und der Runtime CSS.

- Themes - Themes ermöglichen es, alle .css Files, grafischen Skin Dateien und programmatischen Skins in eine .swc File zu kompilieren. Der Flex Compiler kann dann aufgefordert werden, dieses Theme für die Applikation zu verwenden. Dies ist eine gute Option, falls ein Corporate Design vorliegt, das in verschiedenen Anwendungen genutzt werden soll [KL08].
- Runtime CSS - Runtime CSS sind dafür ausgelegt, eine Anwendung zur Laufzeit optisch anzupassen. Es ist möglich, eine .css Datei mitsamt den benötigten Ressourcen in eine .swf Datei zu kompilieren. Diese kann dann in der Anwendung mit Hilfe des StyleManagers geladen werden. Daraufhin verändert sich die visuelle Darstellung der Anwendung entsprechend der übergebenen .swf-Datei. Die Grafiken für Skins müssen normalerweise in eine Anwendung eingebettet werden. Das bedeutet, dass sie mit in die .swf Datei der Anwendung kompiliert werden. Mit Runtime CSS besteht die Option, diese Grafiken in einzelne zugehörige .swf Dateien aufzuteilen und diese dann bei Bedarf zu laden. Diese Option bietet sich an, wenn dem Nutzer zur Laufzeit die Möglichkeit geboten werden soll, zwischen verschiedenen visuellen Darstellungen der Anwendung umzuschalten.

5.2 WPF Stile und Templates

Die WPF bietet ebenfalls zwei Verfahren, um die visuelle Darstellung der Steuerelemente zu verändern: Stile und Templates. Beide Verfahren werden nachfolgend untersucht.

5.2.1 Stile

Ähnlich wie Flex bietet die WPF die Möglichkeit, Stileigenschaften von Steuerelementen zu verändern z.B. Farbenwerte für den Forder- oder Hintergrund. Diese können in XAML direkt inline übergeben werden. Für eine höhere Wiederverwendbarkeit bietet die WPF die Klasse Style an. Ein Objekt dieser Klasse enthält eine Sammlung von

Stileigenschaftszuweisungen. Es gibt zwei grundlegende Möglichkeiten, ein Style-Objekt einem Steuerelement zuzuweisen. Eine Variante ist, dem Style-Objekt eine Id über das Attribut *„x:Key“* zuzuweisen und diese Id der Eigenschaft *„Style“* eines Steuerelements zu übergeben. Die andere Möglichkeit bietet das Attribut TargetType des Style-Objekts. Mit diesem Attribut lässt sich ein Steuerelementtyp festlegen. Alle Steuerelemente dieses Typs nehmen dann die Stileigenschaften des Style-Objekts an. Falls ein Steuerelement über verschiedene Stilquellen beeinflusst wird, wirkt, ähnlich wie in Flex, eine Hierarchie. Lokale Einstellungen z.B. inline werden dabei globaleren Einstellungen vorgezogen. So ist es möglich z.B. alle Buttons einer Anwendung grün zu färben und einzelne Spezialfälle dann gezielt gelb zu färben. Zusätzlich zu diesen Möglichkeiten bietet die WPF die Option, unterschiedliche Stile, abhängig von Benutzerinteraktionen mit dem Steuerelement, zu definieren. Diese Möglichkeit wird über Trigger geboten.

WPF Trigger

Mit der Hilfe von Triggern lassen sich dynamische Aktionen innerhalb eines Style-Objekts definieren, die zur Laufzeit ausgelöst werden. Eine Trigger-Aktion kann das Umsetzen einer Stileigenschaft sein oder auch das Starten einer Animation beinhalten. Ein Trigger besitzt eine Bedingung, ist diese erfüllt wird seine Aktion ausgeführt. Es gibt drei unterschiedliche Arten von Triggern die kurz in Tabellenform (Tabelle 5.4) vorgestellt werden.

Property Trigger	wird ausgelöst wenn eine Eigenschaft einen bestimmten Wert annimmt. Zum Beispiel wenn *„isMouseOver“*, true entspricht.
Data Trigger	wird ausgelöst wenn eine über Databinding (siehe Kapitel Kapitel 2) verknüpfte Variable einen bestimmten Wert annimmt. Zum Beispiel falls der Inhalt eines Textfelds gebunden ist und in dieses Textfeld ein bestimmter String eingegeben wurde.
Event Trigger	wird ausgelöst sobald ein bestimmtes Event auftritt, zum Beispiel „MouseEnter“.

Tabelle 5.4: WPF Trigger

Zusätzlich ist es möglich, mehrere Triggerbedingungen zu kombinieren und eine Aktion nur auszulösen wenn alle Bedingungen erfüllt sind. Dazu stehen einem die Klassen Multi-Trigger für Property-Bedingungen und MultiDataTrigger für DataTrigger-Bedingungen

zur Verfügung. Wenn es nicht ausreicht die vorgebenen Stileigenschaften anzupassen, um die gewünschte Darstellung zu erreichen, gibt es über Templates die Möglichkeit, die visuelle Erscheinung grundlegend zu beeinflussen.

5.2.2 Templates

Die grundlegende Darstellung eines Steuerelements wird von Templates (Vorlagen) gesteuert. Es gibt drei wesentliche Templates die nachfolgend kurz erläutert werden. Dies ist das ItemsPanelTemplate, welches für die Anordnung der einzelnen Elemente innerhalb eines Steuerelements verantwortlich ist. Des weiteren bieten DataTemplates die Möglichkeit, die Darstellung von Daten zu beeinflussen, während das ControlTemplate die eigentliche Darstellung des Steuerelements repräsentiert.

- ItemsPanelTemplate - Dieses Template ist für Steuerelemente relevant, die vom Typ ItemsControl abgeleitet sind. Jedes dieser Steuerelemente besitzt bereits ein default-ItemsPanelTemplate. Die Aufgabe dieser Art von Steuerelementen ist das Darstellen von Daten. Ein typisches Beispiel dafür ist die List-Komponente. Über das ItemsPanelTemplate wird die Anordnung der einzelnen Elemente in einem solchen Steuerelement definiert. Dieses Template beinhaltet ein Panel Element, dem ein Layout Container übergeben werden kann. Bei einem Menü ist dies z.B. standardmässig ein Wrap Panel, wodurch die einzelnen Menüpunkte von links nach rechts, nebeneinander angeordnet werden. Über die Eigenschaft *„ItemsPanel“* kann dem Steuerelement ein eigenes ItemsPanelTemplate übergeben werden. Übergibt man dem Menü ein ItemsPanelTemplate, welches als Panel-Element ein Stackpanel enthält, werden die Menüpunkte untereinander angeordnet statt wie bisher nebeneinander.

- DataTemplate - Mit der Hilfe von DataTemplates kann man das Aussehen von Daten definieren. Das Template enthält eine visuelle Darstellung (visuell Tree[3]) und kann allen Steuerelementen die von ItemsControl abgeleitet sind übergeben werden. Es ist z.B. vorstellbar, dass man innerhalb einer Liste mit einem Datenobjekt arbeitet, das einen Pfad zu einem Bild und einen Namen mitliefert, beide Werte liegen in String-Form vor. Mit Hilfe des DataTemplates kann man jetzt eine beliebige Darstellung dieser beiden Werte realisieren. So wäre es möglich, mit Hilfe

[3]ein visuell Tree entspricht einer Hierarchie von verschachtelten Steuerelementen

eines Stackpanels, beide Werte untereinander anzuzeigen und den Bildpfad an eine Image-Komponente zu übergeben, sodass das eigentliche Bild angezeigt wird. Es würde dann statt einer String-Ausgabe ein beschriftetes Bild dargestellt werden.

- ControlTemplate - Das ControlTemplate ist für die Visualisierung eines Steuerelements verantwortlich. Es kann über das Attribut „`Template`“ an ein Steuerelement übergeben werden. Jedes Steuerelement besitzt bereits ein default-Template. Ansonsten würde es gar nicht angezeigt werden können, den die gesamte Visualisierung wird innerhalb des Templates definiert. Das Standardaussehen eines Steuerelements wird über den default-Style bestimmt. Innerhalb dieses Stils wird das ControlTemplate gesetzt. Das WPF-Framework bietet mehrere default-Styles. Diese werden Theme-Styles genannt. Sie entsprechen der Darstellung des eingestellten Windows-Themes des ausführenden Computers. In einem eigenen ControlTemplate kann eine beliebige Visualisierung erzeugt werden. Um die eigene Darstellung und die Logik eines Steuerelements zu verbinden, können die Eigenschaften des Steuerlements an die der visuellen Elemente gebunden werden. So ist es z.B. möglich auf die Eigenschaft der Vordergrundfarbe zuzugreifen und innerhalb der eigenen Visualisierung beliebige Elemente in dieser Farbe darzustellen, indem man ihre Farbeigenschaften an die der Vordergrundfarbe bindet.

5.2.3 Organisation von Stilen und Templates

Um eine hohe Wiederverwendbarkeit der Stilen zu gewährleisten bietet die WPF die Option, sie als logische Ressourcen innerhalb eines Resource Dictionary zu speichern. Man erhält Zugriff auf die Inhalte einer solchen Ressource-Datei indem man sie über das Attribut „`Ressources`“ innerhalb einer Anwendung referenziert. Ist die Datei referenziert, erhält man einfachen Zugriff auf die einzelnen Stile über ihre Id. Alternativ geschieht die Zuweisung für alle Komponenten eines bestimmten Typs indem keine ID vergeben wird sondern auf die Klasse des Typs verwiesen wird (vgl. Abschnitt 5.2.1). Templates werden meist ebenfalls innerhalb von Stilen zugewiesen. Der Grund dafür ist, dass sie nur über ID zugewiesen werden können. Man müsste sie also für jede Instanz einer Komponente separat definieren. Indem man sie innerhalb eines Stiles zuweist ist es über diesen Umweg möglich, sie auch einem Steuerelemententyp zuzuordnen.

5.3 Vergleich

Flex und die WPF bieten beide ein Konzept, um vorgegebene visuelle Eigenschaften eines Steuerelements zu verändern. Bei beiden Frameworks besteht die Möglichkeit, diese Eigenschaften für einzelne oder für Gruppen von Elementen zu definieren. Die WPF bietet hier zusätzlich den Vorteil, über Trigger auf sehr spezielle, durch Nutzereingaben herbeigeführte Zustände reagieren zu können und ermöglicht somit eine sehr flexible Anpassungmöglichkeit des Designs. In Flex kann diese Funktionalität ebenfalls erreicht werden, dies wäre allerdings recht umständlich. Über EventListner müsste die Anzeige für die jeweiligen Zustände manuell angepasst werden[4]. Um das visuelle Erscheinungsbild grundlegend zu ändern bietet Flex das Skinning. Skinning bedeutet das für die Zustände der Komponente die grafischen Elemente ausgetauscht werden. Die WPF liefert hier einen wesentlich flexibleren Ansatz. Über die einzelnen Templates hat man vielfälltige Möglichkeiten, die Darstellung der Elemente von Grund auf zu verändern. Gerade das Control Template ermöglich für jedes Steuerelement eine völlig freie Visualisierung. Flex bietet hier im Vergleich die Option, den einzelnen Zuständen der Komponente, jeweils eine eigene Grafik[5] zuzuweisen. Beide Frameworks bieten die Möglichkeiten die optischen Anpassungen extern abzuspeichern.

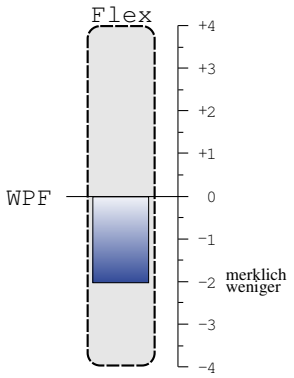

Abbildung 5.2: Vergleich Stile

Bewertung: Beide Frameworks bieten umfangreiche Möglichkeiten, die Steuerelemente visuell individuell anzupassen. Sowohl beim Spezifizieren der Darstellung über die Stil-Attribute als auch bei der Neugestaltung der Elemente bietet die WPF allerdings eine wesentlich breitere Palette an Möglichkeiten. Mit Hilfe der Trigger können bei der Darstellung sehr differenzierte Benutzereingaben berücksichtigt werden. Über das Konzept der Templates ist die WPF in der Lage, eine völlig freie Darstellung der Steuerelemente zu ermöglichen. Insgesamt bietet die WPF in dem Bereich individuelle

[4]Flex bietet bereits einige Stilattribute, die auch von der Benutzereingabe abhängig sind, zum Beispiel „`focusAlpha`“. Dies sind jedoch sehr wenige im Vergleich zu der Komplexität, die die WPF Trigger in Bezug auf die Benutzereingabe ermöglichen.

[5]die Programmatic Skins stellen keine Grafik dar, liefern aber in diesem Fall das gleiche Ergebnis.

Gestaltung von Steuerelementen erheblich mehr Möglichkeiten. Flex wird aus diesem Grund mit -2 (merklich weniger Leistungsumfang, vgl. Abbildung 5.2) im Vergleich zu WPF bewertet.

6 Prototypenentwicklung

6.1 Einleitung

Das Ergebnis der praktischen Arbeit sollte der Prototyp einer Rich Internet Applikation sein. Diese Anwendung sollte einen virtuellen Desktop[1] bieten auf dem der Benutzer die einzelnen Modulfunktionalitäten nutzen kann. Vorgesehen war dabei, Beispielmodule zu implementieren, die Echtzeitkommunikation ermöglichen und zusätzlich Module zu entwickeln, die diese Kommunikation unterstützen. Bei diesem Projekt gab es zwei Schwerpunkte. Einmal die Entwicklung der Softwarearchitektur des Modulkonzepts. Mit diesem Kriterium beschäftigt sich die Diplomarbeit von Mirko Kunath „Enwicklung einer ‚Rich Internet Application' mit Flex unter Verwendung von Architektur-Frameworks". Der praktische Schwerpunkt der eigenen Arbeit ist die Implementierung der grafischen Benutzeroberfläche des Projekts mit dem Anspruch, dabei ein breites Spektrum der von Flex gebotenen Funktionalitäten zu nutzen. Somit sollte ein praktischer Eindruck der Leistungsfähigkeit von Flex im Bereich der GUI-Entwicklung gewonnen werden

Im Folgenden wird zuerst eine Übersicht über die Funktionen des entwickelten Projekts gegeben. Dabei wird der Fokus auf die Entwicklung der grafischen Benutzerschnittstelle gelegt und aufgezeigt, welche Erwartungen an die Implementierung sich durch die theoretische Untersuchung„ entwickelt haben. Dazu werden die vier theoretischen Gliederungspunkte Steuerelemente, Layout, Animation und visuelle Anpassbarkeit einzeln betrachtet. Anhand eines repräsentativen Beispielmoduls wird gezeigt, wie die Funktionalitäten der einzelnen Bereiche implementiert wurden (vgl. Abbildung 6.1).

