DIPLOMARBEIT

V.watch
Videoüberwachung mit Flash

ausgeführt an der

Höheren Abteilung für Informationstechnologie
der Höheren Technischen Lehranstalt Wien 3 Rennweg

im Schuljahr 2006/07

durch

Stefan Csizmazia
Johannes Hartl
Thomas Seif

unter der Anleitung von

Prof. Dipl.-Ing. Stephan Wieninger – Hauptbetreuer

Wien, am 19.04.2007

KURZFASSUNG

V.watch ist eine Software, die viele Vorteile von so genannten Instant Messengern, wie z.B. ICQ, MSN Messenger oder Skype, und herkömmlichen Systemen zur Videoüberwachung vereint. Kernbereiche von V.watch sind also die Videoüberwachung sowie die Kommunikation der Benutzer via Video, Audio und Text. Die großen Vorteile daran sind Plattformunabhängigkeit und hohe Benutzerfreundlichkeit durch kurze Einarbeitungszeit und Wegfall einer Installation. Gewährleistet werden diese durch die hundertprozentige Realisierung in Adobe's Macromedia Flash.

Auf dem Rechner des Benutzers wird ausschließlich ein Adobe Flash Player benötigt. Serverseitig setzt unser Programm auf dem Flash Media Server 2 der Firma Adobe auf.

Zusätzlich zur Kommunikation mittels Audio-, Video- und Textnachrichten in Konferenzen, haben Benutzer von V.watch die Möglichkeit, einander private Nachrichten zu schicken oder Dateien zu versenden. In Konferenzen können beliebig viele Dateien freigegeben und so den anderen Teilnehmern auf einfachste Art und Weise zur Verfügung gestellt werden.

Videodaten sind im Programm in zwei Auflösungen und drei verschiedenen Qualitätsstufen verfügbar. Ausschlaggebend sind die derzeitige Netzauslastung und die Anforderungen der anderen Benutzer. Das Umschalten erfolgt stets vollautomatisch.

Das Programm ist durchgängig benutzerfreundlich und grafisch ansprechend gestaltet – auch eine ausführliche Hilfe ist vorhanden.

Für Administratoren des V.watch-Systems gibt es eine Logging-Funktion mit grafischer Darstellung und Sortierung der Einträge. Außerdem können Aktivierungsschlüssel für neue Benutzer und eine „Message of the Day" jederzeit generiert werden.

ABSTRACT

V.watch is software, combining lots of conveniences of socalled instant messengers like ICQ, MSN Messenger or Skype and video watching systems. The main modules of V.watch are video watching as well as communication between users via video, audio and text. The main benefits are its platform independence and its high usability because of the short period of vocational adjustment and the abolition of setup. Those advantages are ensured by a complete realization in Adobe's Macromedia Flash.

Additionally to the communication using video, audio and text messages in meetings, the users have the possibility to send private messages and files among each other. In meetings, an unlimited number of files can be uploaded and transferred to other participants.

Video streams in the program are available in two resolutions and three different quality levels. The two crucial factors are the network utilization and the other users' requests. The changes happen fully automated.

Besides a detailed help, the highly usable and graphically appealed design are available in the whole program.

There is a log-system for the administrators of the V.watch-system, containing a graphical display of all the log-entries and the possibility to sort them. Additionally, an activation-code for new users and the "message of the day" can be generated as often as needed.

EHRENWÖRTLICHE ERKLÄRUNG

Ich versichere,

- dass ich meinen Anteil an dieser Diplomarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe
- und mich auch sonst keiner unerlaubten Hilfe bzw. Hilfsmittel bedient habe.

Wien, am 19.04.2007

PRÄAMBEL

Die Inhalte dieser Diplomarbeit entsprechen den Qualitätsnormen für "Ingenieurprojekte" gemäß § 29 der Verordnung des Bundesministers für Unterricht und kulturelle Angelegenheiten über die Reife- und Diplomprüfung in den Berufsbildenden Höheren Schulen, BGBl. Nr. 847/1992, in der Fassung der Verordnungen BGBl. Nr. 269/1993, Nr. 467/1996 und BGBl. II Nr. 123/97.

Liste der betreuenden Lehrer:

- Prof. Dipl.-Ing. Stephan Wieninger – Hauptbetreuer
- Prof. Mag. Roman Jerabek – Hauptbetreuer-Stellvertreter
- Prof. Mag. Andreas Fink – Betreuer
- Prof. Dipl.-Ing. Herbert Sasshofer – Betreuer

Liste der Kooperationspartner:

- bm:ukk (vormals bm:bwk): Österreichisches Bundesministerium für Unterricht, Kunst und Kultur
- BRZ Bundesrechenzentrum GmbH
- Logitech Inc.
- Microsoft Corporation
- ELWERA Installationsgesellschaft mbH

INHALTSVERZEICHNIS

VORWORT

> *„V.watch ist eine innovative Lösung zur Überwachung von Räumlichkeiten mit der Möglichkeit zur Interaktion. Eingeloggte User haben die Möglichkeit, Übersichten, Slideshows und Großansichten von mehreren Clients zu sehen. Will man nun mit der anderen Seite Kontakt aufnehmen, startet man entweder eine direkte Kommunikation oder sogar eine Konferenz mit mehreren Teilnehmern. Dieser Teil des Programms wird V.meet genannt und ermöglicht eine Interaktion mittels Video, Audio und Textnachrichten. Auch eine Einbindung von Bewegungs- und Brandmeldern ist möglich, was aus V.watch eine ernst zu nehmende Lösung im Bereich der Überwachung macht.*"

Mit diesen Worten haben wir unser Projekt bzw. unser Produkt vor knapp einem Jahr beschrieben. Auch nun, wo alle Arbeiten abgeschlossen und das Buch fertig sind, sind sie immer noch gültig – abgesehen davon, dass sie bei Weitem nicht mehr den gesamten Funktionsumfang unserer Software wiedergeben. Im Laufe unserer Arbeit haben wir zusätzlich zu den Hauptzielen noch einige Kannziele realisiert, von denen wir glauben, dass sie die Qualität von V.watch deutlich erhöhen. Dazu zählen unter anderem das Versenden und Freigeben von Dateien und das Private-Nachrichten-System.

Obwohl uns diese Arbeit einiges an Zeit und Nerven gekostet hat, sind wir mit dem Endergebnis doch sehr zufrieden. Wir haben alle geplanten Ziele erreicht bzw. Programmteile realisiert. Maßgeblich daran beteiligt waren unsere Betreuer, allen voran Stephan Wieninger, die uns jederzeit nach bestem Wissen und Gewissen unterstützten. Eine große Hilfe waren auch unsere Sponsoren, die wichtige Infrastruktur zur Verfügung gestellt haben.

In dieser Diplomarbeit hat jedes Teammitglied die Hintergründe und Inhalte seiner Arbeit ausgearbeitet und niedergeschrieben. Die Kapitelüberschriften enthalten den Nachnamen des jeweiligen Autors in Kurzform (CSI, HAR, SEI).

Alle personenbezogenen Bezeichnungen dieser Diplomarbeit gelten sowohl in ihrer weiblichen als auch in ihrer männlichen Form.

VERZEICHNIS DER TABELLEN

VERZEICHNIS DER ABBILDUNGEN

1 EINLEITUNG

Wie in der Kurzfassung dieses Buches bereits ausführlicher beschrieben, ist V.watch eine Kombination aus Instant Messenger und Überwachungssystem. Die Software setzt komplett auf Adobe's Macromedia Flash auf.

In den nachfolgenden Kapiteln sind einerseits die theoretischen Grundlagen unserer Arbeit und andererseits praktische Anwendungen mit Code-Beispielen, Grafiken und Tabellen beschrieben. Code ist dabei der Übersichtlichkeit halber immer speziell formatiert:

Beispielcode `_root.onEnterFrame` im Textfluss.

```
// Codestück außerhalb des Textflusses
_root.onEnterFrame = function() {
      // ...
}
```

Zu Beginn werden die Werkzeuge erklärt, mit denen V.watch entwickelt wurde: Adobe Flash 8 und Adobe Flash Media Server 2. Dazu zählen auch das Flash Video Format und Flash Komponenten. Außerdem werden Alternativen zum Flash Media Server 2 vorgestellt und bewertet.

Grundlagen der Client-Server-Kommunikation in unserem Programm werden ebenso behandelt wie Probleme und ihre Lösungen, die uns im Laufe der Zeit besonders beschäftigt und gefordert haben.

Auch dem wichtigen Punkt Design & Usability ist ein eigenes Kapitel gewidmet, in dem Layout und User-Interface-Überlegungen sowie deren Umsetzungen beschrieben werden.

Da eine Software ohne Hardware nutzlos ist, wird in unserer Arbeit auch auf die für V.watch wichtigen Bereiche Audio und Video eingegangen. Es werden die von uns verwendeten Geräte, wie Kameras und Mikrofone, vorgestellt und ihre Funktionsweisen beleuchtet.

Alles in Allem gibt dieses Buch einen guten Überblick über die Arbeit am Projekt bzw. Programm V.watch und die Ergebnisse, die daraus entstanden sind.

2 ADOBE FLASH {CSI}

Adobe Flash ist ein Autorensystem, also ein System, mit dem interaktive und multimediale Inhalte relativ leicht entwickelt und für verschiedene Verwendungszwecke aufbereitet und verteilt werden können. Ein wichtiger Punkt dabei ist, den Autor soweit zu unterstützen, dass die Entwicklung schnell und vielleicht sogar auch ohne viel Hintergrundwissen möglich ist.

2.1 Das Programm

Flash wurde ursprünglich von der Firma Macromedia entwickelt. Nachdem diese vom Multimedia-Riesen Adobe übernommen wurde, heißt das Programm nun Adobe Flash. Das Programm liegt derzeit in der Version 8 vor – die Version 9 gibt es bereits als Beta-Version.

Durch seinen enorme Funktionsumfang und die weite Verbreitung, die laut Adobe bei 98% aller Desktoprechner liegt (siehe *Abbildung 2.1–e Verbreitung des Flash Players ([ADOB2006a])*), ist Flash für mich *DIE* Schnittstelle für verschiedene Medien. Es kann mit Audio, Video, Text, Interaktionselementen und vielem mehr gearbeitet und programmiert werden – dem Benutzer sind fast keine Grenzen gesetzt.

Im Folgenden wird die „Professional"-Version von Adobe Flash 8 näher behandelt, da diese einen größeren Funktionsumfang an Klassen und Komponenten (siehe *Kapitel 2.4, Flash-Komponenten {SEI}*) sowie eine bessere Unterstützung für Flash Video (siehe *Kapitel 2.2, Flash Video {CSI}*) bietet, als die „Basic"-Version.