[1] ein virtueller Desktop bezeichnet in diesem Fall eine Fenster basierte Benutzerschnittstelle ähnlich Windows.

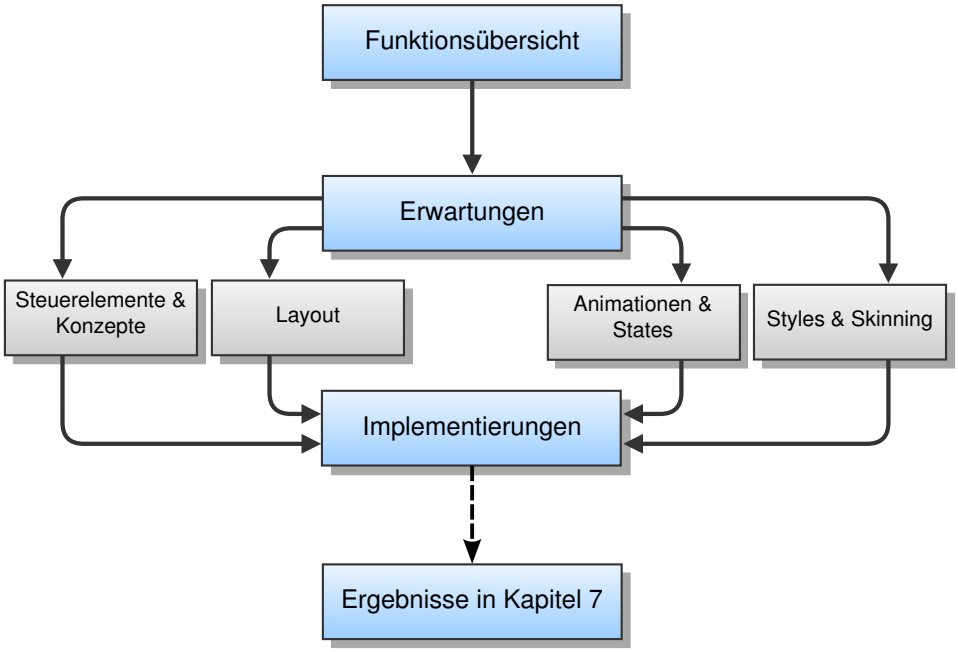

Abbildung 6.1: Übersicht Praxisbeschreibung

6.2 Funktionsübersicht der Anwendung

Der größte Teil der Funktionalität der Anwendung wurde in einzelne Module unterteilt. Die Anwendung bietet mit dem virtuellen Desktop eine Möglichkeit, diese Module zu nutzen und deren Erscheinungsbild mit Hilfe der auswählbaren Stile visuell anzupassen. An dieser Stelle soll eine Übersicht über die entwickelten Module und ihre wesentliche Funktionalität gegeben werden. Es wurden sechs Module entwickelt, die nachfolgend vorgestellt werden. Abbildung 6.2 zeigt diese Module (Login-, Global-Chat-, Instant-Messenger-, Videochat-, Notepad- und RSS-Modul) im Zusammenhang mit der erstellten Anwendung.

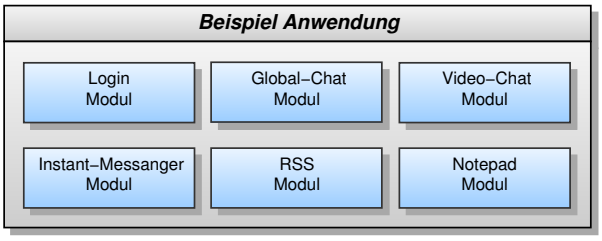

Abbildung 6.2: Module und die Beispiel-Anwendung

6.2.1 Login-Modul

Da ein Login-Prozess bei vielen Anwendungen benötigt wird, wurde die dafür benötigte Funktionalität als ein eigenes Modul ausgelagert. Dieses bietet einen Bereich zur Eingabe von Nutzerdaten undzusätzlich die Option, sich in einem gesonderten Bereich zu registrieren. Darüber hinaus gibt es einen Bereich zum Anfordern einer E-Mail, falls das Passwort vergessen wurde. Die Abbildung 6.3 zeigt den Loginbereich mit eingeblendeter Registrierungsoption.

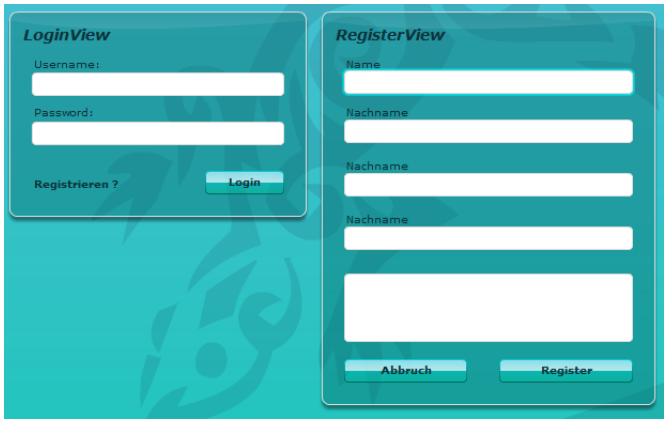

Abbildung 6.3: Login-Modul

6.2.2 Instant-Messenger-Modul

Dieses Modul ist für Nutzer gedacht, die regelmäßig die Seite besuchen und privat und in Echtzeit über diese kommunizieren wollen. Es bietet die Möglichkeit, andere Nutzer in einer Freundesliste zu speichern. Zusätzlich wird angezeigt, ob diese Nutzer gerade angemeldet sind und somit für Kommunikation zur Verfügung stehen. Um ein Gespräch mit einem Freund zu starten, kann man diesen anklicken. Es öffnet sich ein Chatfenster in dem der Gesprächsverlauf dargestellt wird und welches zusätzlich die Texteingabe ermöglicht. In dem Chatfenster, das ein Gespräch zwischen zwei Nutzern auf Textbasis repräsentiert, gibt es zusätzlich eine Schaltfläche die es ermöglicht über das Videochatmodul (Abschnitt 6.2.4) eine Videoverbindung zu dem Gesprächspartner aufzubauen. Innerhalb der Freundesliste bietet das Instant-Messenger-Modul zusätzlich die Option, eine Suchoberfläche einzublenden. Diese ermöglicht es, nach anderern Benutzer zu suchen und diesen dann der eigenen Freundesliste hinzuzufügen. Abbildung 6.4 zeigt das Instant-Messenger-Modul mit geöffneten Chatfenstern.

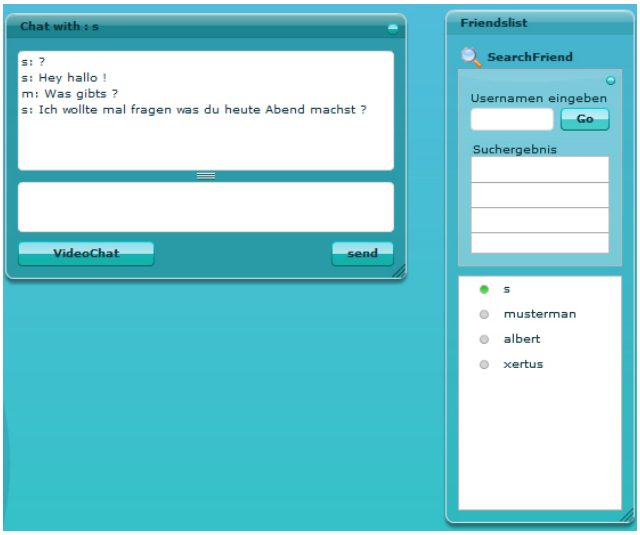

Abbildung 6.4: Instant-Messenger-Modul

6.2.3 Global-Chat-Modul

Dieses Modul zeigt jeden Nutzer, der sich aktuell auf der Webseite angemeldet hat, in einer Liste an und ermöglicht, miteinander über Texteingaben zu kommunizieren (Chat). Die Texteingaben werden dabei für alle Nutzer sichtbar in einem Fenster dargestellt. Dieses Modul bietet dem Nutzer die Möglichkeit, schnell und unkompliziert mit allen Anwesenden zu sprechen. Es bietet sich an für schnelle Anfragen oder Diskussionen. Abbildung 6.5 zeigt das Global-Chat-Modul mit mehreren angemeldeten Nutzern.

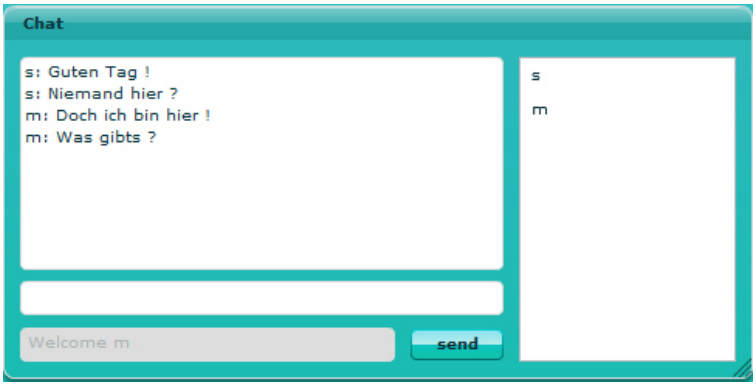

Abbildung 6.5: GlobalChat-Modul

6.2.4 Videochat-Modul

Das Videochat-Modul ermöglicht die Kommunikation zwischen zwei Nutzern mit Hilfe einer Video- und Audio-Verbindung. Die Funktionalität ist einem eigenen Modul gekapselt, hat aber keine eigene visuelle Darstellung (außer die Darstellung der Videofenster). Eingeleitet wird diese Kommunikation wie im Abschnitt 6.2.2 erläutert über das Instant-Messenger-Modul. Nach Initiierung erscheinen bei den beiden beteiligten Nutzern zwei Fenster, die jeweils das eigene Kamerabild und das Bild des Gesprächspartners zeigen. Ist keine Webcam vorhanden wird nur Ton übertragen. Abbildung 6.6 zeigt die zwei geöffneten Kamerafenster.

Abbildung 6.6: VideoChat-Modul

6.2.5 Notepad-Modul

Das Notepad-Modul stellt die Funktionalität eines kleinen Notizblocks zur Verfügung. Es ermöglicht die Eingabe von kleinen Textnotizen. Aus dem eingebenden Text wird eine Überschrift generiert, der zusätzlich das Erstellungsdatum hinzugefügt wird. Die Überschriften der einzelnen Notizen werden untereinander aufgelistet und können angeklickt werden, woraufhin die zugehörige Notiz angezeigt wird. Zusätzlich gibt es innerhalb der Notiz eine Schaltfläche, um diese zu löschen. Abbildung 6.7 zeigt das Notepad-Modul mit einigen eingefügten Notizen.

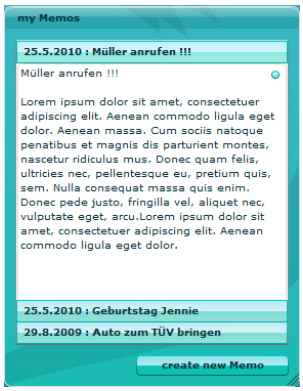

Abbildung 6.7: Notepad-Modul

6.2.6 RSS-Modul

Das RSS-Modul ermöglicht es, Text-, Audio- und Video-RSS-Feeds anzuschauen. Dazu erhält man eine Übersicht über verfügbare Anbieter (Channels die vorausgewählt wurden), der eigene Channels hinzugefügt werden können. Wählt man einen Channel aus, erhält man dessen Beschreibung und eine Auflistung der vorhandenen Beiträge (Feed). Wählt man nun einen konkreten Feed aus wechselt die Ansicht je nach Art des Feeds. Handelt es sich um einen Text-Feed wird dieser in einem Textfeld angezeigt. Bei Video- und Audio-Feeds erscheint jeweils ein einfacher Player, der den Beitrag abspielt. Abbildung 6.8 zeigt das RSS-Modul mit einer Auflistung von Feeds innerhalb eines Channels.

Abbildung 6.8: RSS-Modul

6.3 Erwartungen vor der Implementierung

Durch die theoretische Betrachtung hat sich vor der eigentlichen Implementierung eine Vorstellung von der Leistungsfähigkeit des Flex-Frameworks entwickelt. Davon abhängig haben sich Erwartungen gebildet, welche Funktionalitäten bei der Implementierung der grafischen Benutzerschnittstelle möglich sind und umgesetzt werden sollen. Um später einen Vergleich zwischen der praktischen Implementierung und den vorab entstandenen Vorstellungen (vgl. Abschnitt Kapitel 7) zu ermöglichen, werden die Erwartungen in die gleichen Bereiche wie die theoretische Untersuchung unterteilt (Steuerelemente und Konzepte, Layout, Animation & States und Styles & Skinning).

- Steuerelemente und Konzepte - Hier wurde erwartet, dass es mit Hilfe der Beschreibungssprache MXML sehr einfach und übersichtlich möglich ist, die Flex

Steuerelemente zu nutzen und zu konfigurieren. Außerdem sollten die vorgefertigten Steuerelemente eine breites Spektrum von auftauchenden Problemstellungen lösen können. Die unterstützenden Konzepte wie z.B. Databinding und Drag and Drop sollten eine komfortable Entwicklung ermöglichen.

- Layout - Flex bietet eine Vielzahl von Layout und Navigationscontainern. Dementsprechend sollte eine Anordnung der Steuerelemente, auch bei Veränderung des Layouts abhängig von Nutzereingaben (Interaktivität), flexibel möglich sein. Es wurde erwartet, dass die Anforderung des Projekts, ein dynamisches Fenstersystem zu entwickeln, erfüllt werden können.

- Animationen & States - Die Benutzerschnittstelle sollte mit Hilfe der Effekte leicht mit Animationen aufgewertet werden können. Zusätzlich wurde erwartet, dass die Nutzung von States ermöglicht, komplexere Veränderungen der Steuerelemente zu implementieren und diese zu animieren.

- Styles & Skinning - Es wurde erwartet, dass es komfortabel möglich ist, ein individuelles Design für die Anwendung zu erstellen. Dieses sollte auch zur Laufzeit variierbar sein.

6.4 Implementierungen

Im Rahmen dieser Diplomarbeit wurde die grafische Benutzerschnittstelle der beschriebenen Rich Internet Application entwickelt. Da die wesentlichen Prinzipien und Konzepte zum Implementieren der Oberfläche bei den einzelnen Modulen oft ähnlich sind, soll an dieser Stelle ein Beispielmodul im Detail untersucht werden. Auf dieser Weise soll ein repräsentativer Eindruck von der praktischen Arbeit mit dem Flex-Framwork vermittelt werden. Zur Unterscheidung der Funktionalitäten wird auf die vier eben skizzierten Hauptpunkte zurückgegriffen (vgl. Abschnitt 6.3). Für jeden dieser Punkte werden die wesentlichen Arbeitsschritte aufgezeigt und erläutert. Als Beispielmodul wird das Instant-Messenger-Modul (vgl. Abschnitt 6.2.2) ausgewählt, da es eine breite Funktionsvielfalt bietet. Die Anwendung verfügt aufgrund ihrer Gliederung durch ein Architektur-Framework (PureMVC) über einen speziellen Aufbau. Dieser Aufbau wird nur soweit erläutert, wie er für die Entwicklung der Benutzeroberfläche von Bedeutung ist.

Zum Verständnis wichtig ist die Tatsache, dass die Benutzerschnittstelle in einzelne Teile (je nach Aufgabe) aufgeteilt wird. Als Basis für einen solchen, abgetrennten Teil wird eine abgeleitete Container-Komponente benutzt (in diesem Fall meistens der Canvas-Container). In diesem Container werden die benötigten Steuerelemente nach Bedarf angeordnet. Der Vorteil liegt darin, dass dieser Teil der Benutzerschnittstelle wie eine eigene Komponente gehandhabt wird, welche innerhalb der Anwendung beliebig oft wiederverwendbar ist. Eine solche Komponente wird View genannt.

Vorstellbar ist zum Beispiel, dass man den Login-Bereich einer Anwendung zu einem Teil zusammenfassen möchte. Über den Flexbuilder kann man mit Hilfe der Option "neue MXML-Component" ein abgeleitetes Canvas erstellen lassen. Es entsteht eine leere MXML-Komponente. Nun stehen einem die gleichen Möglichkeiten wie in einer Anwendung zur Verfügung, innerhalb dieser MXML-Komponente, Steuerelemente für den Login-Bereich anzuordnen, z.B. zwei Textinput-Felder für Name und Passwort und einen Login-Button. Die entstandene Komponente wird nachfolgend als LoginView bezeichnen. Diese View enthält nun die Benutzeroberfläche des Login-Bereichs und kann beliebig in der Anwendung platziert werden. Innerhalb des benutzen Architektur-Frameworks gehört zu jeder View eine Actionscript-Datei. Diese enthält die Anwendungslogik zu diesem Teil der Benutzerschnittstelle (View) und wird als Mediator bezeichnet.

Nachfolgend wird gezeigt, wie die Aufteilung der Benutzerschnittstelle bei dem Instant-Messenger-Modul vorgenommen wurde.

Aufbau des Instant-Messenger-Moduls

Da das Instant-Messenger-Modul unter verschiedenen Gesichtspunkten untersucht werden soll, wird vorerst dessen grundlegende Struktur vorgestellt. Die Benutzerschnittstelle des Instant-Messenger-Moduls wurde in drei Views aufgeteilt, ChatUserList.mxml, Chat-Window.mxml und SearchUser.mxml (Abbildung 6.9 zeigt die drei Views des Instant-Messenger-Moduls). Aus Sicht des Anwenders repräsentiert die ChatUserList dabei die Basis. Sie enthält die Freundesliste und die Schaltfläche, um die Suchoption zu öffnen. Das ChatWindow stellt das Fenster dar, in dem ein Gespräch zu einem ausgewählten Freund stattfindet. Es enthält einen Bereich, in dem der Chat-Text dargestellt wird, eine Eingabe für Textnachrichten und zusätzlich eine Schaltfläche, um die Videokommunikation zu starten. Die SearchUser-View enthält ein Textfeld für die Namenssuche und eine Ergebnisliste.

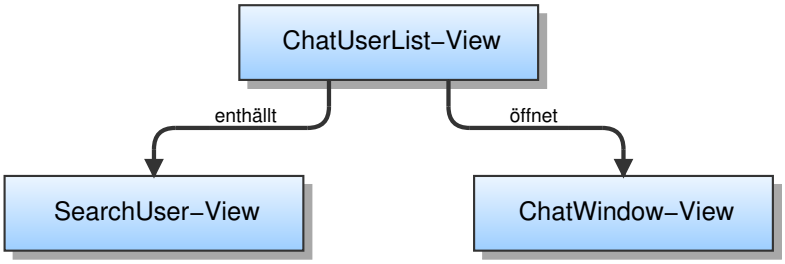

Abbildung 6.9: Instant-Messenger-Modul, View Aufteilung

Um eine gute Übersicht zu gewährleisten werden die Views (MXML-Dateien) in vier Bereiche abhängig von ihrer Aufgabe unterteilt: State-Definition, Transitions/Animationen, <mx:Script>-Tag (beinhaltet Actionscript-Code) und einem Definitionsbereich für MXML-Komponenten (vgl. Abbildung 6.10). Diese Teile sind nicht zwangsweise alle in einer View vorhanden, sondern werden nur erstellt, wenn auch Bedarf an entsprechender Funktionalität besteht. In den folgenden Abschnitten wird die Implementierung der einzelnen Views des Instant-Messenger-Moduls erläutert. Dabei wird jeweils der Fokus auf einen der vorgestellten Schwerpunkte gelegt.

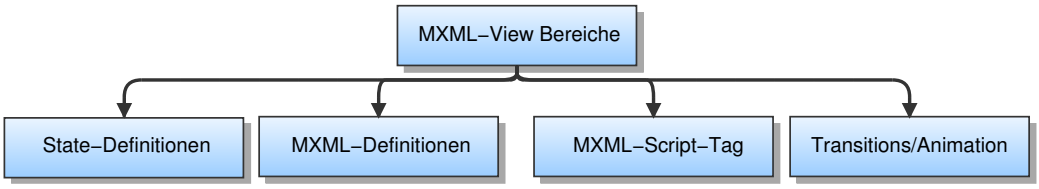

Abbildung 6.10: Aufteilung der Views

6.4.1 Steuerelemente und Konzepte

In diesem Abschnitt wird auf die Steuerelemente und dazugehörigen Konzepte eingegangen, die innerhalb der Views des Instant-Messenger-Moduls genutzt wurden. Daher liegt der Fokus auf den Definitionsbereichen der MXML-Komponenten und zugehöriger Funktionalität innerhalb des <mx:Script>-Tags.

ChatUserList-View

In dem Definitions-Bereich der ChatUserList.mxml wurden vier Komponenten angelegt. Die eigens erstellte View-Komponente SearchUser, der Linkbutton, das Datagrid welches die Freundesliste darstellt und zusätzlich eine Stylebinding-Komponente[2]. Um einen Eindruck von der MXML Beschreibungssprache zu bekommen wird der Defintionsbereich im Listing 6.1 gezeigt. Im Folgenden wird auf die Bedeutung der einzelnen Komponenten und ihrer Attribute eingegangen.

Innerhalb der ChatUserList.mxml wurde die SearchUser-View angelegt aber über das Attribut *"visible"* unsichtbar geschaltet, denn im Ursprungszustand soll die Suchoption nicht angezeigt werden. Da die Suchfunktion in eine eigene Komponente gekapselt wurde ist die eigentliche Funktionalität der Suchfunktion zu diesem Zeitpunkt nicht relevant. So lange die Search-User-View nicht sichtbar ist kann sie nicht genutzt werden. Sobald sie eingeblendet wird, steht ihre Funktionalität zur Verfügung.

Um die Suchfunktion anzuzeigen wurde eine Schaltfläche vom Typ Linkbutton genutzt. Um seine Funktion für den Benutzer zu verdeutlichen wurde zusätzlich zu seiner Beschriftung ein Icon darauf platziert. Um dies zu ermöglichen wurde innerhalb des Script-Tags ein Bild eingebettet und unter dem Namen "Icon" zugänglich gemacht. Dadurch ist es jetzt sehr einfach möglich, dem Linkbutton innerhalb des MXML Tags dieses Bild zuzuweisen. Die Zuweisung geschieht über das Attribut *"icon"*. In Listing 6.1 erkennt man das der Name des Bildes "Icon" noch einmal in geschweifte Klammern eingefasst wurde und dass dies auch bei anderen Wertzuweisungen innerhalb der MXML-Tags der Fall ist. Auf diese Weise ist es möglich, direkt innerhalb der MXML Definition Actionscriptcode zu verwenden und so z.B. auf Variablen zuzugreifen, die in dem <mx:Script> Tag angelegt wurden. Bei dem *"click"*-Attribut des Linkbuttons kommt dies erneut zum tragen. Dieses Attribut erwartet die Funktionalität die beim Klicken auf die Schaltfläche ausgelöst werden soll. Über eine Actionscript-Abfrage wird der Zustand der Komponente umgeschaltet, was zur Folge hat das die Suchfunktion eingeblendet wird. Dieser Vorgang wird mit der Stylebinding-Komponente im Abschnitt 6.4.3 über Animationen und States genauer erläutert.

[2] diese Komponente wird durch die externe Bibliothek, bigflexlib zur Verfügung gestellt http://code.google.com/p/bigflexlib/

Listing 6.1 ChatUserList.MXML, Ausschnitt Definitionsbereich 1

```
<components:SearchUser id="searchUser" height="173"
        right="0" left="0" y="19" visible="false" />

<mx:LinkButton label="SearchFriend"
        top="-2" left="-4" icon="{Icon}"
        click="{ (currentState==SEARCH_STATE )? currentState='':
                currentState = SEARCH_STATE }"/>

<mx:DataGrid id="friendsDatagrid"
        left="0" right="0" bottom="0" top="19"
        showHeaders="false" dataProvider="{friends}"
        click="{ friendsDataGridClick( event ) }"
        dragEnter="dragEnterHandler(event)"
        dragDrop="dragDropHandler(event)" dropEnabled="false">

        <mx:columns>
                <mx:DataGridColumn width="16" >
                        <mx:itemRenderer>
                        <mx:Component>
                        <mx:Canvas>
                                <customComponents:blinkingMail
                                        vis="{data.wantsToTalk}"/>
                        </mx:Canvas>
                        </mx:Component>
                        </mx:itemRenderer>
                </mx:DataGridColumn>

                <mx:DataGridColumn width="16">
                        <mx:itemRenderer>
                        <mx:Component>
                        <customComponents:onlineStatus
                                isOnline="{data.isOnline}"/>
                        </mx:Component>
                        </mx:itemRenderer>
                </mx:DataGridColumn>

                <mx:DataGridColumn labelFunction="friendName"/>
        </mx:columns>
</mx:DataGrid>

<styles:StyleBinding target="{friendsDatagrid}" style="top"
        value="{listTopProperty}" />
```

Über die DATAGRID-Komponente wurde die Anzeige der Freundesliste realisiert. Im Listing 6.1 ist erkennbar, dass die Definition des Datagrids wesentlich komplexer ist als die der anderen Komponenten. Zur Realisierung der DataGrid-Funktionalität wurde auf verschiedene zusätzliche Konzepte zurückgegriffen. Dies sind Databinding, Label-Funktionen, Item-Renderer und Drag and Drop. Diese Konzepte werden nach der Erläuterung der Definition des Datagrids vorgestellt.

Über das Attribut *"id"* wird dem Datagrid ein Name zugewiesen über den es mit Hilfe von Actionscript ansprechbar ist. Danach folgt über *"top"*, *"left"*, *"right"* und *"bottom"* die Positionierung des Datagrids. Darauf wird im Abschnitt 6.4.2, Layout genauer eingegangen. Mit Hilfe des Attributs *"showHeaders"* wurde die Spaltenbezeichnung des Datagrids ausgeschaltet, da diese als störend empfunden wurde. Damit geht die Funktion des Sortierens für den Nutzer verloren, da diese über einen Klick auf die Kopfzeile ausgelöst wird. Diese Funktion ist allerdings für diese Userliste verzichtbar.

- Databinding - Ein sehr wichtiges Attribut ist der *"dataProvider"*. Der Dataprovider erwartet die vom DataGrid darzustellenden Daten. In diesem Fall wird dem Dataprovider eine ArrayCollection (ähnlich einem Array) mit dem Namen friends übergeben. Diese ArrayCollection enthält die Liste der einzelnen Freunde und deren Metadaten, wie dem Online-Status[3] und dem wantsToTalk-Status[4]. Die ArrayCollection wurde innerhalb des <mx:Script> Tags angelegt und dort als "Bindable" (ermöglicht Databinding) und „public“ (öffentlich) deklariert. Durch die Übergabe an das *"dataProvider"*-Attribut ist sie an das DataGrid gebunden. Das bedeutet, dass das DataGrid informiert wird, falls Änderungen in der ArrayCollection stattfinden und es sich darauf hin automatisch aktualisiert. Für den Entwicklungsprozess bedeutet dies, dass der ChatUserList-Mediator jederzeit eine beliebig gefüllte Freundesliste in die ChatUserList.mxml schreiben kann und deren Anzeige sich dementsprechend aktualisiert. Dies ist sehr komfortabel, da der Mediator mit dem restlichen System verbunden ist und über Änderungen der Freundesliste informiert wird und jeweils eine aktuelle Version der Liste erhält. Nach dieser Erklärung zum Thema Databinding und dessen Nutzbarkeit, soll der Fokus jetzt auf die Darstellung der Daten über das DataGrid gerichtet werden. Würde man die friends-ArrayCollection so wie beschrieben an das DataGrid übergeben, würde es bereits zu

[3]Der Online-Status enthält die Information ob der Nutzer gerade eingeloggt ist.
[4]Der wantsToTalk-Status enthält die Information ob der Freund ein Gespräch beginnen möchte.

einer Anzeige der Daten kommen. Ohne weitere Konfiguration würde das DataGrid in diesem Fall den Objekt-Typ der ArrayCollection-Elemente überprüfen und für jede gefundene Eigenschaft die dieser Typ besitzt, eine Spalte anlegen und die entsprechenden Daten dort anzeigen. Die `friends`-ArrayCollection besteht aus Objekten vom Typ, `ChatFriendAndMetaData`. Das Objekt `ChatFriendAndMetaData` enthält zwei Boolean-Werte (`isOnline`, `WantsToTalk`) und ein weiteres Objekt friend vom Typ `ChatUserVO` welches den Login-Namen, Realnamen und Passwort enthält (vgl. Abbildung 6.11).