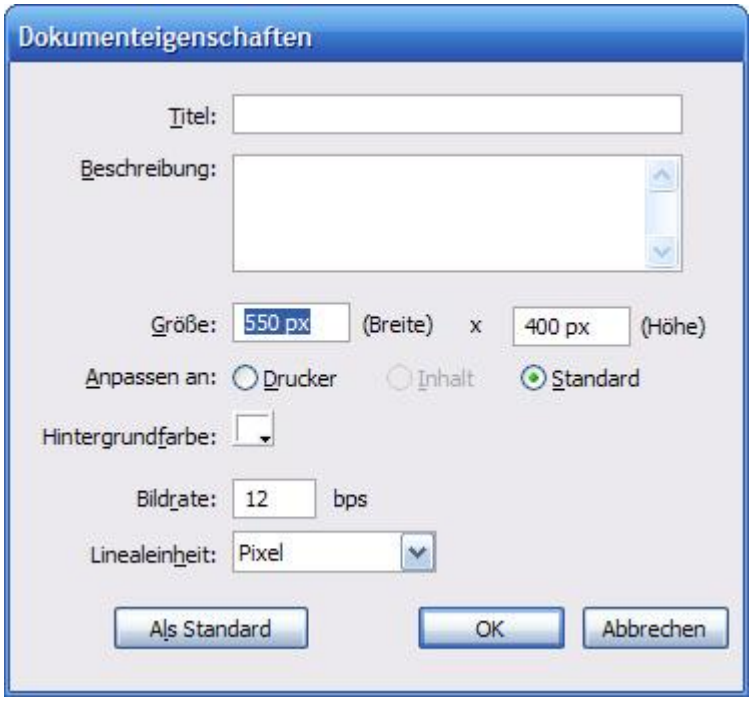
2.1.1 Die Programmoberfläche

Sobald Flash 8 gestartet wird, werden alle Module geladen und ein Startbildschirm angezeigt. In diesem Bildschirm hat der Benutzer die Möglichkeit, Dateien zu öffnen, neue Dateien zu erstellen oder Vorlagen zu öffnen und zu bearbeiten.

2.1.1.1 Dokumenteigenschaften

Jedes Flash-Dokument (.FLA-File) hat einige Eigenschaften, die zu beachten und anzupassen sind. Zum einen wären da die Bühnenabmessungen und deren Hintergrundfarbe, zum anderen die Bildrate. Diese wird in Bildern pro Sekunde angegeben und liegt standardmäßig bei zwölf, da ältere Rechner bei höheren Raten zu sehr ausgelastet würden. Sie ist die Basis für Animationen, die in der Zeitleiste ablaufen.

Abbildung 2.1–a FLA-Dokumenteigenschaften



Um ein fertiges Projekt später publizieren zu können, muss es exportiert werden. Das Format dazu ist SWF („Shockwave Flash"). SWF-Files können nur mit einem Flash Player geöffnet, aber nicht mehr bearbeitet werden. Änderungen müssen daher immer im FLA-File gemacht werden, welches dann erneut exportiert wird.

Um sicher zu stellen, dass auch Benutzer, die keinen Flash Player installiert haben, SWF-Files öffnen können, gibt es die Möglichkeit, so genannte Projektoren zu exportieren. Diese sind nichts anderes als der Flash Player als ausführbares Programm (.EXE unter Windows, .HQX unter Macintosh) mit direkt implementiertem SWF-File. Der Flash Player wird also direkt mitgeliefert.

Alternativ können Flash-Projekte auch als QuickTime-Movies (.MOV) oder HTML-Seiten exportiert werden, in die das SWF-File schon automatisch eingebettet ist.

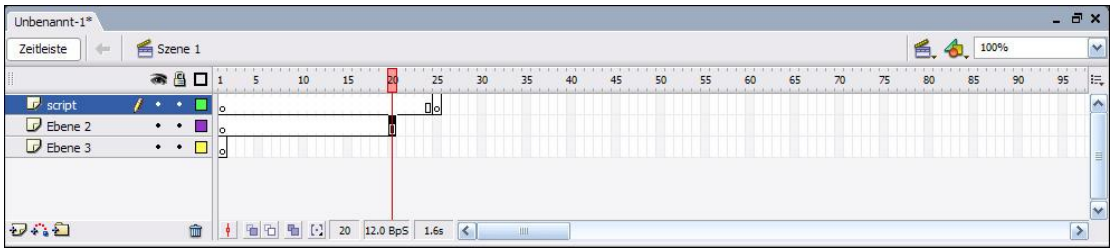
Um Inhalte auch für mobile Endgeräte wie z.B. PDA (Personal Digital Assistent) oder Smartphones anbieten zu können, ist Flash Lite (derzeit in der Version 2.x) notwendig. Dieses ist gratis und als Add-On[↗] zu Flash zu installieren.

Zum Testen von mobilen Anwendungen bietet Flash 8 einen integrierten Emulator für mobile Endgeräte, die Flash Lite unterstützen. Aus einer Dropdown-Liste kann ausgewählt werden, welche Art von Inhalt für welches Gerät exportiert werden soll (z.B. Screensaver, Wallpaper, Stand-alone Player, etc.), um die Kompatibilität zum Gerät zu gewährleisten und das Projekt richtig zu exportieren.

2.1.1.2 Zeitleiste

Die Zeitleiste ist das zentrale Werkzeug zum Verwalten von Ebenen, Bildern und Abläufen – jede Ebene hat dabei ihre eigene Spur.

Abbildung 2.1–b Flash-Zeitleiste



In so genannten Schlüsselbildern werden die Bühnenelemente platziert und gegebenenfalls ActionScript-Code eingebunden. Sobald die SWF-Datei im Flash Player abgespielt wird, werden die Zeitleiste durchlaufen und die darauf angeordneten Bilder der Reihe nach angezeigt. Wie schnell das geschieht, also wie viele Bilder pro Sekunde angezeigt werden, ist in den Dokumenteigenschaften definiert.

Die oberste Instanz ist die Hauptzeitleiste – sie gilt für die ganze Bühne. Jedes Objekt, wie `MovieClips` und Schaltflächen, hat dazu noch eine eigene Zeitleiste, die, bis auf die Anzahl der Bilder pro Sekunde, von der Hauptzeitleiste unabhängig ist.

2.1.1.2.1 Animationen

Da eine sprunghafte Abfolge von Bildern in der Zeitleiste sehr ruckartig abgespielt wird, gibt es die Möglichkeit, richtige Animationen zu erstellen. Dies kann auf zwei Arten geschehen: Bild für Bild oder Tweens.

Bild-für-Bild-Animationen sind sehr aufwendig, da vom Autor jedes Bild einzeln erstellt wird und richtig angeordnet werden muss. Wird das SWF abgespielt, erscheinen die Einzelbilder als animierte Grafik.

Als Erleichterung gibt es die Möglichkeit, so genannten Tweens (Kurzform für „in-between") automatisch berechnen zu lassen. Dabei werden vom Autor nur ein Anfangs- und ein End-Bild erstellt – Flash erledigt den Rest und berechnet alle Zwischenbilder. Bei Tweens ist es unter anderem auch möglich, eine Beschleunigungskurve anzugeben, um die Bewegungen natürlicher wirken zu lassen.

Mittels ActionScript ist es möglich, in der Zeitleiste herum zu springen und den Abspielvorgang zu starten und zu stoppen.

Mit den Befehlen `gotoAndPlay()` und `gotoAndStop()` kann direkt zu bestimmten Bildern, die entweder mit ihrer Nummer oder mit einem Namen angesprochen werden, navigiert werden. Die Befehle `start()` und `stop()` beginnen oder beenden den Abspielvorgang an der jeweiligen Position.

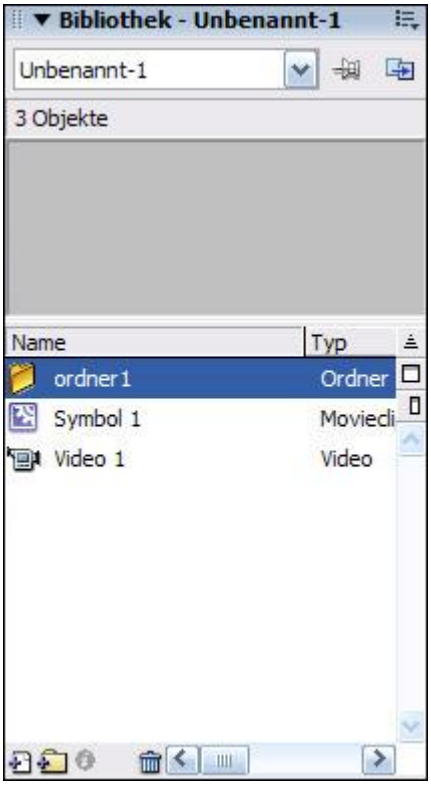
Zu Beginn war es noch Gang und Gebe, dass ActionScript auf mehrere Ebenen und auch in `MovieClips` einzeln verteilt geschrieben wurde. Diese Art der Programmierung war extrem unübersichtlich, weshalb heute meist eine eigene Script-Ebene in der Hauptzeitleiste erstellt wird, die nichts als ActionScript-Code enthält.

In unserem Projekt haben wir mehrere .as-Dateien (ActionScript-Textdateien) erzeugt, die den Programmcode beinhalten und im ersten Schlüsselbild des Hauptfiles inkludiert werden. ([BRUN2005] S. 246f)

2.1.1.3 Bibliothek

Jedes Flash-Projekt hat eine eigene Bibliothek, in der alle in dieser Applikation verwendeten Elemente enthalten sind. Das können Grafiken, Fotos, Sounds, Komponenten, `MovieClips`, Schaltflächen, Video-Objekte etc. sein. Für eine bessere Übersicht gibt es die Möglichkeit, die Elemente in Verzeichnisse zu gliedern.

Abbildung 2.1–c Flash-Bibliothek



Alle Elemente, die in der Bibliothek aufgelistet sind, können in Flash beliebig oft auf die Bühne gezogen und dort verwendet werden. Zusätzlich ist es auch möglich, sie für ActionScript zu exportieren. Bei diesem Vorgang erhält das entsprechende Objekt einen Bezeichner, über den es mit ActionScript angesprochen werden kann. So können Objekte zur Laufzeit dynamisch auf die Bühne geladen und SWF-Dokumente im Bezug auf die Dateigröße entsprechend klein gehalten werden.

2.1.1.4 Werkzeuge

Flash ist ein vektorbasiertes Programm, d.h. dass Formen nachträglich ohne Qualitätsverlust bearbeitet werden können.

Um Grafiken bzw. Formen zu erstellen, gibt es unter anderem das Linien-, das Ellipsen-, das Rechteck-, das Freihand-, das Pinsel- und das Stift-Werkzeug. Damit können, wie aus mehreren Grafikprogrammen gewohnt, die gewünschten Objekte gezeichnet werden.

Abbildung 2.1–d Flash-Werkzeugleiste

Leider bietet Flash in diesem Bereich bei Weitem nicht so viele Features wie z.B. Adobe Illustrator[↗]. Vor allem das Arbeiten mit Verläufen ist im Vergleich zu Adobe Illustrator und Photoshop sehr aufwändig.

Sehr praktisch ist daher, dass Vektorobjekte aus Adobe Illustrator kopiert und in Flash direkt eingefügt werden können. Die Pfade und Ankerpunkte bleiben dabei bestehen.

2.1.1.5 Weitere Bedienfelder

Neben der Bibliothek und der Werkzeugleiste gibt es in Flash noch einige andere Bedienfelder, die frei auf der Programmoberfläche platziert werden können. Dazu zählen unter anderem der Farbmischer und der Komponenten-Inspektor. Alle Bedienfelder sind in Flash unter dem Menüpunkt „Fenster" zu finden.

2.1.2 Programmierung

Um das volle Potential von Flash ausnützen zu können, ist es notwendig, ActionScript zu beherrschen.

ActionScript ist eine objektorientierte Programmiersprache, die auf dem ECMAScript-Standard (ECMA-262) basiert. Derzeit wird ActionScript 2.0 verwendet, die nächste Generation wird ab Flash 9 implementiert sein. ([ADOB2005g] S. 5, [ADOB2005c] S. 102)

Seit Version 1.0 ist ActionScript immer umfangreicher geworden. Heute ist es objektorientiert und unterstützt unter anderem die Verarbeitung von XML-Bäumen[↗] und die Steuerung von Audio- und Videoinhalten. Vor allem in Kombination mit Komponenten (siehe *Kapitel 2.4, Flash-Komponenten {SEI}*) ist ActionScript eine mächtige Schaltzentrale, da damit absolut alles gesteuert werden kann.

Die in Flash integrierte visuelle Programmierumgebung ist ziemlich einfach gehalten, bietet aber Features wie z.B. einen Syntaxcheck, automatische Formatierung und Befehls-Komplettierung sowie Code-Hinweise. Befehle und Funktionen können per Drag'n'Drop aus der Script-Bibliothek geholt werden. Das Generieren von Programmcode ist so auch ohne Syntaxkenntnisse zu bewerkstelligen.

Außerdem ist es möglich, manche Objekteigenschaften direkt per Hand bzw. Cursor zu setzen. Das soll jenen Leuten die Arbeit erleichtern, die beispielsweise nicht auf Komponenten und deren Vorteile verzichten wollen aber keine Erfahrung in deren Programmierung haben.

Diese Arbeitsweise wird auch als RAD (Rapid Application Development) bezeichnet und hilft dabei, z.B. Benutzeroberflächen in sehr kurzer Zeit zu erstellen.

Flash unterstützt außerdem die Auslagerung von ActionScript-Code in externe Dateien. Diese haben die Ende .AS und sind reine Textdateien – sie können also mit jeder beliebigen Programmierumgebung erstellt und bearbeitet werden. Beim Exportieren müssen sich diese Script-Dateien im selben Verzeichnis wie die FLA-Datei befinden. Sobald das SWF-File besteht, werden sie nicht mehr zusätzlich benötigt, da Flash alle Inhalte in eine Datei zusammenführt.

2.1.3 Hilfe/Dokumentation

Eine der wichtigsten Funktionen in Flash ist die integrierte Hilfe. Darin sind Tutorials, Fallbeispiele und Referenzhandbücher enthalten.

Standardmäßig gibt es zehn Bücher – die wichtigsten sind das „ActionScript 2.0–Referenzhanduch" sowie das „Komponenten-Referenzhandbuch". All diese Bücher sind sowohl direkt mit Flash lokal installiert als auch jederzeit in den so genannten „LiveDocs" (zitiert als [ADOB2005d]) online verfügbar.

2.1.4 Stärken/Schwächen

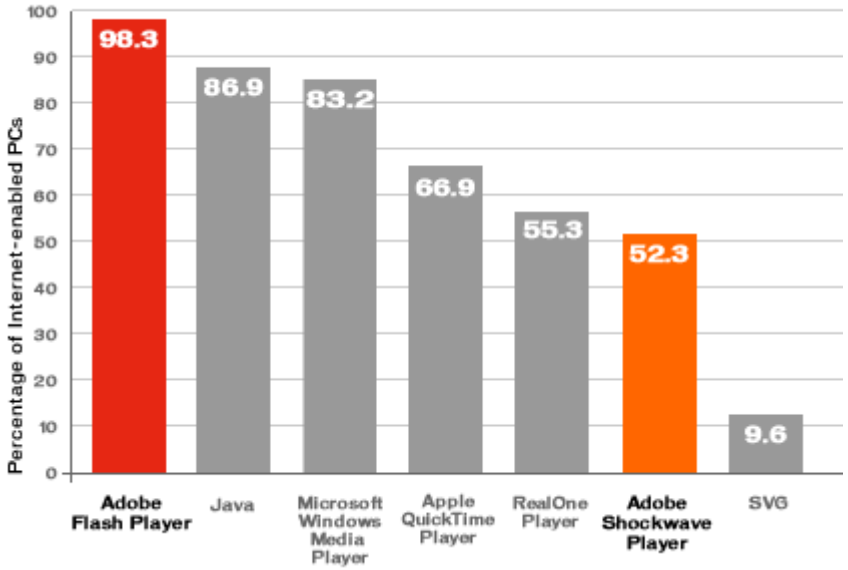
Jede Software hat ihre Vorteile und Nachteile gegenüber anderen Produkten – so auch Flash. Warum wir uns in unserem Projekt genau für Flash entschieden haben, wird in diesem Abschnitt erläutert.

2.1.4.1 Stärken

Flash 8 ist ein sehr umfangreiches Autorensystem, das mit einer sehr weiten Verbreitung aufwarten kann. Heutzutage werden Animationen und Multimedia-Angebote im Internet fast ausschließlich mit Flash erstellt und auch große Portale wie YouTube.com und MySpace.com setzen bei Audio- und Videoinhalten auf Flash. ([ADOB2006a], [ADOB2006b])

Folgende Abbildung zeigt die Verbreitung des Flash Players im Vergleich mit anderen Multimedia Systemen. Die Studie wurde im Dezember 2006 vom Institut „Millward Brown" im Auftrag von Adobe durchgeführt. ([ADOB2006a])

Abbildung 2.1–e Verbreitung des Flash Players ([ADOB2006a])



Die Einfachheit der Oberfläche erlaubt es auch relativ unerfahrenen Benutzern schnell und effizient multimediale und benutzerfreundliche Inhalte für Web- und Standalone-Anwendungen, die in sich abgeschlossen sind und nicht mit anderen Instanzen kommunizieren, zu erstellen. Die Komponenten, anpassbare vorgefertigte Interaktionselemente (siehe *Kapitel 2.4, Flash-Komponenten {SEI}*), tragen dazu sehr viel bei.

Ein weiterer Vorteil ist die mächtige Programmiersprache ActionScript. Damit lassen sich alle Abläufe, Benutzerinteraktionen etc. steuern. Auch komplexere mathematische Operationen wie Matrizenrechnungen sind mittlerweile möglich.

Aufgrund der Übernahme von Macromedia durch die Firma Adobe haben sich im nun gemeinsamen Produktkatalog einige neue Möglichkeiten aufgetan. Ein gutes Beispiel sind die Filter, die es ab der Version 8 in Flash gibt. Diese Filter sind bereits seit einiger Zeit in Adobe Photoshop[↗] verfügbar und können nun auch in Flash verwendet werden – natürlich auch dynamisch mit ActionScript gesteuert.

Mithilfe einer Komprimierung erstellt Flash beim Export extrem kleine Dateien, die problemlos auf den Plattformen Macintosh, Windows und Linux abgespielt werden können. Diese Komprimierung erzeugt im Fall von V.watch SWF-Dateien, die um zwei Drittel kleiner sind als die zugehörigen FLA-Dateien.

Durch die rasante Entwicklung werden zwar sehr oft neue Versionen des Flash Players veröffentlicht, diese sind aber immer abwärtskompatibel.

Die mit insgesamt mehreren tausend Seiten sehr umfangreiche Dokumentation ist gemeinsam mit dem riesigen Entwickler- und Support-Netzwerk ein Garant dafür, dass auf jede Frage eine Antwort und für jedes Problem eine Lösung gefunden wird.

2.1.4.2 Schwächen

Obwohl Flash im Prinzip leicht zu bedienen ist, braucht es schon sehr gute Kenntnisse im Programm selber und in der ActionScript-Programmierung, um alle Möglichkeiten auszunutzen. Diese sind nur mit großem Zeitaufwand zu erlangen.

Ein Nachteil an ActionScript 2.0 ist, dass die Unterstützung von Objektorientierung noch mangelhaft ist. Zwar ist objektorientierte Programmierung damit bereits möglich, elementare Konzepte wie abstrakte Klassen und Mehrfachvererbung werden aber noch nicht unterstützt, wodurch ActionScript derzeit noch eher mit der einfachen Script-Sprache JavaScript, als beispielsweise mit der höheren objektorientierten Programmiersprache JAVA verglichen werden kann.

Dass Flash trotz der hohen Verbreitung immer noch ein proprietäres System ist, sollte man ebenfalls nicht vergessen. Es gibt also keine andere Alternative als bei Adobe einzukaufen, sofern man das Programm kreativ nutzen und Flash-Anwendungen erstellen möchte. Der Flash Player ist hingegen frei verfügbar.

2.2 Flash Video {CSI}

Mit der Einführung von Flash Video hat Macromedia einen großen und wichtigen Schritt getan, um aus Flash ein mächtiges Multimedia-Werkzeug zu machen. Nun sind Entwicklern von derlei Angeboten fast keine Grenzen mehr gesetzt. Dieses Kapitel beschreibt die Funktionsweise sowie Vor- und Nachteile von Flash Video.

2.2.1 Allgemein

Flash Video (FLV) wurde im Jahr 2002 veröffentlicht und wird seit dem Flash Player 6 unterstützt. Seitdem ist die Verbreitung des Flash Players enorm gestiegen.

Bis dahin war es sehr schwierig, Videos für das Internet zu erzeugen und zum Endbenutzer zu bringen. Probleme waren, neben der oft sehr geringen Bandbreite der Internetanbindung, auch fehlende Standards, an denen sich Entwickler orientieren hätten können. ([ADOB2004a])

FLV ist, ähnlich wie AVI ein Containerformat, in dem codierte Audio- und Videodaten zusammen gespeichert sind.