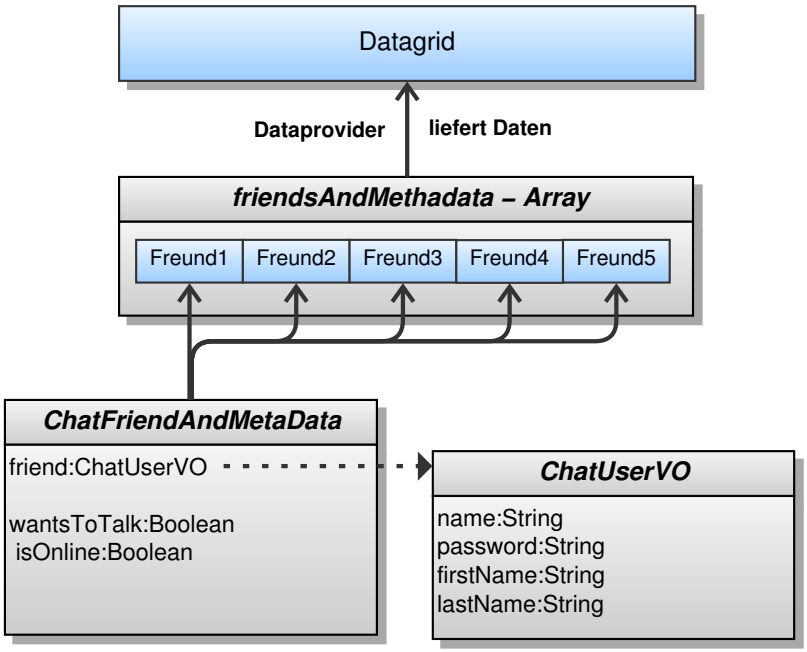

Abbildung 6.11: Dataprovider und Datentypen des ChatUserList Datagrids

Das DataGrid würde in diesem Fall drei Spalten anzeigen. In zwei Spalten würden die Boolean-Werte als Text aufgelistet werden und in der dritten Spalte würde sich der Text "[`object`]" wiederholen, da das DataGrid den Datentyp `ChatUserVo` nicht auflösen kann. Die gewünscht Anzeige sollte aber die Darstellung eines Icons für die Boolean-Werte und zugehörige die Anzeige des Login-Namens sein. Um dies zu erreichen wurde auf zwei Konzepte zugegriffen. Dies sind die bereits erwähnte Label-Funktionen und der ItemRenderer. Im Listing 6.1 sieht man, dass das DataGrid noch einen inneren Tag hat, den <mx:colums>-Tag. Innerhalb dieses Tags kann jede Spalte über den Tag <mx:DataGridColumn> einzeln konfiguriert werden.

Einstellbar ist z.B. Spaltenbreite, Headerbezeichnung oder welche Objekt-Variable der ArrayCollection in diesem Feld angezeigt werden soll. In diesem Fall wurde für die ersten beiden Spalten nur eine Breite festgelegt und es folgt jeweils ein weiterer Tag <mx:ItemRenderer>, um die Icons innerhalb des DataGrids darzustellen. Für die letzte Spalte wurde nur eine Funktion mit dem Namen friendName an das Attribut *"labelFunction"* übergeben.

- Label-Funktion - Die Funktion, die dem Attribut *"labelFunction"* übergeben wird (hier `friendName`), wird innerhalb des <mx:Script>-Tags angelegt und bekommt einen Parameter vom Typ Object übergeben. Dieses Objekt entspricht einem Datensatz der ArrayCollection. Die eigentliche Aufgabe einer Label-Funktion ist es, an Hand dieses übergebenen Datensatzes eine beliebige Textausgabe innerhalb der DataGrid-Spalte zu erzeugen (vgl. Abbildung 6.12).

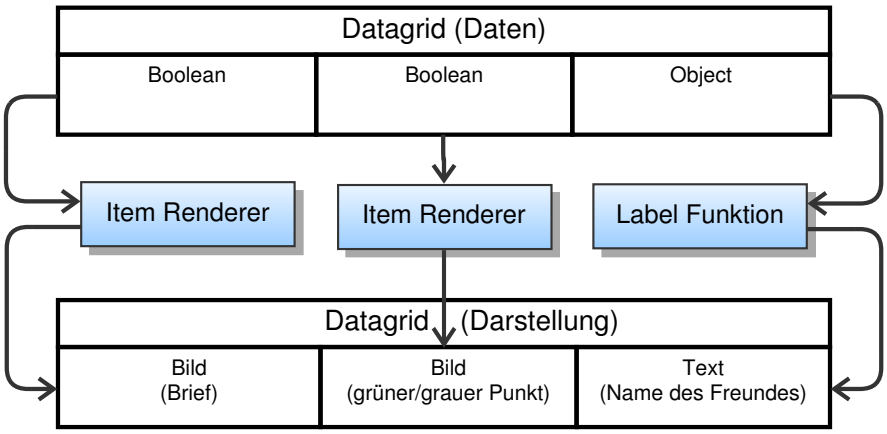

Abbildung 6.12: Label Function und Item Renderer des ChatUserList Datagrids

An dieser Stelle soll der Login-Name eines Nutzers in der Spalte angezeigt werden. Da die `friends`-ArrayCollection eine Sammlung von `ChatFriendAndMetaData`-Objekten darstellt, ist auch das übergebene Objekt von diesem Typ [5]Eigentlich ist das Objekt vom Typ Objekt es ist allerdings möglich, es auf den benötigten Datentyp zu casten. Es ist also kein Problem aus diesem Objekt den Login-Namen auszulesen. Dieser Name wird von der Label-Funktion zurückgegeben. In der Ausführung bedeutet dies, dass das DataGrid für jeden Datensatz die Label-Funktion aufruft.

[5]Eigentlich ist das Objekt vom Typ Objekt es ist allerdings möglich, es auf den benötigten Datentyp zu casten.

Die Funktion extrahiert den Login-Namen aus dem Datensatz, sodass dieser in dem DataGrid angezeigt werden kann. Mit Hilfe der Label-Funktion ist es also sehr komfortabel möglich, die eigenen Daten in ein beliebiges Format zu bringen.

- ItemRenderer - Ein ItemRenderer wird dann benötigt, wenn ankommende Daten nicht als Text sondern grafisch dargestellt werden sollen (vgl. Abbildung 6.12). Im Fall dieses Moduls sollte der Boolean-Wert `isOnline` durch ein runde Grafik dargestellt werden. Diese Grafik sollte grün erscheinen, falls der Benutzer online ist und grau falls nicht. Des Weiteren sollte für den Boolean-Wert `wantsToTalk`, im Fall true ein kleiner blinkender Brief angezeigt werden und im Fall von `false` keine Darstellung erfolgen. Diese Funktionalität lässt sich folgendermaßen implementieren. Es gibt grundlegend zwei ItemRenderer Konzepte, Drop-In Item Renderer und Inline-Item Renderer[6]. In diesem Fall wird ein Inline-Item Renderer verwendet. Listing 6.2 zeigt den Teilausschnitt aus Listing 6.1, in dem der ItemRenderer für den Online-Status implementiert wurde.

Listing 6.2 ChatUserList.MXML, Ausschnitt Definitionsbereich 2

```
<mx:DataGridColumn width="16">
        <mx:itemRenderer>
        <mx:Component>
        <customComponents:onlineStatus  isOnline="{data.isOnline}"/>
        </mx:Component>
        </mx:itemRenderer>
</mx:DataGridColumn>
```

Innerhalb des <mx:itemRenderer>-Tags wurde ein <mx:Component>-Tag angelegt. In diesem Tag ist es nun möglich, eine beliebige Komponente zu implementieren. Das kann auch ein Container sein, in dem entsprechende Kind-Komponenten angeordnet werden. Es ist also möglich komplexe Komponenten darzustellen. In diesem Fall wurde vorher eine separate Komponente mit dem Namen, onlineStatus erstellt, die nun innerhalb des <mx:Component>-Tags eingesetzt wurde. Die onlineStatus-Komponente beinhaltet eine graue und eine grüne Schaltfläche und zeigt abhängig von der Eigenschaft `isOnline` jeweils eine davon an. Nun ist es wichtig den Boolean-Wert des DataGrids an die Komponente zu übergeben. Dazu kann man die Eigenschaft `data` nutzen. Diese gewährt den Zugriff auf den Inhalt des

[6] detaillierte Informationen findet man bei Balderson et al.[BEH+09]

Datensatzes. Hier wird über `data.isOnline` ausgelesen, ob der Benutzer online ist oder nicht. Dieser Wert wird an die onlineStatus-Komponente übergeben, die nun ihre Anzeige entsprechend des Boolean-Wertes anpasst. Im Fall des `wantsToTalk`-Status ist die Implementierung ähnlich. Dort wurde ebenfalls eine Komponente erstellt, die entsprechend eines übergebenen Boolean-Wertes die Anzeige eine blinkenden Briefs liefert. An dieser Stelle soll noch erwähnt werden, dass es zusätzlich zu dem ItemRenderer noch das verwandte Konzept des ItemEditor gibt. Dieses ermöglicht ebenfalls die Anzeige von Komponenten innerhalb des DataGrids. In diesem Fall dienen diese Komponenten aber der direkten Eingabe von Daten. Die Funktionalität der Listenoberfläche ist an dieser Stelle vollständig vorgestellt. Freunde werden aufgelistet, es wird angezeigt, ob diese online sind und es ist möglich, einem Gegenüber zu signalisieren, dass man ein Gespräch führen möchte. Es soll nun gezeigt werden, wie man neue Nutzer zu dieser Liste hinzufügen kann. Eigentlich beinhaltet die SearchUser-View die dafür benötigte Funktionalität. Es wurde aber noch eine alternative Möglichkeit implementiert, um die Drag and Drop-Möglichkeiten des Flex-Frameworks zu testen. Diese Möglichkeit sieht vor, dass man Nutzer, die man im Global-Chat-Modul sieht, mit Hilfe von Drag and Drop direkt in seine Freundesliste ziehen kann.

- Drag and Drop - Drag and Drop beschreibt die Möglichkeit, über die Bewegung von visuellen Objekten durch den Nutzer bestimmte Aktionen auszuführen. In diesem Fall soll durch die Bewegung eines Benutzers aus der Liste des Global-Chat-Moduls in das Instant-Messenger-Modul ein Hinzufügen dieses Nutzers zur Freundesliste ausgelöst werden. Das Global-Chat-Modul bietet ebenfalls eine Nutzerliste, die über ein DataGrid realisiert ist. Dieses DataGrid[7] enthält aber andere Datenobjekte da es die Zustände wie `isOnline` und `wantsToTalk` nicht benötigt. Würden beide Listen die gleichen Datenobjekte führen, könnte man eine sehr einfache Standartvariante des Drag and Drop implementieren, indem man in dem Quellen-DataGrid dem Attribut *"dragEnabled"* und in dem Ziel-DataGrid dem Attribut *"dropEnabled"* jeweils `true` zuweist. Dieser einfache Weg stand in diesem Fall allerdings nicht offen. Deshalb wurde untersucht, welche Möglichkeiten Flex bietet, um diese Funktion manuell umzusetzen. Die Option des Herausziehens aus dem Global-Chat-DataGrid, die über das Attribut *"dragEnabled"* erreicht werden kann, ist für diesen Fall

[7] Eigentlich enthält die ArrayCollection des DataGrids die Datenobjekte.

bereits ausreichend. Das gewählte Listenelement wird neben dem Mauszeiger angezeigt (es wird gezogen). Außerdem zeigt ein kleines zusätzliches Symbol an, wo ein gültiger Ablageplatz für dieses Element ist. Ohne weitere Implementierungen wird jede Stelle der Bedienoberfläche als ungültiger Ablageplatz angezeigt. Es sollte nun erreicht werden, dass die Freundesliste als gültige Option zur Ablage angezeigt wird und der abgelegte Nutzer der Freundesliste hinzugefügt wird. Dazu wurden bei dem Ziel-DataGrid in der ChatUserList.mxml einige Attribute gesetzt (vgl. Listing 6.3). Es fällt auf, dass dem Attribut *"dropEnabled"* der Wert false zugewiesen wurde. Damit wurde die Standardfunktionalität ausgeschalten, die auf Grund der unterschiedlichen Listen-Daten-Typen zu einem Fehler führen würde. Zur Problemlösung wurden innerhalb des <mx:Skript>-Tags zwei Funktionen angelegt, `dragEnterHandler` und `dragDropHandler`. Diese Funktionen wurden jeweils einem entsprechendem Event, `dragEnter` und `dragDrop` zugewiesen. Diese Events treten ein, wenn ein gehaltenes Objekt über das Datagrid gezogen wird bzw. darauf abgelegt wird. Die zugehörigen Funktionen werden ausgeführt. Innerhalb der Funktionen ist es möglich, auf verschiedene Daten zugreifen, die das Event mitliefert, z.B. welches Element gehalten wurde. So kann überprüft werden, ob das empfangene Element den erwarteten Benutzernamen in sich trägt. Bei einem gültigen Element[8] wurde das entsprechende Symbol umgesetzt so das für den Anwender erkenntlich wird das er das Element hier ablegen kann. Beim Ablegen wird das übergebene Objekt gespeichert und das System über den Erhalt benachrichtigt.

Listing 6.3 ChatUserList.MXML, Ausschnitt Definitionsbereich 3

```
<mx:DataGrid id="friendsDatagrid" showHeaders="false"
        left="0" right="0" bottom="0" top="19"
        dataProvider="{friends}"
        click="{ friendsDataGridClick( event ) }"
        dragEnter="dragEnterHandler(event)"
        dragDrop="dragDropHandler(event)"
        dropEnabled="false">
</mx:DataGrid>
```

Nachfolgend wird erläutert, wie die Search-User-View das Hinzufügen von Freunden realisiert.

[8] Im Fall dieses Projektes gibt es nur eine Quelle von Drag and Drop. Deshalb ist eigentlich jedes Element gültig.

Search-User-View

Die bereits für ChatUserList-View verwendeten Konzepte finden auch in der Search-User-View Anwendung. Aus diesem Grund wird lediglich auf Besonderheiten und nur kurz auf die allgemeine Funktionsweise dieser View eingegangen. In dem Definitionsbereich der Search-User-View wurde mit Hilfe einer TextInput-Komponente, eines Label und eines Buttons die Eingabemöglichkeit für einen Suchtext realisiert. Der Suchtext wird an das System weitergegeben und Benutzer, die dem Suchtext entsprechen, werden zurückgeliefert[9]. Diese zurück gelieferten Benutzer werden in Form einer ArrayCollection verwaltet. Diese ArrayCollection ist, wie bereits die ArrayCollection in der ChatUserList-View, an ein DataGrid gebunden. Dieses DataGrid gibt die Suchergebnisse aus. Die Besonderheit ist, dass zu jedem Nutzer direkt innerhalb des DataGrids eine Schaltfläche aufgelistet wird, die es ermöglicht, diesen Benutzer der Freundesliste hinzuzufügen. Diese Funktionalität wurde wieder mit Hilfe eines ItemRenderers umgesetzt. Um herauszufinden, welche Schaltfläche gedrückt wird (es gibt ja nur eine Event-Handler-Funktion), wurde dem `"click"`-Ereignis zusätzlich zum Event noch die `data`-Eigenschaft übergeben, in der sich der aktuelle Datansatz befindet[10]. Dies wird in Listing 6.4 ersichtlich, in dem die Definition der Spalte mit den Schaltflächen gezeigt wird. Im data-Attribut ist die Information über den hinzugefügten Nutzer enthalten, die in der `addBtnClicked`-Methode ausgelesen und dann weitergeleitet wird. Zusätzlich wird in der Such-Ergebnisliste noch die Tooltip-Funktion des DataGrids verwendet. Mit dieser Funktion ist es möglich, für jede Zeile noch eine zusätzliche Ausgabe in Form eines Tooltips zu erzeugen, wenn man mit dem Mauszeiger über das entsprechende Element fährt. Dazu ist es möglich, eine Ausgabefunktion zu erstellen, die man dem DataGrid über das Attribut *"dataTipFunction"* übergeben kann. Das Konzept ist mit dem der Label-Funktion zu vergleichen. Leider wird der Tooltip ungünstig positioniert, so dass man den Wert in der DataGrid-Spalte nicht mehr lesen kann. Es wird kein Attribut angeboten mit dessen Hilfe eine Verschiebung des Tooltips möglich wäre. Als letzte View des Instant-Messenger-Moduls soll nun noch die ChatWindow-View betrachtet werden.