2.2.1.1 Vorteile

Die Vorteile von Flash Video liegen auf der Hand:

- es ist direkt in Flash abspielbar
- hohe Verbreitung des Flash Players
- hohe Komprimierung bei guter Qualität
- Video & Audio in einem File
- sowohl Streaming (siehe *Kapitel 2.2.2.3, Streaming*) als auch progressives Herunterladen (siehe *Kapitel 2.2.2.2, Progressiver Download*) möglich

Neben Adobe Flash kann auch das Web-Entwicklungs-Programm Adobe Dreamweaver Flash Video in Webseiten einbinden.

Beispiele, in denen Flash Video in großem Umfang angewendet wird, sind

- die Community-Plattform MySpace (http://www.myspace.com) sowie
- die Videoportale YouTube (http://www.youtube.com),
- MyVideo (http://www.myvideo.com) und
- Google Video (http://video.google.com)

Konkurrenzformate wie z.B. Real oder QuickTime-Video sind zwar auch vereinzelt in Verwendung, die zum Abspielen benötigten PlugIns sind aber bei weitem nicht so weit verbreitet wie der Flash Player (siehe *Abbildung 2.1–e Verbreitung des Flash Players ([ADOB2006a])*).

2.2.1.2 Nachteile

Da beim Übertragen von Flash Video immer nur die Unterschiede zwischen zwei aufeinander folgende Einzelbilder gesendet werden, sind die Anforderungen an das Rohmaterial, welches codiert wird, ziemlich hoch. Durch diese Komprimierungsmethode wird einerseits extrem viel Bandbreite gespart, andererseits muss man bei Videos, die viele Bewegungen oder Störungen, wie z.B. Rauschen durch Lichtmangel oder schlechten Bildsensor, beinhalten, recht große Abstriche bei der Bildqualität in Kauf nehmen. ([ADOB2004a])

2.2.2 Verwendung von Flash Video

Grundsätzlich gibt es drei Möglichkeiten, Video in Flash einzubinden: in der Zeitleiste eingebettet, als Download oder als Stream[↗].

2.2.2.1 Eingebettetes Flash Video

Das Einbetten in der Zeitleiste empfiehlt sich nur dann, wenn das Video sehr kurz ist und keine zugehörige Audiospur hat. Für alle anderen Einsatzbereiche sollte auf das Einbetten verzichtet werden, da für jeden Frame im Video ein Schlüsselbild in der Zeitleiste (siehe *Kapitel 2.1.1.2, Zeitleiste*) erstellt wird, worunter die Übersichtlichkeit für den Programmierer sehr leidet.

Außerdem ist die Anzahl der Schlüsselbilder in der Zeitleiste auf 16.000 begrenzt, die Videobildrate und die Bildrate der Flash-Zeitleiste (siehe *Kapitel 2.1.1.2, Zeitleiste*) müssen gleich sein und die gesamte Datei muss herunter geladen werden, bevor die Wiedergabe gestartet wird. ([ADOB2004a])

2.2.2.2 Progressiver Download

Progressiver Download von Flash Video bedeutet, dass die FLV-Datei erst zur Laufzeit in das SWF geladen wird. Der Abspielvorgang kann bereits gestartet werden, sobald das erste Segment auf der Festplatte gespeichert wurde. SWF und FLV sind also zwei getrennte Dateien, was das erneute Publizieren nach Änderungen in einer der beiden Dateien leicht macht. ([ADOB2004a])

2.2.2.3 Streaming

Streaming bietet sogar noch mehr Vorteile als das progressive Herunterladen. Hier kann z.B. auf Bandbreitenprobleme oder Userinteraktionen reagiert und das Videoangebot dementsprechend dynamisch geändert werden. Außerdem kann die Wiedergabe noch schneller begonnen werden als beim progressiven Download. Diese Art von Datenaustausch ist notwendig, um Live-Streams zum Server zu senden und diese dort weiter zu verteilen. Auch Video-Konferenzen, wie sie in V.watch realisiert sind, bauen auf dem Live-Video-Streaming auf. ([ADOB2004a])

2.2.2.4 Empfehlungen der Firma Adobe

Folgende Tabelle zeigt Empfehlungen der Firma Adobe, welche Publikations-art von Flash Video in welcher Situation verwendet werden sollte:

Tabelle 2.2-a Empfehlungen zur FLV-Verwendung ([ADOB2004a] S. 1)

Einsatzbereich	eingebettet	progressiv	streaming
Clip ist weniger als 5 Sekunden lang	X	X	
Clip ist zwischen 5 und 30 Sekunden lang		X	X
Clip ist über 30 Sekunden lang			X
Wenig Zuschauer erwartet		X	
Mittlere bis hohe Zuschauerzahl erwartet			X
Sofortiger Start			X
Schutz des geistigen Eigentums			X
Live-Videostreams			X
Variable Streaminggeschwindigkeiten basierend auf der Bandbreite des Besuchers			X
Unterstützt ab Flash Player-Version	6	7	6

2.2.3 Erstellung von Flash Video

Seit der Version 8 von Adobe Flash gibt es zwei Möglichkeiten, FLV-Dateien zu erstellen. Zum Einen bietet Flash direkt die Möglichkeit, Videos mit anderen Formaten als FLV neu zu codieren, andererseits gibt es den Flash 8 Video Encoder als eigenständiges Programm. Dieses bietet zusätzlich zu Codierungseinstellungen für Audio und Video (Bildrate, Bitrate, Bildqualität, etc.) auch noch die Möglichkeit, Cue Points zu setzen. Cue Points sind spezielle Schlüsselbilder in FLV-Dateien, auf die in Flash mittels ActionScript reagiert werden kann. Sie können entweder direkt angespielt werden oder während der Wiedergabe ActionScript-Ereignisse auslösen. Außerdem unterstützt der Flash 8 Video Encoder eine Stapelverarbeitung für die automatische Codierung von mehreren Videodateien.

Für folgende Programme wird von Adobe außerdem ein PlugIn für den FLV-Export angeboten:

- Adobe After Effects (Windows und Macintosh)
- Apple Final Cut Pro (Macintosh)
- Apple QuickTime Pro (Windows und Macintosh)
- Avid Xpress DV (Windows und Macintosh)

2.2.4 Codierung von Flash Video

Bevor ein Video verwendet werden kann, muss es natürlich erst ins gewünschte Format gebracht, also „codiert" werden. Wie die Codierung von Flash Video funktioniert und worauf geachtet werden muss, wird in diesem Abschnitt erklärt.

2.2.4.1 Was ist ein Codec?

Ein Codec (engl. Abkürzung für coder/decoder) ist ein Verfahren bzw. Algorithmus mit dem definiert wird, wie analoge oder digitale Daten codiert und decodiert werden. Meistens erfolgt gleichzeitig eine Kompression bzw. Dekompression, die meist verlustbehaftet oder, im Idealfall, verlustfrei ist.

2.2.4.2 FLV-Codecs

Als der FLV-Standard im Jahre 2002 vorgestellt wurde, gab es nur einen Codec zur Komprimierung von Video-Dateien: Sorenson Spark (H.263) (siehe *Kapitel 2.2.4.4, Sorenson Spark (H.263)*). Erst seit kurzem (ab Flash Player 8) wird ein zweiter Codec, der VP6 von der Firma On2 (siehe *Kapitel 2.2.4.5, On2 VP6*), unterstützt. ([ADOB2004a])

Audiospuren in FLV-Dateien werden immer MP3-codiert. MP3 steht für „MPEG Audio Layer 3" und ist ein standardisiertes Komprimierungsverfahren für Audiodaten, bei dem versucht wird, für den Menschen nicht hörbare Frequenzen heraus zu filtern und so trotz Datenreduktion keine spür- bzw. hörbaren Verluste zu erzeugen. ([BRUN2005] S. 125)

Folgende Tabelle zeigt eine Übersicht der Codecs für FLV. Die Material-Version gibt an, in welcher SWF-Version der Flash-Film exportiert werden muss. In der rechten Spalte ist die Version verzeichnet, die der abspielende Flash Player mindestens haben muss, um die ensprechende SWF-Datei abspielen zu können.

Tabelle 2.2-b FLV-Codec-Unterstützung von SWF und Flash Player ([ADOB2004a] S. 1f)

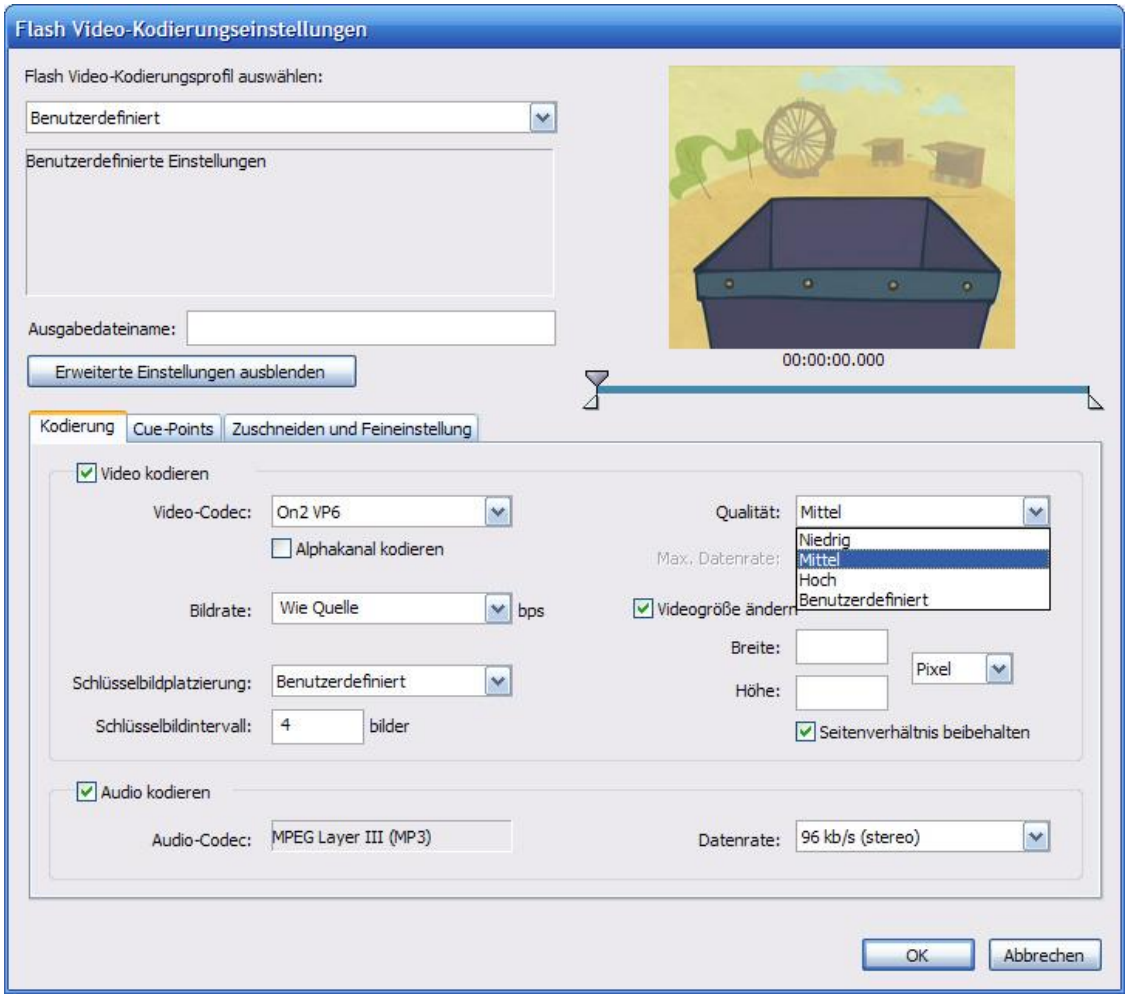
Codec	Material (SWF)-Version	Flash Player-Version
Sorenson Spark	6	6, 7, 8
	7	7, 8
On2 VP6	6	8
	7	8
	8	8