[9] Für den Prototyp wurde keine Datenbank implementiert. Der Web-Server simuliert Treffer in einer Datenbank und liefert diese als statische Werte zurück.

[10] Die data-Eigenschaft wurde in der ChatUserList-View bereits benutzt, um die Status-Boolean-Werte an die Icon-Komponenten zu übergeben (siehe Beschreibung ItemRenderer).

Listing 6.4 ChatUserList.MXML, Ausschnitt Definitionsbereich 4

```
<mx:DataGridColumn width="55">
        <mx:itemRenderer>
        <mx:Component>
        <mx:Canvas>
                <mx:Script>
                <![CDATA[
                import modules.chatModule.model.vo.ChatUserVO;
                ]]>
                </mx:Script>
                <mx:Button height="18" width="50" label="add"
                        click="outerDocument.addBtnClicked(
                        event, data as ChatUserVO)"/>
        </mx:Canvas>
        </mx:Component>
        </mx:itemRenderer>
</mx:DataGridColumn>
```

ChatWindow-View

Die ChatWindow-View ist aus Sicht der Steuerelemente und Konzepte nicht sehr interessant und entsprechend ist die Erläuterung dazu sehr kurz. Mit Hilfe von zwei Text-Area-Komponenten wurde sowohl die Texteingabe als auch Textausgabe realisiert. Ein Button bietet die Möglichkeit eingegebenen Text zu senden, ein anderer eröffnet die Möglichkeit, eine Anfrage für einen Videochat zu senden und somit das VideoChat-Modul zu nutzen.

6.4.2 Layout

In diesem Abschnitt wird auf das Layout der Anwendung eingegangen. Dazu wird kurz die spezielle Situation erläutert, die sich durch die genutzte Softeware-Architektur für das Layout ergibt. Danach wird auf die Layout-Struktur der Anwendung eingegangen und nachfolgend auf das Layout innerhalb der Module.

Grundsituation durch die Vorgabe der Architektur

Für die Architektur der Anwendung wurde das Architektur-Framework Pure-MVC verwendet. Zusätzlich wurden die Hauptfunktionen der Anwendungen zu Modulen zu-

sammengefasst, die jeweils eine eigene Pure-MVC Architektur beinhalten[11]. Durch diese Grundsituation ergab sich eine Problemstellung für das Layout der Module. Die Module sollten eine eigene Darstellung besitzen. Alternativ müsste man die Benutzeroberfläche der Module jedes Mal neu implementiert, sobald man sie in einer Anwendung benutzen möchte, was viel zu aufwendig wäre. Die Module selbst wiederum, wissen aber nichts über ihre Anordnung innerhalb der Anwendung, sollten aber natürlich frei positionierbar sein. Manche Module wie z.B. das Login-Modul würden aber, bei normaler Implementierung (innerhalb eines Containers) ihre Größe sehr stark verändern. Der Login-Bereich z.B. ist sehr klein, doch wenn man den Registrierungsbereich dazu einblendet wäre die Gesamtabmessung sehr groß. Zusätzlich würde sich der Registrierungsbereich immer an der gleichen Stelle relativ zum Login-Bereich befinden, was ein sehr unflexibles Layout der Anwendung zur Folge hätte. Aus diesem Grund wurde eine Architektur entwickelt, in der die einzelnen Module ihre Views (ihre Darstellung) beim Systemstart an die Anwendung übergeben[12]. Die Anwendung kann diese Views dann beliebig nutzen (positionieren, ein-/ausblenden, etc.) während die Module weiterhin autonom die Steuerung dieser Views übernehmen. Weitere Details können in der Diplomarbeit von Mirko Kunath [Kun10] nachgelesen werden. Insgesamt ergibt sich daraus die Situation, dass es einen äußeren Layoutprozess gibt, der die Anordnung der Modul-Views innerhalb der Anwendung regelt und einen inneren Layoutprozess der sich mit der Anordnung der Steuerelemente in den Modul-Views beschäftigt.

Das äußere Layout

Als äußeres Layout wird die Struktur der Benutzeroberfläche bezeichnet die, die Anordnung der Module in der Anwendung organisiert.

- Die Layoutstruktur der Anwendung - Innerhalb der Flex-Anwendung wurde eine View erstellt die, die Aufgabe hat den Hauptteil der Benutzeroberfläche zu strukturieren, um eine große Übersichtlichkeit innerhalb des Programms zu gewährleisten. Diese View ist die GUIManager.mxml. Die GUIManager.mxml ist vom Typ Canvas abgeleitet und hat somit keine eigene Darstellung. Sie erstreckt sich wie in Listing 6.5 sichtbar über den ganzen Anzeigebereich der Anwendung (Breite und Höhe jeweils 100%).

[11] Diese Funktionalität wurde von Mirko Kunath implementiert dessen praktischer Schwerpunkt die Architektur der Anwendung war.

[12] Im Fall von dynamisch erstellten Views werden diese erst zur Laufzeit zur Anwendung übergeben.

Listing 6.5 GUIManager

```
<views:GUIManager id="guiManager"
        x="0" y="0" width="100%" height="100%"/>
```

Der GUIManager selber enthält nur eine Viewstack-Komponente. Eine Viewstack-Komponente beinhaltet mehrere Container (mit Inhalt) zwischen denen umgeschaltet werden kann. Im Fall dieses Projekts wird hier zwischen den Hauptteilen der Anwendung umgeschaltet (vgl. Abbildung 6.13), also dem Login-Bereich (LoginUIContainer.mxml) der zum Start angezeigt wird und dem virtuellen Desktop (Desktop.mxml). Hier könnten bei Bedarf später weitere Teile hinzugefügt werden. Der Viewstack wird durch einen erfolgreichen Login-Prozess, vom Login-Bereich zu dem virtuellen Desktop umgeschaltet. Der virtuelle Desktop soll an dieser Stelle im Fokus stehen da dort die Anzeige des Instant-Messenger-Moduls realisiert wird.

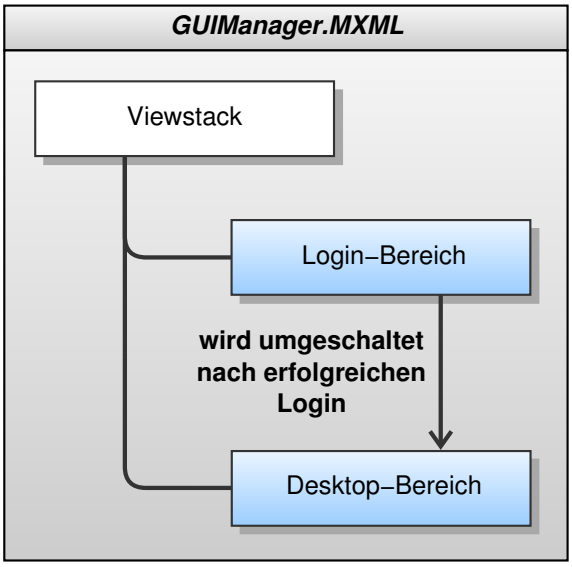

Abbildung 6.13: GUIManager, Viewstack

- Der virtuelle Desktop (Desktop.mxml) - Sobald man sich erfolgreich eingeloggt hat, wird der virtuelle Desktop eingeblendet. Dieser virtuelle Arbeitsbereich ist durch den Hintergrund definiert auf dem die verschiedenen Modul-Views in skalierbaren und positionierbaren Fenstern angezeigt werden. Um die Fenster-Funktionalität zu ermöglichen wurde auf eine zusätzliche Komponente der Webseite Factum Vicesimus

Quartus[13] zugegriffen. Diese Komponente heisst Superpanel. Das Superpanel ist vom Panel-Container abgeleitet und die Darstellung ist diesem ähnlich. Zusätzlich bietet es die Fensterfunktionalität z.B. skalieren, positionieren und schließen. Das Superpanel wurde für das Projekt modifiziert z.B. um die Visualisierung der Anwendung anzupassen. Für die einzelnen Module (vgl. Abschnitt 6.2) wurde nun jeweils ein leerer Superpanel-Container in der Desktop.mxml, angelegt[14]. Dabei wurde die Größe und die Position der Fenster für den Start festgelegt. Wird die Anwendung gestartet, erhält der Desktop-Mediator[15] nacheinander die einzelnen Views der Module. Er identifiziert diese und füllt die vorbereiteten leeren Superpanels nun mit den entsprechenden Views. Dabei wird die View-Höhe und -Breite jeweils auf 100% gesetzt, wodurch sich die Views, der Größe des Superpanel-Containers anpassen. Die Views der Module sind dabei so konzipiert, dass die Steuerelemente bei Größenänderung sinnvoll organisiert werden. Durch dieses Konzept ist in der Anwendung insgesamt ein sehr dynamisches und flexibles Layout entstanden. Im nächsten Abschnitt soll an Hand des Instant-Messenger-Moduls gezeigt werden wie das Layout innerhalb der Modul-Views umgesetzt wurde.

Das innere Layout

Das innere Layout bezieht sich auf die Benutzeroberfläche innerhalb der einzelnen Module. Für die Benutzeroberfläche der Module (Views) war ein wichtiges Kriterium, dass sie gut zu skalieren ist, da die Module später in dem größenveränderbaren Superpanel-Container dargestellt werden soll. Um zu zeigen wie diese Skalierbarkeit erreicht wurde, wird auf die ChatWindow-View des Instant-Messenger-Modul eingegangen. Die ChatWindow-View enthält zwei Button-Steuerelemente und einen VDividedBox-Container in dem zwei TextArea-Komponenten angeordnet sind. Der Container der hauptsächlich genutzt wurde, um die Views zu organisieren ist der Canvas-Container. Dieser Container bietet, mit Hilfe des Constrains-Konzepts, die Möglichkeit, die einzelnen Steuerelemente individuell anzuordnen[16]. Auch die ChatWindow-View nutzt als Basis ein Canvas-Container. Da

[13] http://www.wietseveenstra.nl/blog/2007/04/flex-superpanel-v15/

[14] Ausnahme sind dynamisch erstellte Fenster wie z.B. das ChatWindow des Instant-Messenger Moduls. Hier wird das Superpanel zur Laufzeit generiert.

[15] Zur Erinnerung: Der Mediator enthält die Programm-Logik der View und ist das Bindeglied zwischen der View und dem restlichen System.

[16] Es wäre auch vorstellbar, die gewünschte Anordnung über die verschachtelte Nutzung von Containern mit automatischem Layout zu erreichen.

dieser Container ein absolutes Layout nutzt, kann man alle Steuerelemente frei darin anordnen. In diesem Fall sollen die Steuerelemente aber auf die Größenänderung des Canvas reagieren. Daher wurde auf die Hilfe von Constrains zugegriffen. Das bedeutet, dass zur Positionierung der Steuerelemente keine x,y-Koordinaten genutzt wurden, sondern die Attribute *"left"*, *"right"*, *"top"*, *"bottom"* (vgl. Abschnitt 3.1.4). Mit diesen Attributen kann man das Steuerelement an die Kanten des Canvas binden, so dass das Steuerelement immer den angebenden Abstand zu der entsprechenden Kante hält. Aus der Definition, die im Listing 6.6 [17] gezeigt wird, ist z.B. ersichtlich, dass der send-Button sich immer in der unteren rechten Ecke befindet, da der Abstand zur rechten sowie unteren Container-Kante „0“ ist. Durch die Angaben von zwei Attributen die sich auf die gleiche Richtung beziehen, wie es bei der VDividedBox erkennbar ist (z.B. *"right"* und *"left"*), wird erreicht, dass diese Komponente ihre Breite und Höhe durch die Abstände zum Rand des Containers definiert. Das bedeutet, dass die Breite und die Höhe der VDividedBox sich der Größe der ChatWindow-View anpassen. Mit Hilfe der Constrains wurde also ein Layout erstellt, bei dem sich die Buttons immer in den unteren Ecken des Canvas befinden, während die VDividedBox und damit die TextArea-Steuerelemente den restlichen Platz ausfüllen. Skaliert man nun die gesamte ChatWindow-View, passen sich die darin enthaltenen Elemente der neuen View-Größe an. Somit wird der zur Verfügung stehende Platz auch bei unterschiedlichen Größen der ChatWindow-View sehr gut ausgenutzt.

Listing 6.6 ChatWindow.MXML, Definition der Steuerelemente

```
<mx:Button label="VideoChat" width="120" left="0" bottom="0"/>

<mx:Button label="send" width="55" right="0" bottom="0"  />

<mx:VDividedBox id="chatText" left="0" right="0" top="0" bottom="30">
        <mx:TextArea id="chatOutput" width="100%" height="75%"/>
        <mx:TextArea id="chatInput" width="100%"/>
</mx:VDividedBox>
```

[17] Alle fürs Layout unwichtigen Attribute wurden in diesem Listing entfernt. Es ist also nur ein Ausschnitt. Außerdem wird das Attribut *„bottom”* der VDividedBox im Projekt dynamisch gesetzt und wurde hier zur Verdeutlichung hinzugefügt.

6.4.3 Animationen & States

In diesem Kapitel wird gezeigt, wie Animationen und States in der Anwendung implementiert wurden. Dafür wird die ChatWindow.mxml des Instant-Messenger-Moduls betrachtet. In dieser View wurde beispielhaft eine Animation umgesetzt, die bewirkt, dass sich unwichtige Teile der ChatWindow-View ausblenden, wenn diese eine gewisse Zeit nicht benutzt wird. Verlässt die Maus den Bereich eines ChatWindow (MouseOut-Event) werden nach Ablauf einer Zeitspanne die Buttons ausgeblendet und nur noch die Textfelder dargestellt (vgl. Abbildung 6.14). Die Textfelder werden dabei auf die Größe der gesamten View skaliert. Sobald die Maus über das ChatWindow fährt (MouseOver-Event) werden die Buttons wieder eingeblendet. Die Idee dahinter ist, das die Buttons nicht benötigt werden solange man nicht aktiv im ChatWindow agiert und man durch das Ausblenden eine übersichtlichere Nutzeroberfläche erhält[18]

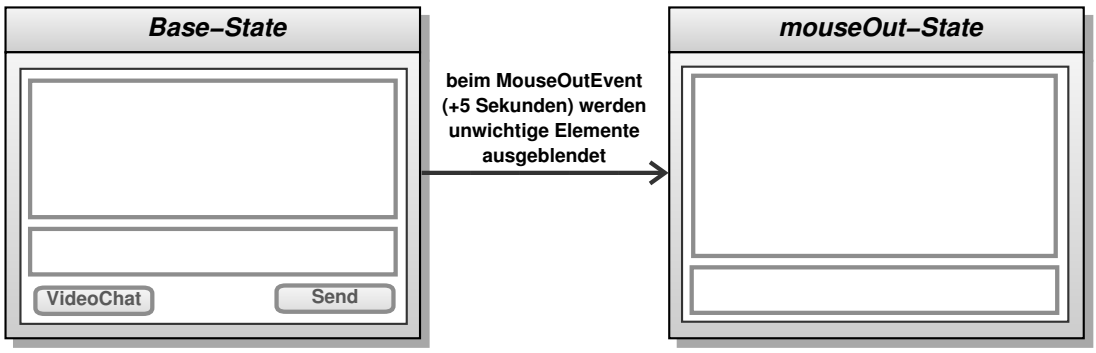

Abbildung 6.14: State Übergang

. Die erforderlichen Veränderungen werden durch Animationen weich dargestellt. Zur Implementierung wurden States, Transitions und Effekte genutzt. Außerdem wurde ein eigener Effekt erstellt, da die Buttons ausgeblendet werden sollten, aber der Standarteffekt "Fade" in diesem Fall nicht ausreicht.

Eigenen Effekt erstellen

Die zwei Buttons des ChatWindows werden über die Animation des Alphawertes aus- und eingeblendet. Allerdings gibt es in Flex Probleme, wenn man auf den Alpha-Wert von Schriften zugreifen möchte (die Buttons sind beschriftet). Denn entweder man integriert

[18] Dieses Konzept könnte man auch bei anderen Modulen nutzen, sodass auf dem virtuellen Desktop insgesamt, möglichst nur aktuell wichtige Elemente eingeblendet sind.

einen gesamten Schriftsatz in die Anwendung (einbetten), wodurch die Anwendung vergrößert wird oder man kann den Alpha-Wert der Schriften nicht beeinflussen. Das heißt im Fall der Buttons würde zwar die Schaltfläche langsam ausgeblendet werden, die Beschriftung würde jedoch ruckartig verschwinden. Da die Schriftart nicht eingebettet werden sollte, wurde ein eigener Fade-Effekt implementiert, der dieses Problem umgeht. Dieser Effekt legt zum Effektstart einen geringen Blur-Filter[19] auf das Animationsobjekt, blendet es anschließend ein oder aus und entfernt den Blur-Filter nach Ablauf der Animation wieder. Durch den angewendeten Blur-Filter wird die Anzeige des Animations-Objekts intern als Bild behandelt und die Schrift wird, wie gewünscht, ein- bzw. ausgeblendet. Nachfolgend wird genauer darauf eingegangen, wie diese Funktionalität für den Prototyp umgesetzt wurde.

Für diesen Anwendungsfall wurde ein Tween-Effekt erstellt. Ein Tween-Effekt zeichnet sich dadurch aus das, er eine onTweenUpdate-Methode besitzt. Diese Methode wird für jeden einzelnen Animationszwischenschritt aufgerufen (sehr häufig) und man kann innerhalb dieser Methode entsprechend Einfluss auf das Animationsobjekt nehmen. Effekte werden innerhalb des Flex-Frameworks über das Factory-Muster implementiert[20]. Das bedeutet, man erstellt ein Factory-Objekt was nach aussen hin aussieht wie ein Effekt. Eigentlich erstellt dieses Factory-Objekt erst zur Laufzeit eine Instanz des eigentlichen Effekts. Um den Tween-Effekt zu implementieren, wurde zuerst eine Klasse `FadeWithBlur` erstellt, die von `TweenEffect` abgeleitet ist (dies ist die Factory). In dieser Klasse kann jetzt der Typ des Effekts angegeben werden, den die Factory erzeugen soll. Zusätzlich können noch öffentliche Eigenschaften erstellt werden, die man später als Attribut in MXML an den Effekt übergeben kann.

Als von der Factory zu erzeugende Instanz wurde die Klasse `FadeWithBlurInstance` angeben und als Eigenschaften `alphaFrom` und `alphaTo` deklariert[21]. Die Klasse `FadeWithBlurInstance` ist von `TweenEffectInstance` abgeleitet und enthält die eigentliche Effekt-Logik. Im Wesentlichen wurden in dieser Klasse drei Methoden überschrieben, `play()`, `onTweenUpdate()` und `onTweenEnd()`. Die `play()`-Methode wird aufgerufen, sobald der Effekt gestartet wird. In dieser Methode wird der Blur-Filter[22] auf das Animationsobjekt gelegt. Danach startet mit Hilfe der Funktion `createTween` der Interpolationsvorgang.

[19] Ein Blur-Filter läst ein Element unscharf erscheinen.
[20] Dies ist ein Software-Muster. Es soll an dieser Stelle nicht weiter darauf eingangen werden. Die praktischen Konsequenzen werden erläutert.
[21] Die Werte dieser Eigenschaften werden einfach an die Effekt-Instanz weitergegeben.
[22] Der Blur-Filter ist so gering eingestellt, dass das Animationsobjekt weiterhin ganz normal aussieht.

Interpoliert wird zwischen den Werten `alphaFrom` und `alphaTo`. Für jeden neu errechneten Zwischenwert wird dabei die Methode `onTweenUpdate()` aufgerufen. In dieser Funktion steht einem mit der Eigenschaft `value` der aktuell berechnete Zwischenwert zur Verfügung. Dieser wird an die `alpha`-Eigenschaft des Animationsobjekts übergeben, wodurch die Transparenz über die Zeit verändert wird. Ist der Interpolations-Endwert erreicht, wird über die Methode `onTweenEnd()`der Blur-Filter wieder entfernt. Innerhalb der Anwendung kann dieser Effekt jetzt wie gewohnt in MXML genutzt werden. Angesprochen wird der Effekt über den Namen seiner Factory (`FadeWithBlur`).

Implementierung der Animation

Nachfolgend wird auf die Implementierung der Animation des erstellten Prototypen am Beispiel der ChatWindow-View eingegangen. Dabei wird gezeigt wie States und Transitions (Effekte) im Prototypen angewandt wurden. Außerdem zeigt dieser Abschnitt welche Problemfelder bei der Implementierung aufgetreten sind und wie diese gelöst wurden.

- States - Das Ziel der Veränderung innerhalb der View ist es, zwei Schaltflächen auszublenden sowie einen Container zu vergrößern (vgl. Abbilung 6.14 (der Übergang zwischen den Fenstern). Dies schien ein ideales Szenario für den Einsatz von States und zugehörigen Transitions zu sein. Daher wurde zur Umsetzung ein State mouseOut angelegt. Dieser State repräsentiert die View-Darstellung mit ausgeblendeten Schaltflächen. Listing 6.7 zeigt die Definition dieses States, in dem die zwei Schaltflächen entfernt werden und ein Style-Attribut des chatText-Containers einen Wert von "0" zugewiesen bekommen. Der Wert des *"bottom"* definiert den Abstand des Containers (dieser enthält die Textfelder) zum unteren Rand der Komponente (Constrain). In dem Zustand `mouseOut`, ist der Abstand "0", der Container und damit die Textfelder füllen also den Platz bis zum unteren Rand des Containers aus. Im Ursprungszustand ist der Abstand zur Unterkante "30" groß Die Komponente lässt damit im Ursprungszustand Platz für die zwei Schaltflächen. Die Zustände könnten nun bereits mit Hilfe der `setCurrentState`-Methode umgeschalten werden und die gewünschte Änderung würde eintreten. Allerdings ohne Animationen.

Listing 6.7 ChatWindow.MXML, State-Definition

```
<mx:states>
        <mx:State name="mouseOut">
                <mx:RemoveChild target="{sendButton}"/>
                <mx:RemoveChild target="{videoChatButton}"/>
                <mx:SetStyle target="{chatText}" name="bottom" value="0"/>
        </mx:State>
</mx:states>
```

- Transistions und zugehörige Effekte - Um die Übergänge zwischen den States zu animieren wurden Transitions angelegt. Für jeden der beiden Übergänge wurde jeweils ein Effekt zugewiesen in dem die Animationen des Übergangs definiert wird (siehe Listing 6.8)[23].

Listing 6.8 ChatWindow.MXML, Transitions

```
<mx:transitions>
        <mx:Transition effect="{toFadeOut}"
                fromState="" toState="mouseOut"/>
        <mx:Transition effect="{toFadeIn}"
                fromState="mouseOut" toState=""/>
</mx:transitions>
```

Im Folgenden soll nun der `toFadeOut`-Effekt beleuchtet werden. Der `toFadeIn`-Effekt wird ausgeklammert, da die Schwerpunkte in den beiden Effekten ähnlich sind. Listing 6.9 zeigt, dass der `toFadeOut`-Effekt eine Sequenz ist, in der mehrere Teileffekte nacheinander ablaufen. Der `toFadeOut`-Effekt steht für den Übergang, bei dem die Schaltflächen ausgeblendet werden und der `ChatText`-Container sich ausdehnt. Als erster Schritt der Animation werden die Schaltflächen ausgeblendet. Hier wurde der selbst erstellte `FadeWithBlur`-Effekt genutzt, um diesen Effekt in die Sequenz einzufügen. Hier trat bereits das erste Problem auf. Beim Umschalten von States werden erst alle Änderungen die der State mit sich bringt durchgeführt und erst dann die Animationen der Transistions. Da aber die Schaltfläche im State `mouseOver` entfernt wurde konnte sie nicht mehr animiert (ausgeblendet) werden. Für diesen Fall, bietet Flex, Action-Effects. Diese bieten die Möglichkeit in die Ablaufreihenfolge der Operationen einzugreifen. Es soll an dieser Stelle

[23] Hinweis: Über einen leeren String ("") wird der BaseState angesprochen.

aus Platzgründen nicht im Detail darauf eingegangen werden[24]. Wichtig ist, dass mit dem <mx:RemoveChildAction>-Tag, in der Sequenz festgelegt wird, dass die remove-Operationen der Schaltflächen zeitlich als zweite Aktion nach dem Ausblenden erfolgen. Mit Hilfe dieses Tags verläuft das Ausblenden erfolgreich.

Listing 6.9 ChatWindow.MXML, Definition toFadeOut-Effekt 1

```
<mx:Sequence id="toFadeOut" startDelay="5000">
        <customEffects:FadeWithBlur  duration="400" startDelay="0"
                targets="{[sendButton,videoChatButton]}"
                alphaFrom="1" alphaTo="0"/>
        <mx:RemoveChildAction targets="{[sendButton,videoChatButton]}"/>
        <mx:AnimateProperty property="bottom" target="{chatText}"
                isStyle="true" duration="1000" fromValue="30" toValue="0"/>
</mx:Sequence>
```

Ab diesem Zeitpunkt wurde der Entwicklungsprozess der Übergangseffekte etwas unangenehm. Ursprünglich wurde nun als letzter Tag ein AnimateProperty-Tag in die Sequenz eingefügt (siehe Listing 6.9). Dieser AnimateProperty-Effekt beeinflusste das Attribut *"bottom"* des chatText-Containers und hat für die Ausdehnung des Containers gesorgt. Die Übergänge wurden an dieser Stelle korrekt animiert und die gesamte Animation war eigentlich fertig. Allerdings wirkte etwas störend. Die Animation hatte insgesamt eine Dauer von 1,4 Sekunden, was recht lang ist. Wurde in dieser Zeit durch eine schnelle Bewegung der Maus ein erneuter Statewechsel eingeleitet, wurde die Animation des Übergangs abgebrochen und die gegenläufige Übergangsanimation gestartet. Dadurch entstand ein starker Sprung der Animation. Dieser Sprung kam zu Stande, da die Effekte der Übergangssequenzen, Startwerte besitzen (z.B. *"alphaFrom"*). Dieser Startwerte waren statisch und entsprachen den Endwerten der gegenläufigen Animation (siehe Listing 6.9). Da die Animation der ChatWindow.mxml das Nutzerempfinden verbessern sollten, wurde eine solch sprunghafte Animation als nicht tragbar angesehen. Die Lösung dieses Problems stellte sich als recht komplex heraus und soll hier beschrieben werden.

- spezielle Problemlösung - Als erstes sollte erreicht werden das die Startwerte der Animationen z.B. *"alphaFrom"* oder *"fromValue"* den aktuellen Werten der Steuerelemente entsprechen. Wie im Listing 6.10 ersichtlich ist das bei dem

[24] Weiterführende Informationen bei Waldminghaus [Wal09]

FadeWithBlur-Effekt kein Problem. Mit Hilfe der geschweiften Klammern wird (bei dem Attribut *"alphaFrom"*) einfach auf den aktuellen Alpha-Wert der sendButton-Schaltfläche zugegriffen. Anders bei dem Style-Attribut *"bottom"* des chatText-Containers. Auf diesen Wert konnte nicht zugegriffen werden, da es sich um ein Style-Attribut handelt und Stile nur über die getStyle()-Methode abrufbar sind. Zur Lösung sollte das *"bottom"*-Style-Attribut, mit der Hilfe von Databinding, an eine Variable gebunden werden und diese dann animiert werden, so dass sich der *"bottom"*-Wert immer dem Wert dieser Variable anpasst. Es stellte sich aber heraus das Style-Attribute nicht gebunden werden können. Also wurde eine externe Bibliothek, bigFlexLib zur Hilfe genommen. Diese Bibliothek bietet über die Klasse StyleBinding die Möglichkeit des Bindings für Style-Attribute. Dieser Umweg ergab den MXML-Code der im Listing 6.10 zu sehen ist. In diesem Listing wird die Variable chatTextBottomValue animiert, statt dem *"bottom"*-Styleattribut im Listing 6.9. Die Startwerte der Animationen waren an dieser Stelle erfolgreich angepasst. Allerdings sprang die Animation noch immer. Die in den State-Definition angebenen Werte für das Style-Attribut *"bottom"* (vgl. Listing 6.7) wurden beim Statewechsel unabhängig von den Startwerten der Animationen der Transitions übernommen. Also wurde der <SetStyle>-Tag aus der Statedefinition entfernt. Dieser Schritt hatte zur Folge das nicht mehr alle Veränderung zwischen den States in der State-Definition vorhanden waren, was eine große Unübersichtlichkeit des Codes zur Folge hatte. Trotzdem ruckelte die Animation an diesem Punkt weiterhin. Die endgültige Lösung lag darin, die Effekt-Sequenzen (toFadeIn und toFadeOut) zusätzlich bei einem State-Wechsel mit Hilfe der Methode stop() manuell anzuhalten.