2.2.4.3 Codierungsvorgang

Bei der Erstellung von Flash Video sollte generell darauf geachtet werden, dass das Ausgangsmaterial eine möglichst hohe Qualität hat und im Idealfall vorher noch gar nicht oder nur schwach komprimiert worden ist. Da in FLV-Dateien nur die Pixel-Unterschiede zwischen den einzelnen unkomprimierten Schlüsselbildern gespeichert werden, sollte das Ausgangsmaterial so wenig Bewegung wie möglich beinhalten. Je weniger Pixel sich bis zum nächsten Schlüsselbild verändern, desto weniger Daten fallen an. ([ADOB2004a] S. 1f)

Folgende Grafik zeigt einen Screenshot[↗] vom „Flash 8 Video Encoder", der mit der Professional-Version von Flash 8 gemeinsam ausgeliefert wird. Das Programm dient der Erstellung von Flash Video und wurde im Zuge der Einarbeitung in die Themen unserer Diplomarbeit von uns verwendet.

Abbildung 2.2–a Flash 8 Video Encoder (Screenshot)



Wichtige Parameter bei der Erstellung von Flash Video sind:

- Bitrate:

  Die Bitrate definiert, welche Menge an Daten (Bit) pro Sekunde (bit/s) das Flash Video besitzen soll und sollte je nach Internetanbindung der Endbenutzer ausfallen.

  Die folgende Formel zeigt, dass die Datenrate direkt proportional zur Framerate (Anzahl der Bilder pro Sekunde) und der Auflösung ist:

frame height x frame width x frame rate (fps) = total bits/sec.
((320 x 240 x 30fps)/1000)/2.5 = 922 000 bps

  Die Division durch 2.5 resultiert aus dem Kompressionsverhältnis von 2.5:1, das der Sorenson Spark Codec erreicht.

Eine Reduktion der Framerate um die Hälfte ergibt folglich eine halbierte Datenrate:

((320 x 240 x 15fps)/1000)/2.5 = 460 000 bps

Reduziert man hingegen die Auflösung um die Hälfte und lässt die Framerate bei 30 Bildern pro Sekunde, so wird die Datenrate sogar geviertelt:

((160 x 120 x 30fps)/1000)/2.5 = 230 000 bps

([ADOB2004b] S. 1f)

- Schlüsselbilder:
  Wird Flash Video codiert, gibt es zwei Arten von Daten. Zum Einen sind das Schlüsselbilder, also unkomprimierte Bilder in voller Auflösung. Für die Zeit zwischen den Schlüsselbildern gibt es so genannte Deltabilder. Diese werden nicht wirklich als eigene Bilder übertragen, sondern die Daten stellen immer nur die Veränderung der Pixel zum vorherigen (Schlüssel-)bild dar. Aus diesen Daten errechnet der Decoder bei der Wiedergabe die Deltabilder.

  Beim Codieren von Flash Video ist das Schlüsselbildintervall ein entscheidender Parameter. Es gibt an, wie viele Deltabilder zwischen den einzelnen Schlüsselbildern vorkommen sollen. Je größer dieses Intervall ist, desto schwieriger wird es für den Decoder qualitativ hochwertige Deltabilder zu berechnen – noch schwieriger wird es, wenn viele Pixelveränderungen vorkommen. Es ist also essenziell, das richtige Schlüsselbildintervall zu wählen – zu wenige Schlüsselbilder liefern ein verschwommenes Video, zu viele Schlüsselbilder lassen die Dateigröße schnell anschwellen.

  Außerdem ist zu beachten, dass nur Schlüsselbilder zum Spulen und direkten Ansteuern verwendet werden können.

  Standardmäßig exportiert der Flash 8 Video Encoder mit einem Schlüsselbildintervallwert von 4.

Ein geringes Schlüsselbildintervall wirkt sich nicht direkt auf die Datenrate, sondern auf die Dateigröße des FLV-Files aus.

Mit nachfolgender Formel wird anhand der Anzahl der Schlüsselbilder pro Sekunde und der Framerate berechnet, wie viele Schlüsselbilder im Endeffekt verfügbar sind:

keyframe interval (sec.) x frame rate (fps) = keyframe value
1s x 12fps = 1 Schlüsselbild alle 12 Frames
5s x 30fps = 1 Schlüsselbild alle 150 Frames

([ADOB2004b] S. 1f)

2.2.4.4 Sorenson Spark (H.263)

Aufgrund der Tatsache, dass der Flash Player 8 noch ziemlich jung und daher noch nicht weit verbreitet ist, setzen fast alle großen Anbieter von FLV-Videos im Internet auf diesen Codec, obwohl dieser schon recht alt ist – H.263 wurde 1995/96 von der International Telecommunication Union (ITU) als Standard verabschiedet; er basiert auf dem Standard H.261 von 1990.

Spark ist eine von der Firma Sorenson Media Inc. verbesserte Version des H.263 Codecs. Sorenson Spark ist bei jenem Videomaterial am effektivsten, in dem relativ wenige Veränderungen zwischen den einzelnen Frames vorkommen. Dieser Codec ist ursprünglich für Video-Konferenzen entwickelt worden und ist daher auf niedrige Datenraten optimiert. ([SORE2005])

2.2.4.5 On2 VP6

VP6 ist der Premium-Codec der US-amerikanischen Firma On2. Er wurde dazu entwickelt, in einem möglichst breit gefächerten Anwendungsspektrum eine möglichst gute Qualität zu liefern. So ist es neben der Anwendung auf PC-Architekturen auch möglich, VP6 auf so genannten Embedded Systems und im HD-Bereich („High Definition", Auflösung = 1920x1080 Pixel) einzusetzen. Dabei liefert der Codec in allen Fällen gleichwertige, oft sogar bessere Ergebnisse bezüglich Decodierungs-Geschwindigkeit, Komplexität und Bildqualität als der H.264 bzw. H.263 Standard. ([ON22006])

Des Weiteren ist VP6 auf Echtzeit-Anwendungen spezialisiert, d.h. der Prozessor wird optimal ausgelastet, während aus- und eingehende Streams gleichzeitig codiert bzw. decodiert werden. Dabei muss aber eine kleine Verminderung der Qualität des ausgehenden Streams, also beim Codieren, in Kauf genommen werden. ([ON22006])

Aufgrund dieser vielen Vorteile und da einige wichtige Klassen erst ab Flash 8 verfügbar sind, haben wir uns dafür entschieden, V.watch für Flash Player ab Version 8 zu entwickeln.

2.3 XML in Flash {SEI}

Im folgenden Kapitel wird eine sehr effektive und moderne Methode, Daten in Flash zu importieren, erläutert: Die des XML-Imports. Es werden in den nächsten Abschnitten unter anderem die Vor- und Nachteile und die Funktionsweise dieser Übertragung aufgezeigt und ebenso zwei Einsatzgebiete in unserer Diplomarbeit erklärt. Außerdem wird eine kleine Einführung in das Thema durchgeführt.

2.3.1 Was ist XML?

XML, also „Extensible Markup Language", erlaubt es dem Ersteller eines Dokuments festzulegen, wie dieses auszusehen hat. Bisher war es nur möglich, zum Beispiel bei einer Web-Seite, vorgefertigte Definitionen eines Dokumentes zu verwenden. So gab es bei der Auszeichnungssprache HTML zwei Arten von Tags[↗]. Eine Art formatierte und die andere strukturierte das Dokument. XML erlaubt nun, mit der Erstellung eigener Tags, auch die Definition von Dokumenten zu beinflussen.

> In dem folgenden Beispiel wollen wir aufzeigen, wie eine Software mit einem Dokument umgeht und es interpretiert. Es werden zwei Ausschnitte aus Dokumenten, die mit den beiden Auszeichnunssprachen HTML und XML verfasst wurden, betrachtet.
>
> Mit HTML, also der älteren Variante der Auszeichnungssprachen kann man vor einen Text den Tag `<b>` schreiben, um diesen Text später im Fettdruck darzustellen.

Mit XML ist es nun möglich den verschiedenen Elementen eines Dokuments selbst erstellte Tags zuzuordnen. Folgender Auszug aus einem XML Dokument ist Teil dieses Beispiels.

```
<BUCH>
      <TITEL> Die Blechtrommel </TITEL>

      <AUTOR> Günter Grass </AUTOR>
</BUCH>
```

I

Ein Web-Browser würde nun das Dokument abarbeiten und beim <b>-Tag den Text fett drucken. Würde der Browser auf den XML-Teil des Beispieles treffen, könne er wahrscheinlich nicht sehr viel damit anfangen. Ein anderes Programm, das diese XML-Tags interpretieren kann, könnte aus ihnen lesen, dass es ein Buch mit dem Namen „Die Blechtrommel" von dem Autor „Günter Grass" gibt.

Aufgrund dieses Beispiels kann gezeigt werden, dass XML mit diesen Selbstdefinitionen von Dokumenten einen großen Spielraum für Entwickler bereitstellt.

([DÜNH1998])

Dieses Grundwissen über XML war für unsere Diplomarbeit sehr wichtig. Wir benötigten für einige Teilbereiche von V.watch eine schnelle Datenquelle und konnten aufgrund dieses Wissens auf XML bauen.