Listing 6.10 ChatWindow.MXML, Definition toFadeOut-Effekt 2

```
<mx:Sequence id="toFadeOut" startDelay="5000">
        <customEffects:FadeWithBlur duration="400"
                startDelay="0" targets="{[sendButton,videoChatButton]}"
                alphaFrom="{sendButton.alpha}" alphaTo="0"/>
        <mx:RemoveChildAction targets="{[sendButton,videoChatButton]}"/>
        <mx:AnimateProperty property="chatTextBottomValue"
                target="{this}" fromValue="{chatTextBottomValue}"
                duration="1000" toValue="0" isStyle="false"/>
</mx:Sequence>
```

Auf Grund dieses recht aufwendigen Arbeitsprozesses wurde darauf verzichtet, diese Funktionalität auch in anderen Modulen umzusetzen.

6.4.4 Styles & Skinning

Wie bereits im Kapitel 5.1 beschrieben können in Flex Stil-Eigenschaften verändert werden, um das visuelle Erscheinungsbild von Steuerelementen zu verändern. In diesem Projekt wurde eine Sammlung dieser Veränderung in CSS-Dateien zusammengefasst. Diese Sammlungen von Einstellungen werden in dieser Diplomarbeit als Themes bezeichnet. Der Anwender kann zur Laufzeit ein Theme auswählen, worauf hin die entsprechenden CSS-Einstellungen geladen werden und sich die Darstellung der Anwendung ändert.

Grundsituation

Die Funktionalität des Theme-Wechsels wurde direkt in der Anwendung und nicht innerhalb der einzelnen Module implementiert. Es ist also nicht möglich, die Module individuell zu gestalten. Das hat einen technischen Hintergrund. Die Gestaltung wird mit Hilfe von CSS-Dateien und entsprechenden Selektoren realisiert. Ein sehr wichtiger Selektor, der Typ-Selektor kann aber nur direkt in der Anwendung verwendet werden. Das bedeutet, ein korrekter Verweis auf eine CSS-Datei die Typ-Selektoren enthält, funktioniert nur innerhalb der MXML-Datei die von dem Application-Container abgeleitet ist. Aus diesem Grund wird in diesem Abschnitt nicht auf das Instant-Messenger-Modul sondern auf die DiplomPraxis.mxml und die zugehörigen CSS-Dateien eingegangen.

Die Basis CSS-Datei

Direkt beim Start wird der Anwendung als Style, die CSS-Datei FreshGreen.css zugewiesen (siehe Listing 6.11). Dieses Theme spielt eine Sonderrolle, denn in dieser CSS-Datei werden grundlegende Zuweisungen gemacht, die zur gesamten Laufzeit der Anwendung gültig sind, z.B. ob innerhalb eines DataGrids Trennlinien angezeigt werden oder welche Icons die Fensterfunktionalität des Superpanels visualisieren. Die anderen Stile fügen später nur ihre entsprechenden Veränderungen hinzu oder überschreiben vorhandene Einstellungen.

Listing 6.11 CSS-Zuweisung

```
<mx:Style source="./application/assets/css/FreshGreen.css"/>
```

Erstellung der Themes

Drei der Themes (CSS-Dateien) wurden selber erstellt. Sie variieren die Darstellung der Anwendung durch die Nutzung der vielen Stil-Attribute die das Flex-Framework zur Verfügung stellt. Die Einstellungen waren umfassend und vielseitig und es wurden alle Arten der Selektoren (vgl. Abschnitt 5.1) genutzt, um eine differenzierte Anpassung zu ermöglichen. Eine Auflistung der Veränderung ist an dieser Stelle aus Platzgründen nicht möglich. Abbildung 6.15 zeigt jedoch den Unterschied zwischen dem Standartdesign von Flex und der Darstellung durch das FreshGreen-Theme am Beispiel des Login-Bereichs.

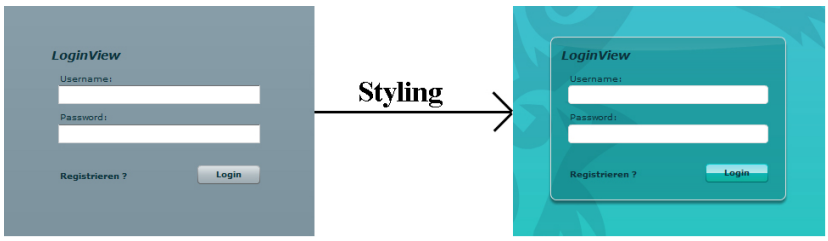

Abbildung 6.15: Themes und Styling

Ein viertes Theme wurde auf der Basis des Component-Skinnings erstellt. Die visuellen Elemente der Komponenten wurden also durch Grafiken ersetzt. Den Arbeitsprozess zum Erstellen dieser grafischen Elemente unterstützt Adobe mit vielen seiner Grafikprogramme, z.B. Illustrator, Photoshop oder Flash. Im Fall von Photoshop bietet Adobe z.B. eine vorgefertigte Schablone in der die Grafiken für alle Steuerelemente und deren Zustände aufgelistet sind. Diese können einzeln editiert und auch korrekt exportiert werden. Da Component-Skinning nur einen kleinen Teil dieser Diplomarbeit einnimmt aber verhältnismäßig aufwendig zu erstellen ist, wurde kein eigener Component-Skin erstellt sondern auf ein bereits vorhandenes Theme zurückgegriffen[25].

Zuweisung der Themes zur Laufzeit

Um die Themes und damit die Darstellung der Anwendung zur Laufzeit zu wechseln, wurde auf das Konzept der Runtime-CSS zugegriffen. Es ist möglich die CSS-Dateien mit den benötigten Grafiken (falls Grafiken in der CSS-Datei zugewiesen werden) zu einer .swf-Datei zu kompelieren. Der Flexbuilder ermöglicht dies über die Option: "Compile CSS to SWF". Auf diese Weise war es einfach, das komplexe herunter geladene Theme in das

[25] http://scalenine.com/

eigene Projekt zu integrieren, indem die CSS-Daten zu einer .swf-Datei kompilierte wurde und in den entsprechenden Ordner der Anwendung kopiert wurde. Das Flex-Framework bietet die Klasse StyleManager mit deren Hilfe die Theme-.swf-Dateien geladen werden können und die Darstellung der Anwendung entsprechend aktualisiert wird. Zusätzlich ist es auch möglich, ein Theme wieder zu entfernen. So kann bei Auswahl eines Themes zur Laufzeit sehr komfortabel das alte Aussehen entfernt und das neue geladen werden.

7 Zusammenfassung

Ziel dieser Arbeit war es Herauszufinden, welche Funktionalitäten das Flex-Framework zur Verfügung stellt um eine leistungsstarke und moderne GUI zu entwickeln. Dazu wurde diese Arbeit in eine theoretische Untersuchung mit einem Vergleich zur WPF und eine darauf aufbauende, prototypische Implementierung unterteilt. Die Untersuchungsergebnisse beider Teile werden in diesem Kapitel vorgestellt und anschließend zusammengefasst.

7.1 Zusammenfassung des Theorieteils

In der Theorie sollten die technischen Möglichkeiten von Flex untersucht und im Vergleich zur WPF bewertet werden. Dazu wurden vier wesentliche Schwerpunkte untersucht und einzeln bewertet (vgl. Abbildung 6.15).

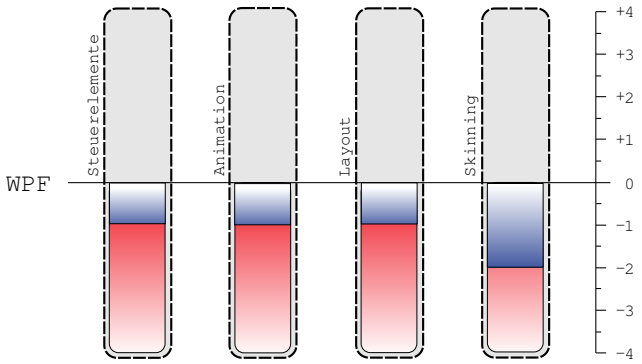

Abbildung 7.1: Gesamtbewertung

Wie man dieser Abbildung erkennen kann, wurde Flex dabei jedes Mal schlechter als die WPF bewertet. Daraus könnte man schlussfolgern, dass Flex nicht besonders geeignet scheint um GUI's zu entwickeln. Dies ist aber nicht der Fall. Folgende Punkte

müssen bei dieser Bewertung berücksichtigt werden: Grenzen des theoretischen Vergleichs, Veröffentlichungszeitpunkte, Relation zwischen WPF und Flex.

7.1.1 Grenzen des theoretischen Vergleichs

Da es eine theoretische Untersuchung war, wurde hauptsächlich der Leistungsumfang der Frameworks untersucht. Um eine vollständige Beurteilung für einen praktischen Einsatz geben zu können müssten aber zusätzlich noch andere Punkte berücksichtigt werden. Relevant wären zusätzlich z.B. Kriterien wie Einsteigerfreundlichkeit oder Einsatzeffizienz der verfügbaren Funktionalitäten. Die Untersuchung und der Vergleich dieser Kriterien würde aber zusätzlich umfangreiche praktische Tests erfordern und wird deshalb in dieser Arbeit vernachlässigt. Grundsätzlich sind diese Aspekte jedoch oft gegensätzlich zur Komplexität eines Frameworks. WPF wird z.B. ein schwerer Einstieg nachgesagt, während bei Flex auffällt, dass oft eine einfache, dafür aber einsteigerfreundliche Möglichkeiten angeboten wird bestimmte Funktionalitäten zu nutzen wie z.B. Effekte in Kombination mit Triggern (vgl. Abschnitt Abschnitt 4.1.3).

7.1.2 Veröffentlichungszeitpunkt

Der WPF wurden über das WPF-Toolkit viele zusätzliche Funktionen zwischen den eigentlichen Release-Versionen hinzugefügt. Die WPF hat sich also fließend weiterentwickelt, während Flex hauptsächlich zu den Releases-Terminen eine Aufwertung erhält. Zum Zeitpunkt der Arbeit stand das Release einer neuen Flex Version (Flex 4.0) kurz bevor. Dieser Punkt lässt die Flex-Bewertung etwas negativer erscheinen, da die getestet Version kurz davor war durch eine Neue ersetzt zu werden, die sicherlich eine andere Bewertung erzielt hätte. Zum Zeitpunkt des Releases von Flex 3.0 waren z.B. viele Funktionen im WPF-Framework noch gar nicht integriert, wie z.B. das mächtige Datagrid oder der Visual State Manager. Auch zu diesem Zeitpunkt wäre die Bewertung anders ausgefallen.

7.1.3 Relation zwischen WPF und Flex

Die WPF gilt als eines der leistungsstärksten Frameworks (wenn nicht das Leistungsstärkste) zur GUI-Entwicklung. Flex wurde also mit der technischen Spitze verglichen und die herausgearbeiteten Unterschiede liegen eher im Detail. Die Leistungsfähigkeit des Flex-Frameworks befindet sich demnach auf einem ähnlichen Niveau. Ein zusätzlicher

Punkt ist, dass Flex eine Web-Technologie ist und damit einigen technischen Grenzen unterliegt, die die Laufzeitumgebung im Browser mit sich bringt (z.B. Security Sandbox). Durch diese technischen Grenzen sind bestimmte Leistungsmerkmale von vornherein ausgeschlossen, wie etwa die Nutzung von externen Ressourcen wie dem Internetexplorer zur Darstellung von HTML-Seiten.

Zusammenfassend wird Flex daher als leistungsfähige Technologie für die GUI-Entwicklung eingestuft, auch wenn Flex im Vergleich zur WPF durchgängig schlechter bewertet wurde.

7.2 Zusammenfassung des Praxisteils

Im praktischen Teil sollte mit Hilfe des im theoretischen Teil angeeigneten Wissens eine prototypische Anwendung entwickelt werden, mit dem Ziel möglichst viele der untersuchten Funktionen zu nutzen, um einen praktischen Eindruck des Entwicklungsprozesses eines GUI mittels des Flex-Frameworks zu gewinnen. Die Anwendung wurde erfolgreich implementiert und es konnte eine Vielzahl der vorgestellten Konzepte genutzt werden (vgl. Kapitel Kapitel 6). Um einen Vergleich mit den theoretischen Ergebnissen zu ermöglichen wurden vor der Implementierung, Erwartungen aus dem erworbenem Wissen abgeleitet. Diese Erwartungen wurden im Abschnitt Abschnitt 6.3 vorgestellt. An dieser Stelle sollen die Erwartungen an den praktischen Eindrücken überprüft werden.

7.2.1 Auswertung: Steuerelemente und Konzepte

Die Flex-Steuerelemente konnten zum größten Teil über die Beschreibungssprache MXML implementiert werden. Diese Arbeitsweise kann als sehr komfortabel bezeichnet werden. Der entstehende MXML-Code ist sehr übersichtlich und Steuerelemente können einfach konfiguriert werden. Zusätzlich bietet der Flexbuilder über den Designmodus die Möglichkeit den MXML-Code zu visualisieren, wodurch man einen Eindruck von der Darstellung der Anwendung bekommt. Diese Möglichkeit der „Vorschau“ ist während des Entwicklungsprozesses sehr hilfreich. Der Umfang der angebotenen Steuerelemente ist ausreichend groß. Zusätzlich können stellenweise recht komplexe Modifizierungen der Steuerelemente vorgenommen werden, z.B. das Anzeigen einer eigenen Komponente innerhalb des DataGrids. Während der Entwicklung sind allerdings kleinere Mängel aufgefallen, wie etwa die schlechte Positionierung des Tooltips innerhalb des DataGrids

oder Probleme die Spaltenbreite des DataGrids korrekt einzustellen[1]. Das Konzept des Databindings wurde die ganze Anwendung hindurch konsequent genutzt. Dabei hat sich gezeigt, dass dieses Konzept eine merkliche Bereicherung und wesentlicher Teil des Flex-Entwicklungsprozesses ist.

Trotz der kleinen Mängel, wurden die Erwartungen in diesem Bereich voll erfüllt.