2.3.2 Nutzen

Für den Entwickler kann das Verfahren der XML-Einbindung viele Vorteile bringen. Man kann zum Beispiel sehr einfach den Programmcode, also die so genannte Businesslogik, von den Texten und Daten, die Datenlogik genannt werden, trennen. Dadurch sind schnelle Änderungen von Inhalten möglich. Auch in unserer Diplomarbeit fand dieser Ansatz seine Anwendung. Wir nutzen jedoch nicht den direkten Datenimport aus Dateien, sondern schickten vielmehr XML-Objekte von einem Benutzer zum anderen. Diese Übertragung funktioniert über Flash sehr schnell und ohne Probleme.

Uns half außerdem, dass Flash unzählige Methoden und Funktionen, um XML-Dokumente einzubinden, zu verwalten und zu bearbeiten, in sich birgt. Somit konnten wir diese versendeten `XML`-Objekte leicht verändern. Es war uns somit möglich Knoten hinzuzufügen, unterzuordnen und auch zu löschen. Knoten sind eigentlich Tags[↗] in einem `XML`-Objekt, was anhand unseres Beispieles den `BUCH`-Tag als Knoten definiert. Außerdem konnten wir diese Objekte durchsuchen und durch gewisse Kriterien verschiedene Elemente aus ihnen herausfiltern.

2.3.3 Funktionsweise

In unserer Diplomarbeit mussten wir keinen Datenimport von XML auf der Seite des Clients machen. Wir gehen also im folgenden Abschnitt von einem bereits im System vorhandenen `XML`-Objekt aus. Um dennoch den Programmcode für das Laden eines externen XML-Dokuments zu zeigen, referenzieren wir an dieser Stelle auf die Macromedia LiveDocs. ([ADOB2005d])

Ein XML-Dokument kann laut dem Muster seiner Hierarchie in Flash leicht angesprochen werden. Wir verwenden in den folgenden Schritten das Beispiel aus dem ersten Abschnitt dieses Kapitels (2.3.1, *Was ist XML?*). Es gibt ein Buch mit dem Namen „Die Blechtrommel" von dem Autor „Günter Grass". Um den Autor des Buches in Flash ansprechen zu können, müssen einige Schritte zuvor erklärt werden.

Man muss sich in Flash Schritt für Schritt in der XML-Hierarchie nach unten tasten. In unserem `XML`-Objekt mit der Bezeichnung „`xml_object`" ist dieses Buch gespeichert. Man muss zuerst das `root`-Element[↗] des `XML`-Objektes ansprechen. Zwischen den einzelnen Hierarchien steht im Programmcode jeweils ein Punkt. Da das Wurzel-Element auch als ein Unterelement angesehen wird, muss auch dieses ansgesprochen werden. Dies erfolgt mit dem Befehl `xml_object.firstChild`, womit das erste Kind, also Unterelement des XML-Dokuments (hier: `BUCH`), angesprochen wird.

Die jeweils ersten Kinder eines Elements können mit dem Befehl `firstChild` angesprochen werden. Natürlich will man auch die weiteren Unterelemente erreichen können. Dies kann mit dem Befehl `childNodes[x]` durchgeführt werden, wobei die Variable `x` für den Index des Kindes, beginnend bei Null, steht. Somit könnte man den Knoten `AUTOR` wie folgt ansprechen:

```
xml_object.firstChild.childNodes[1]
```

Bei XML-Dokumenten gibt es verschiedene Typen von Knoten. Unter anderem auch den Textknoten. Um diesen, also in unserem Beispiel „Günter Grass", auslesen zu können, müssten wir nach der oben gezeigten Zeile noch ein Unterelement mehr auswählen. Mit der folgenden Zeile kann man also den Text des Knotens `AUTOR` auslesen:

```
xml_object.firstChild.childNodes[1].firstChild
```

Zur Sicherheit kann man nun dem obigen Ausdruck am Ende noch den Befehl `nodeValue` anhängen, jedoch ist dies in Flash nicht unbedingt nötig. Neben `nodeValue` stehen noch die Funktionen `nodeType` und `nodeName`, die den Typ (Textknoten, Attributknoten, …) und den Namen des Knoten (=Node) auslesen, zur Verfügung.

Weitere Befehle und Informationen zu diesem Thema sind den Macromedia LiveDocs zu entnehmen. ([ADOB2005d])

2.3.4 Vorteile/Nachteile von XML in Flash

Im folgenden Abschnitt werden, resultierend aus dem Nutzen, die Vorteile und auch die Nachteile von XML-Daten in Flash aufgelistet.

Aus unserer Erfahrung ergeben sich folgende positive Merkmale der XML-Unterstützung in Flash:

- Die Einbindung erfolgt sehr schnell
- Es gibt gute Funktionen für XML in Flash
- Man kann die Business- und die Datenlogik trennen

- Es ist möglich in Flash XML-Objekte zu verändern
- Man kann die Suche in XML-Objekten, die beim Ansteigen der Größe des Objektes an Geschwindigkeit verliert, mit der Xpath-API[↗] beschleunigen

Einige Funktionalitäten vermissen wir hingegen in der derzeitigen Implementierung:

- Es besteht keine Abhängigkeiten der einzelnen Elemente in einem XML-Dokument
- Die Suche in einem großen XML-Dokument ist ohne das Xpath-API[↗] kompliziert und langsam
- Zur schnelleren Suche muss die Xpath-API erlernt werden
- Um in der Hierarchie weit unten stehende XML-Knoten anzusprechen, können komplizierte, lange und unübersichtliche Ausdrücke im Programmcode entstehen
- Man kann in Flash geänderte XML-Dokumente nicht direkt abspeichern

2.3.5 Einsatzgebiete bei V.watch

Auf der Client-Seite gab es in unserer Diplomarbeit zwei Einsatzgebiete für XML. Diese waren der Textchat in der Konferenz und die „Privaten Nachrichten". Wir hielten es hier für ideal, XML zu verwenden, da die Größe der Elemente, also der Nachrichten, in beiden Fällen eine angenehm kleine Datenmenge aufweist. Außerdem musste die Abwicklung der Prozesse relativ schnell abgearbeitet werden, was mit XML wie schon gehört, möglich ist.

2.3.5.1 Private Nachrichten

In diesem Gebiet hilft uns eine XML-Datei am Server die Nachrichten aller registrierten Benutzer abzuspeichern. Dazu wird folgende XML-Struktur gewählt:

```
<pm>
      <user name="User1">
            <message id=0>
                  <sender>…</sender>
                  <date>…</date>
                  <time>…</time>
                  <priority>…</priority>
                  <subject>…</subject>
                  <text>…</text>
                  <read>…</read>
            </message>
      </user>
</pm>
```

Diese Hierarchie bedeutet, dass jeder Benutzer (`User`), durch Namen getrennt, mehrere Nachrichten (`Message`) gespeichert haben kann. Somit kann man leicht nur die Nachrichten eines bestimmten Benutzers ausgeben. Diese Messages sind wiederum mit einer eindeutigen Identifikationsnummer (`id`) zu unterscheiden. Dies ist wichtig, um einzelne von ihnen einfach löschen zu können. Diese Nachrichten haben eine Reihe von Eigenschaften bzw. Unterelementen, wie zum Beispiel den Sender oder das Sendedatum.

Diese Struktur wird beim Aufruf des Posteingangs eines Benutzers ausgelesen und in eine `DataGrid`-Komponente (siehe *Kapitel 2.4.6.5, Die* `DataGrid-Komponente`) eingelesen. Somit kann der Benutzer alle seine Nachrichten sehen und bearbeiten.

Sollte er mit einer Nachricht interagieren, sprich sie löschen oder auf ungelesen setzen wollen, wird dies auf eine spezielle Weise gehandhabt. Ziehen wir den Löschvorgang etwas näher heran. Er wird zuerst lokal am Rechner durchgeführt, damit direkt nach dem Betätigen der Schaltfläche Gewünschtes passiert. Danach wird die Identifikationsnummer der gelöschten Nachricht an den Server gesendet, wo danach diese Nachricht gelöscht wird. Der Client bekommt die neue Anzahl an Nachrichten zurückgesendet. Stimmt diese mit den aktuell im System befindlichen Dateien überein, war der Löschvorgang erfolgreich. Sollte dies nicht der Fall sein, wird eine entsprechende Meldung ausgegeben. Diese Vorgehensweise verhilft uns einerseits Geschwindigkeit zu gewährleisten und kann außerdem auf etwaige Fehler beim Löschen korrekt reagieren.

2.3.5.2 Textchat

Auch in diesem Bereich unserer Diplomarbeit mussten wir auf hohe Geschwindigkeit setzen. Sobald ein neuer Eintrag in dem aktuellen Chat gemacht wird, wird auf jedem Rechner, der an dieser Konferenz teilnimmt, eine Funktion aufgerufen. Diese Funktion bekommt die neue Nachricht und fügt sie auf jedem Rechner an die letzte Zeile des Chats hinzu.

Bei diesem Bereich von V.watch war uns neben der Geschwindigkeit noch wichtig die Chat-History[↗] am Server mitzuspeichern. Sei es, um Konferenzen mitzuloggen oder es in einer späteren Version von unserer Software zu ermöglichen, diesen Chat-Verlauf für den Benutzer sichtbar zu machen.

2.4 Flash-Komponenten {SEI}

Im folgenden Kapitel wird erklärt, was man unter Flash Komponenten versteht, wozu man sie verwenden kann, welche Vor- und Nachteile sie bieten und wie Komponenten im Hintergrund funktionieren. Es wird weiters eine Aufteilung der Komponenten in Kategorien vorgenommen und auf die wichtigsten von ihnen näher eingegangen. Sämtliche Informationen aus dem folgenden Kapitel sind auch dem „Komponenten verwenden"-Handbuch ([ADOB2003a]) zu entnehmen. Zur Recherche für dieses Thema wurde das „Komponenten verwenden"-Handbuch von Flash MX 2004 verwendet. Sollte man Informationen zu Flash 8 benötigen, sind diese im aktuellen „Komponenten verwenden"-Handbuch und auch im Komponenten-Referenzhandbuch von Flash 8 zu finden. ([ADOB2005a], [ADOB2005b])

2.4.1 Einleitung

Adobe's Macromedia Flash Komponenten sind von Macromedia vorgefertigte Elemente, wie zum Beispiel eine Schaltfläche, die mit einfachen Methoden vom Programmierer leicht angesprochen werden können. Sie dienen zur Vereinfachung der Programmierung und gewährleisten, durch ihre Ähnlichkeiten zum Web, eine gute Usability (Benutzerfreundlichkeit; siehe *Kapitel 7.6, Usability*).