7.2.2 Auswertung: Layout

Das Verschachteln verschiedener Container in eine Hierarchie blieb durch die MXML-Struktur immer klar nachvollziehbar und übersichtlich. Der Design-Modus des Flexbuilders unterstützt den Layoutprozess mit der Möglichkeit die Container (auch wenn eigentlich nicht sichtbar) hervorzuheben. Zusätzlich können Constrains in diesem Modus angelegt werden, was einen großen Vorteil darstellt, da man so das Ergebnis gleich betrachten und eventuelle Feineinstellungen vornehmen kann. Die Anordnung über Constrains wurde sehr häufig benutzt und ermöglicht auch bei Skalierung der Container ein genaues Positionieren der Kind-Elemente. Bei den Containern mit automatischem Layout trat in Ausnahmefällen Verwirrung auf, da die Skalierung der Kind-Elemente an manchen Stellen nicht nachvollziehbar geschah. Dies hat eine gewisse Verunsicherung ausgelöst, was zu der bevorzugten Nutzung des Canvas (mit Constrains) geführt hat.

Bei der Entwicklung der prototypischen Anwendung wurde besonderer Wert auf ein dynamisches Layout gelegt. Es sich gezeigt, dass diese komplexe Form des Layouts mit Flex gut realisiert werden kann. Die Erwartungen in diesem Bereich wurden daher, trotz geringer Abstriche bei den automatischen Containern, erfüllt.

7.2.3 Auswertung: Animationen und States

In der Anwendung wurde an mehreren Stellen mit Animationen gearbeitet. Die Zuweisung von Effekten mit Hilfe von Triggern ist sehr einfach und es kann damit eine schnelle Aufwertung des Benutzerinterfaces erreicht werden. Negativ fiel jedoch auf, dass der wichtige Fade-Effekt die Schriften nicht ohne Probleme einbezogen hat (vgl. Abschnitt Abschnitt 6.4.3). Im Rahmen der Entwicklung wurde versucht eigene Effekte zu implementieren um außergewöhnliche Animationen zu erstellen, so sollte beispielsweise eine „Aufrollen“-Animation (eine komplexe Art eine Schaltfläche einzublenden) erstellt werden.

[1] Darauf wurde in der Praxisbeschreibung nicht eingegangen aber es gibt bei dieser Einstellung einen bekannten Fehler

Dabei kam es aber immer wieder zu nicht nachvollziehbaren Reaktionen (Sprünge oder falscher zeitlicher Ablauf), weshalb wurde dieser Effekt später nicht genutzt wurde. Als Alternative dazu, wurde der `FadeWithBlur`-Effekt implementiert. Bei der Kombination von Effekten (Parallel-Klasse) welche gleichzeitig ablaufen fällt auf, dass trotz hochwertiger Hardware schnell die Leistungsgrenzen erreicht werden. Trotz Optimierung kann es so zu Animationssprüngen oder in Einzelfällen sogar zum Ausfall der Animation kommen. An mehreren Stellen wurden die Animationen mit Hilfe des State-Konzepts implementiert. Dabei wurde festgestellt, dass die Arbeit mit States gerade wenn Action-Effects zum Einsatz kommen vergleichsweise hohen Einarbeitungsaufwand erfordert. In Abschnitt Abschnitt 6.4.3 wurde gezeigt das bei raschen Statewechseln[2] schnell eine Leistungsgrenze dieses Konzeptes erreicht wird.

Zusammenfassend bleibt der Eindruck, dass ein großes Potenzial in diesem Bereich existiert, die Entwicklung stellenweise sehr schleppend und zeitaufwendig ist, so dass man gehemmt ist mehr als einfache Formen der Animationen zu implementieren.

7.2.4 Auswertung: Styles und Skinning

Mit der Funktionalität aus diesem Bereich wurde hauptsächlich die Möglichkeit des auswählbaren Themes umgesetzt. Dieser Bereich hatte im Vergleich zur WPF die schlechteste Bewertung. In der Praxis hat sich jedoch gezeigt, dass diese Funktionalitäten des Flex-Frameworks sehr gut zu implementieren sind und ansprechende Ergebnisse damit erzielt werden können. Bereits mit den Stil-Attributen lassen sich umfangreiche visuelle Anpassungen durchführen. Negativ fiel allerdings auf, dass bestimmte Stil-Attribute nicht konsistent bei allen Steuerelementen angeboten werden. In der prototypischen Anwendungen sollte z.B. eine durchgängige Darstellung mit abgerundeten Ecken erfolgen, bestimmte Steuerelemente wie das DataGrid oder auch die Akkordion-Komponente bieten entsprechende Stil-Attribute jedoch gar nicht an. Dies ist sehr unerfreulich da eine einheitliche Darstellung natürlich gewünscht ist. Mit Hilfe der CSS-Dateien und den dazugehörigen Selektoren können Stil-Einstellungen auf verschiedene Weise zugeordnet werden wodurch man eine individuelle Gestaltung für eine ganze Anwendung

[2] beim Wechsel eine States während noch ein anderer Übergang animiert wird. Beim Arbeiten im Bereich der Animation und States entstand der Eindruck, dass eine Vielzahl von Optionen zur Verfügung stehen. Jedoch tauchten immer wieder eine Vielzahl kleiner scheinbar zufallsbedingter Probleme auf, so dass es nie zu einem flüssigen Arbeitsprozess kam und sich die Entwicklungszeit unverhältnismäßig ausdehnte

speichern kann. Besonders positiv fällt dabei die Möglichkeit der Runtime-CSS auf. Das Importieren externe Stil-Einstellungen mitsamt der zugehörigen Grafiken ist damit sehr komfortabel möglich. Ein Umschalten zwischen den unterschiedlichen Darstellungen ist ebenfall einfach zu implementieren.

Insgesamt wurden die Erwartungen in diesem Bereich zum größten Teil erfüllt und durch die komfortable Möglichkeit der Runtime-CSS teilweise sogar übertroffen.

7.3 Abschließende Betrachtung

Die Kernmotivation dieser Diplomarbeit war es herauszufinden, ob die Technologie Adobe Flex für die eigene Firma eine mögliche Lösung zur Realisierung zukünftiger Projekte darstellt. Um dies zu beantworten wurde der Teilaspekt der GUI-Entwicklung untersucht. Die Kernfrage dieser Diplomarbeit war: *"Ist es möglich mit dem Flex-Framework eine leistungsstarke und moderne grafische Benutzeroberfläche zu entwickeln?"* Die theoretische Untersuchung hat gezeigt, dass das Flex-Framework eine Vielzahl an Möglichkeiten bietet ein GUI zu entwickeln. Dabei wurde das Flex-Framework zwar etwas schlechter als das WPF-Framework eingeordnet, wird aber als konkurrenzfähig und damit als gut bewertet. Die praktische Entwicklung hat dieses Ergebnis insgesamt bestätigt. Abschließend kann die gestellte Frage also mit: *„Ja, es ist mit Flex effizient möglich eine leistungsstarke und moderne grafische Benutzeroberfläche zu entwickeln"* beantwortet werden.

7.4 Ausblick

Die gesellschaftliche Bedeutung des Internet wird wahrscheinlich weiter zunehmen. Aus diesem Grund unternehmen martkbestimmende Firmen wie Microsoft und Adobe große Anstrengungen moderne Internettechnologien ständig weiter zu entwickeln. Ziel ist es dabei Einfluss auf die Entwicklung potenzieller, zukünftiger Standarts zu nehmen und damit große Marktanteile und Mitspracherecht für die Zukunft zu sichern. Dieser harte Konkurrenzkampf führt zu immer neuen technischen Ansätzen und somit zu einem sehr schnelllebigen Markt. Adobe veröffentlichte beispielsweise erst kürzlich eine neue Version von Adobe Flex, Flex 4.0. Diese neue Version wird sich in Zukunft mit Technologien wie HMTL 5/ Javascript und Silverlight messen müssen. Dabei setzt Flex 4.0 auf verschiedene Neuerungen welche nachfolgend kurz aufgelistet werden.

1. Einführen von Adobe Catalyst, einem Designer Tool, das den visuellen Entwicklungsprozess der Flex GUI stark unterstützt.
2. Einführen einer neuen Architektur der Steuerelemente (genannt „Spark“) in der Logik, Design und Layout stärker getrennt sind und demzufolge auch unabhängiger variiert werden können.
3. Einführen der neuen MXML Version 2009, welche grafische Werkzeuge wie Catalyst besser unterstützt.
4. Verbesserung des States-Konzeptes bei dem die Syntax verändert wurde um die Nutzbarkeit zu erleichtern.
5. Einführung des Grafikformats FXG, das die Kommunikation zwischen Werkzeugen wie Catalyst, Illustrator und Flex erleichtern soll.
6. Skinning Verbesserung, welche durch die Spark-Komponenten ermöglicht wird. Diese Verbesserung ermöglicht einen besseren Zugriff auf die visuellen Komponententeile unabhängig von ihrer Logik.
7. Angepasstes Layout-Model durch welches viele der alten Flex-Layout-Container entfernt werden und durch ein neueres universelleres Konzept ersetzt wurden.
8. Einführung des Flashbuilders 4.0 (die neue Version des Flexbuilders).
9. Die Geschwindigkeit des Compilers wurde erhöht.
10. Neue Möglichkeiten in der Arbeit mit Text, wodurch die Nutzung der mit dem Flashplayer 10 erschienenen neuen Textengine möglich wird.

Es fällt auf das einige Punkte die in dieser Arbeit negativ aufgefallen sind, in der neuen Version verbessert werden. Zum Beispiel die Komplexität des Skinningprozesses und die Nutzbarkeit des Statekonzeptes. Ein weiterer Brennpunkt ist der derzeit stark umkämpfte und rasant wachsende Markt von Internet Anwendungen für mobile Endgeräte. Grundlegend sind Flex/Flash-Anwendungen für mobile Endgeräte geeignet und effizient einsetzbar. Apple schreibt in seinen Nutzungsbedingungen für das IPhone/ Ipad jedoch die zugrunde liegende Programmiersprache vor (Objective C, C oder C++) und schließt damit Flash/ Flex Anwendungen vom wichtigen Markt der Apple-I-Produkte aus [Tri10].

Ungeachtet von der harten Marktsituation wird, gerade durch die verschiedenen mobilen Endgeräte, der Bedarf an innovativen und leistungsfähigen Benutzerschnittstellen weiter steigen, da diese Geräte auch viele neue Eingabekonzepte mit sich bringen. Ein aktueller Trend ist z.B. beim IPhone/ IPad zu beobachten, welches mit intuitiven Menüstrukturen und einem ausgezeichneten Touchscreen überzeugt. Auch Multitouch-Anwendungen, also Anwendungen welche mit Mehrfachberührungen über den Bildschirm gesteuert werden können, oder 3D Monitore werden in naher Zukunft wahrscheinlich zum Standard gehören. Die Zukunft bietet also weiterhin viele neue Herausforderungen an die Benutzerschnittstellen, der sich die RIA-Technologien stellen müssen.

Literaturverzeichnis

[Ado10] Adobe. Flash player version penetration. http://www.adobe.com/products/player_census/flashplayer/version_penetration.html, 2010. zuletzt abgerufen am 25.05.2010.

[ALMvV07] Chris Andrade, Shawn Livermore, Mike Meyers, and Scott van Vliet. *Professional WPF Programming .NET Development with Windows Presentation Foundation.* Wrox, 2007.

[BEH+09] Joseph Balderson, Peter Ent, Jun Heider, Todd Prekaski, Tom Sugden, Andrew Trice, David Hassoun, and Joe Berkovitzu. *Professional Adobe Flex 3.* Wrox, 2009.

[Ede08] Norbert Eder. Wpf: Routedcommand, routeduicommand, icommand und commands im allgemeinen-eine einführung. http://blog.norberteder.com/index.php?entry=entry080102-192852, 2008. zuletzt abgerufen am 25.05.2010.

[Fac10] Facebook. Press room. http://www.facebook.com/press/info.php?statistics, 2010. zuletzt abgerufen am 25.05.2010.

[Fri07] Dirk Frischalowski. *Windows Presentation Foundation- Grafische Oberflächen entwickeln mit .NET 3.0.* Addision-Wesley, 2007.

[Hub08] Claudius Huber, Thomas. *Windows Presentation Foundation - Das umfassende Handbuch.* Galileo Computing, 2008.

[ITU09] ITU. Information society statistical profiles 2009 - europe v1.01. http://www.itu.int/dms_pub/itu-d/opb/ind/D-IND-RPM.EUR-2009-R1-PDF-E.pdf, 2009. zuletzt abgerufen am 25.05.2010.

[KL08] Chafic Kazoun and Joey Lott. *Programming Flex 3.* O'REILLY, 2008.

[Kun10] Mirko Kunath. *Entwicklung einer Rich Internet Application mit Flex unter Verwendung von Architektur Frameworks*. HTW-Dresden, 2010.

[Mar07] Bernd Marquardt. *Microsoft Windows Presentation Foundation - Crashkurs*. Microsoft Press, 2007.

[Mic07] Microsoft. Hinzufügen von videos zu steuerelementen und 3d oberflächen mit windows presentation foundation (wpf). `http://msdn.microsoft.com/de-de/magazine/cc163455.aspx`, 2007. zuletzt abgerufen am 25.05.2010.

[Mül09] Florian Müller. *Professionelle Rich-Client-Lösungen mit Flex und Java*. Addision-Wesley, 2009.

[MMG08] Miniwatts-Marketing-Group. Internet users in the world growth 1995-2010. `http://www.internetworldstats.com/emarketing.htm`, 2008. zuletzt abgerufen am 25.05.2010.

[Rüt09] Michael Rüttger. *Adobe Flex 3*. mitp, 2009.

[Sch07] Holger Schwichtenberg. *Microsoft .NET 3.0 Crashkurs*. Microsoft Press, 2007.

[Str08] Stephanie Strauß. *Hochinteraktive, grafische Nutzeroberflächen für moderne Webapplikationen (Rich Internet Applications)-Technologievergleich, Einsatz und Implementierung*. HTW-Dresden, 2008.

[Tri10] Andrew Trice. Debatable future for flash cs5 iphone exporter. `http://www.insideria.com/2010/04/debatable-future-for-flash-cs5-iphone-exporter.html`, 2010. zuletzt abgerufen am 25.05.2010.

[Wal09] Petra Waldminghaus. *Adobe Flex 3 - Rich Internet Applications erstellen*. Galileo Computing, 2009.

[Wid08] Simon Widjaja. *Rich Internet Applications mit Adobe Flex 3*. Hanser, 2008.

[Wil08] Jeff Wilcox. Autocompletebox control: The missing guide. `http://www.jeff.wilcox.name/2008/11/autocompletebox-missing-guide/`, 2008. zuletzt abgerufen am 25.05.2010.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit zum Thema

Entwicklung einer grafischen Benutzeroberfläche für „Rich Internet Applications". Untersuchung der Leistungsfähigkeit von Flex im Vergleich zur Desktop-Technologie WPF.

selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt habe.

Dresden, den 25.05.2010

Stephan Lauterbach