Komponenten sind eigentlich `MovieClips`, die nur durch vordefinierte Parameter verändert werden können. Diese Veränderungen können durch Methoden, die der Programmierer aufruft, bewirkt werden. Außerdem hat jede Komponente Eigenschaften, die man auslesen, und Ereignisse auf die man als Entwickler entsprechend reagieren kann. Der Sammelbegriff API[↗] beschreibt genau diese Eigenschaften, Ereignisse und auch die Methoden einer Klasse. Es gibt verschiedene Arten von Komponenten. Es kann sich bei einer Komponente zum Beispiel um ein einfaches Steuerelement für die Benutzeroberfläche (beispielsweise ein Optionsfeld oder ein Kontrollkästchen) oder um Inhalte (wie eine Bildlaufleiste) handeln. Außerdem kann eine Komponente nicht-visuell sein, wie z. B. der `FocusManager`. Diese Art ist zum Steuern anderer Komponenten zuständig. Mit der `FocusManager`-Komponente, kann man zum Beispiel steuern, welches Objekt in der Anwendung den Fokus, und somit den Bereich, in dem der Benutzer mit der Tastatur etwas bewirken kann, erhält.

2.4.2 Technischer Hintergrund

Aus der technischen Perspektive gesehen, basieren die Komponenten zurzeit auf der Version 2 der Macromedia Komponenten-Architektur. In dieser Version basieren alle Komponenten auf gewissen Klassen, die definieren, welche Möglichkeiten der Programmierer hat, mit den Komponenten zu kommunizieren.

2.4.3 Vererbung

Das Grundgerüst der Version 2 bilden die Klassen `UIObject` und `UIComponent`. Die Eigenschaften dieser beiden Klassen, näher beschrieben in den *Kapiteln 2.4.3.1, UIObject* und *2.4.3.2, UIComponent*, werden an alle Komponenten vererbt, das heißt sie können danach all diese Eigenschaften verwenden. Hierbei ist zu beachten, dass die `UIObject` Klasse noch unter der `UIComponent` Klasse liegt, die also die Eigenschaften von `UIObject` damit übernimmt. Außerdem kann es vorkommen, dass eine Komponente Eigenschaften und Methoden einer anderen Komponente erbt.

Hierzu ein Beispiel:

> Die `List`-Komponente (eine einfache Liste, in der man Elemente auflisten kann) hat die Methode `removeAll()`, die alle Elemente aus einer Liste entfernt. Eine Auflistung aller Methoden einer Komponenten-Klasse ist im „Komponenten-Referenzhandbuch" der Flash Hilfe zu finden (*Abbildung 2.4–b Auszug aus dem „Komponenten-Referenzhandbuch"*).
>
> Da die `DataGrid`-Komponente sich nicht wesentlich von der `List`-Komponente unterscheidet (gleiche Liste, nur mit zusätzlichen Spalten), liegt nahe, dass sie die Methoden und Eigenschaften der `List`-Komponente erbt. Somit befindet sich auch die Methode `removeAll()` im Repertoire der `DataGrid`-Komponente und kann damit auch hier verwendet werden.

Zusätzlich hat die `List`-Komponente wiederum, wie jede andere Komponente, als Basis die beiden Grundklassen, `UIObject` und `UIComponent`. Welchen Vererbungspfad eine Komponente hat, kann man leicht aus dem Referenzhandbuch der Komponenten im jeweiligen Kapitel herausfinden. Hier ein Beispiel:

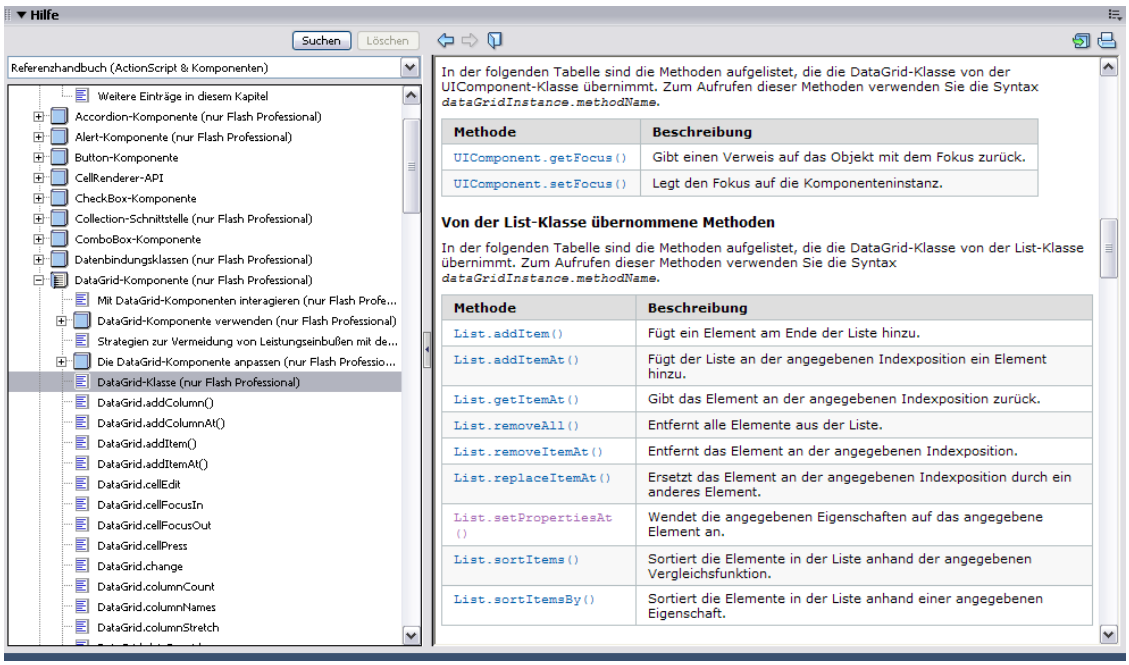
Abbildung 2.4–a Vererbungspfad der DataGrid-Klasse

DataGrid-Klasse (nur Flash Professional)

Vererbung MovieClip > UIObject-Klasse > UIComponent-Klasse > View > ScrollView > ScrollSelectList > List-Komponente > DataGrid

Sollte man seine Informationen öfter aus dem „Komponenten-Referenzhandbuch" entnehmen, sollte man genau auf die Vererbungen achten. In jedem Überbegriff der Navigation (z.B. `DataGrid`-Komponente) findet sich ein Unterpunkt der die komplette Klasse beschreibt (hier: `DataGrid`-Klasse). Öffnet man diesen Unterpunkt, bekommt man alle Eigenschaften dieser Klasse aufgelistet, wobei auch die vererbten Klassen, Methoden und Eigenschaften aufgelistet sind.

Abbildung 2.4–b Auszug aus dem „Komponenten-Referenzhandbuch"



Hier, in diesem kleinen Ausschnitt des Referenzhandbuches der Komponenten, kann man auf der linken Seite (markiert) den wichtigen Unterpunkt sehen. In dem rechten Fenster sind danach die Auflistung der übernommenen Methoden und die dazugehörige Klassen aufgeführt. Der Vererbungspfad und die Hilfeseiten der dezidierten Klasse haben in unserer Diplomarbeit die Recherchearbeit geprägt. Wir konnten somit viel Zeit sparen und fanden immer die nötigen Informationen.

2.4.3.1 UIObject

Die eigentliche Grundklasse für die Komponentenarchitektur der Version 2, die `UIObject` Klasse, hat als Grundlage nur die `MovieClip`-Klasse und unterstützt damit alle Eigenschaften und Methoden, die auch ein `MovieClip` verwendet. Somit ist sie vor allem notwendig, um Objektinstanzen (Komponenten) dynamisch zu erstellen, zu löschen, Ereignisse zu verwalten und ihre Stile und Größe zu ändern. Ein passendes Ereignis zur `UIObject`-Klasse wäre `move`. Es wird ausgelöst, wenn die angegebene Objektinstanz bewegt wird. Außerdem kann mit `setStyle()` die Komponente auf viele verschiedene Weisen grafisch verändert und angepasst werden.

2.4.3.2 UIComponent

Nachdem `UIObject` lediglich zur Erstellung und Veränderung der Komponenteninstanzen zuständig war, kann die `UIComponent`-Klasse das Verhalten der Instanzen verändern und beeinflussen. Genauere Zuständigkeitsbereiche sind laut „Komponenten verwenden"-Handbuch ([ADOB2003a], S.610):

- Fokus und Tastatureingaben empfangen
- Komponenten aktivieren und deaktivieren
- Größen über das Layout ändern

2.4.3.3 SWC-Dateien

Das im folgenden Abschnitt behandelte Thema hatte keinen direkten Einfluss auf unsere Diplomarbeit, jedoch war es für uns wichtig, um Komponenten zu verstehen und somit auf etwaige technische und logische Probleme entsprechend reagieren zu können.

Grundsätzlich sind Komponenten so genannte SWC-Dateien *(Shock Wave Component)*, also Filme, die als Symbole abgespeichert sind. Diese Symbole sind vom Programmierer bei ihrer Verwendung nicht mehr veränderbar und bieten lediglich Parameter an, mit denen die SWC-Dateien angesprochen werden können. Die Version 2 der Komponentenarchitektur wird ab dem Flash Player 6 unterstützt.

Es besteht außerdem noch die Möglichkeit, Komponenten der Version 1 aufzurüsten und für den Flash Player 7, der diese Version eigentlich nicht unterstützt, kompatibel zu machen. Informationen dazu sind in Kapitel 2 des Buches „Komponenten verwenden" zu finden. ([ADOB2003a], S. 30)

2.4.4 Einteilung in Kategorien

Laut des „Komponenten verwenden"-Handbuches können die in Flash MX Professional 2004 beinhalteten Komponenten in fünf Hauptkategorien eingeteilt werden.

Sollten in der folgenden Einteilung Begriffe der Komponenten (z.B. `CheckBoxen`) nicht näher erklärt werden, kann man in *Kapitel 2.4.6, Komponenten in unserer Diplomarbeit* nähere Informationen dazu finden. Alle Begriffe, die direkt erklärt werden, sind nicht im nächsten Abschnitt angeführt.

- Komponenten der Benutzeroberfläche
  Dies sind Steuerelemente, mit denen der Benutzer interagieren kann. Beispiele hierfür wären `Buttons` oder `CheckBoxen`.

- Datenkomponenten
  Diese Art wurde entwickelt, um eine Verbindung zu Datenquellen herzustellen und um sich daraus Informationen zu holen. Eine Datenkomponente wäre die `XML-Connector`-Komponente, die das Senden und Empfangen von XML-Dateien über HTTP[↗] leicht ermöglicht.

- Medienkomponenten
  Mit diesen Komponenten kann man Medien wiedergeben und streamen[↗]. Als Medium könnte zum Beispiel ein Video eingesetzt werden. Die `MediaPlayback`-Komponente wäre ein Repräsentant dieser Komponentenart. Sie ist für den Aufgabenbereich „Streaming"[↗] zuständig und kann nicht zur Wiedergabe von Videos verwendet werden.

- Manager
  Diese Komponenten sind nicht-visuell, das heißt man kann sie nicht sehen. Sie arbeiten im Hintergrund und können beispielsweise den Fokus (also welches Element gerade aktiv ist) oder die Tiefe (welches Element liegt über welchem) von Komponenten verändern und regeln. Für die beiden erwähnten Aufgaben sind der `FocusManager` (Fokus) und der `DepthManager` (Tiefe) zuständig.

- Bildschirme

  Unter dieser Art von Komponenten versteht man lediglich ActionScript-Klassen, mit denen man leicht Formulare und Folien steuern kann. Als Beispiel ist die `Slide`-Klasse gut geeignet. Mit ihrer Hilfe kann man Präsentationsbildschirme zur Laufzeit in Flash verändern.

Eine komplette Auflistung der Klassen und Komponenten zu den jeweiligen Kategorien ist im „Komponenten verwenden"-Handbuch in Kapitel 4 ([ADOB2003a], S. 51ff) zu finden.

2.4.5 Listener

Im folgenden Abschnitt wird erklärt, was man unter einem `Listener` versteht. Außerdem wird seine Verwendung in Verbindung mit Komponenten detailliert beschrieben.

2.4.5.1 Einleitung

Um Komponenten richtig ansprechen zu können, wurden in Flash so genannte `Listener` eingeführt. Sie reagieren auf die vorgefertigten Ereignisse von den verschiedenen Komponenten und bieten dabei dem Entwickler sofort die Möglichkeit auf diese Ereignisse zu reagieren. Sie sind zu vergleichen mit einigen Methoden der `MovieClip`-Klasse, wie zum Beispiel `onPress`, welche ausgeführt wird, wenn auf einen `MovieClip` mit der Maus geklickt wird.

Auf den ersten Blick sind `Listener` ziemlich kompliziert, da sie mehr Programmcode in Anspruch nehmen. Man merkt jedoch schnell, dass es sich lohnt sie zu verwenden, da sie im Endeffekt die Arbeit erleichtern.

2.4.5.2 Funktionsweise

In diesem Abschnitt wird ein `Listener` anhand eines Beispieles erklärt. Es soll etwas passieren, wenn der `Button` mit dem Namen `btn` angeklickt wurde.

Zuerst benötigt man einen `Listener`. Dieser hat den Datentyp `Object`. Die Deklaration einer Variable (also eines „`Objects`“) wird mit `var Variablenname:Variablentyp` durchgeführt.

```
var BspListener:Object = new Object()
```

Danach muss man dem `Listener` zuteilen, bei welchem Ereignis er etwas ausführen soll. In diesem Fall ist das Ereignis `click`, das von der `Button`-Klasse zur Verfügung gestellt wird. `Click` bedeutet, dass der `Listener` anschlagen soll, wenn der Benutzer den `Button` mit der Maus wieder loslässt, nachdem er schon gedrückt hat. In der folgenden Programmzeile wird dem Ereignis `click` eine Funktion zugeteilt. Diese soll ausgeführt werden, sobald das Ereignis eintritt. Das Objekt `evt`, das hier in den runden Klammern steht, dient dazu, um herauszufinden, welches Element gedrückt wurde.

```
BspListener.click = function (evt:Object) {
```

In dieser Funktion ist nun das Objekt `evt` notwendig. Mit `evt.target` wird das gedrückte Element ermittelt. In der folgenden Programmzeile wird abgefragt, ob (engl. `if`) dieses Element der `Button` mit dem Namen `btn` ist. Ist dies der Fall, so wird ein Ereignis ausgelöst.

```
      If(evt.target == btn) {
            //EREIGNIS
      }
}
```

Nach der Definition des `List`eners und seiner Ereignisse muss man den jeweiligen Elementen (hier nur dem `Button btn`) diesen `Listener` zuordnen. Dies geschieht mit `Element.addEventListener`. Diese Methode hat zwei Parameter: Der erste ist das Ereignis, bei welchem der `Listener` anschlagen soll. Dieses Ereignis muss mit `Listener.Ereignis = Funktion` definiert sein. In den zweiten Parameter wird der Name des `List`eners eingetragen.

```
Btn.addEventListener("click", BspListener);
```

Somit können in einem `Listener` viele verschiedene Ereignisse von verschiedenen Elementen definiert werden. Man könnte auch, um einen `Listener` über alle Elemente zu erstellen, `Listener.handleEvent` schreiben und somit jedes Ereignis abfangen. Nach der Definition des `List`eners kann somit jedes Element, mit jedem Ereignis, zugeordnet werden. Hier ein Beispiel mit dem `Listener` von oben:

```
BspListener.handleEvent = function (evt:Object) {
      if(evt.target == btn) {
            //EREIGNIS 1
      }
      if(evt.target == cb) {
            //EREIGNIS 2
      }
}
btn.addEventListener("click", BspListener);
cb.addEventListener("change", BspListener);
```

Das Ereignis `change` wird von der `ComboBox`-Komponente (siehe *Kapitel 2.4.6.6, Die* `ComboBox-Komponente`) bereitgestellt. Es wird ausgelöst, wenn ein anderes Element aus der Liste ausgewählt wird.

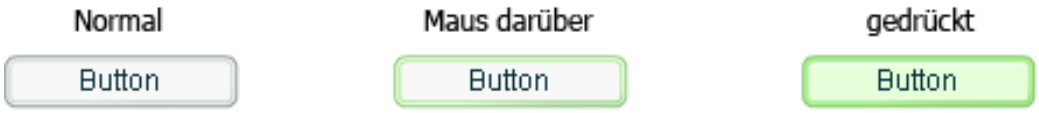
2.4.6 Komponenten in unserer Diplomarbeit

Nachdem die Funktionsweise und Definition von Komponenten erläutert wurden, wird in diesem Abschnitt des Kapitels auf die Komponenten etwas näher eingegangen, die wir in unserer Diplomarbeit verwendet haben. Da es sich in den folgenden Punkten nur um kurze Einführungen zu den einzelnen Komponenten handelt, wird für jede Komponente weiterführende Literatur angegeben und empfohlen. Wir haben in unserer Arbeit ausschließlich Komponenten der Kategorie „Benutzeroberfläche" *(=engl. User Interface – UI)* verwendet.

2.4.6.1 Die Button-Komponente

Diese Komponente ist eine rechteckige Schaltfläche mit abgerundeten Ecken, auf die der Benutzer klicken kann, um ein Ereignis aufzurufen. Ohne grafische Veränderungen sieht diese Komponente wie folgt aus:

Abbildung 2.4–c Aussehen der Button-Komponente



In unserer Diplomarbeit wird der `Button` in vielen verschiedenen Bereichen eingesetzt. Sei es als Login-Schaltfläche oder als Interaktionsmöglichkeiten (mit Icons[↗] darin) in der Übersicht, diese Komponente wird überall verwendet, wo der Benutzer etwas ausführen können soll.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `Button`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 78ff)

2.4.6.2 Die TextInput-Komponente

Die `TextInput`-Komponente ist zu vergleichen mit einem normalen Textfeld in Flash. Dieses normale Textfeld kann in drei verschiedenen Arten auftreten: Als statischer Text, der nicht verändert werden kann, als dynamischer Text, der nur vom Entwickler per Programmcode geändert werden kann und als Eingabetext, den der Benutzer selbst eingeben kann. Die `TextInput`-Komponente ist spezialisiert auf den dritten Typ, den Eingabetext. Man könnte dem Benutzer mit der Eigenschaft `editable` auch die Schreibrechte für die Komponente verbieten und somit einen dynamischen bzw. statischen Text daraus machen, jedoch ist für diese Zwecke die so genannte `Label`-Komponente besser geeignet. In der folgenden Abbildung sieht man den Unterschied der beiden Komponenten.

Abbildung 2.4–d Unterschied zwischen TextInput-Komponente und Label-Komponente

Außerdem ist es möglich mit einer Eigenschaft festzulegen, ob die `TextInput`-Komponente als Passwortfeld dienen soll. Dies äußert sich mit kleinen Sternchen anstelle des Textes und der deaktivierten „kopieren"-Funktion von Windows, das heißt man kann den Text nicht aus dem Feld kopieren.

Abbildung 2.4–e TextInput-Komponente als Passwortfeld

Die `TextInput`-Komponente hat in unserer Diplomarbeit vor allem beim Registrieren und beim Login, zur Eingabe von Benutzerdaten, Anwendung gefunden. Zudem beim Text-Chat in der Konferenz, zur Eingabe der zu sendenden Nachricht, beim Überwachungsmodus, zur Eingabe des Passworts, bei den „Privaten Nachrichten", zur Eingabe des Betreffs einer neuen Nachricht, und beim Öffnen einer neuen Konferenz, zur Eingabe des Titels.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `TextInput`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 569ff)

2.4.6.3 Die TextArea-Komponente

Diese Komponente ist eigentlich ein mehrzeiliges `TextInput`-Feld. Geht der Text über die Höhe der `TextArea`-Komponente hinaus, erscheint automatisch eine vertikale Bildlaufleiste am rechten Rand der Komponente, mit der leicht nach oben und nach unten navigiert werden kann. Diese Leiste kann danach wiederum separat über den Programmcode angesprochen und dynamisch verschoben werden. Sollte der Text über die Länge der Komponente hinausgehen, kann sie sich auf zwei verschiedene Weisen verhalten. Je nachdem wie die Eigenschaft `wordWrap` gesetzt ist, setzt die `TextArea`-Komponente einen automatischen Zeilenumbruch oder fügt eine horizontale Bildlaufleiste ein.

Abbildung 2.4–f Unterschiedliches Aussehen nach Veränderung der Eigenschaft „wordWrap"

Diese Komponente wurde in unserer Diplomarbeit selten verwendet. Anwendungsbereiche lagen im Text-Chat in der Konferenz, zur Anzeige des Chat-Verlaufes, und bei den „Privaten Nachrichten", zur Eingabe einer neuen Nachricht und zum Anzeigen der aktuellen Nachricht im Posteingang.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `TextArea`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 557ff)

2.4.6.4 Die List-Komponente

Diese Komponente dient dazu, Elemente, wie Texte, andere Komponenten oder Grafiken, in einem Listenfeld (mit Bildlaufleiste) anzuzeigen. Dazu verwendet die Liste einen von Null ausgehenden Index, durch den jedes Element unterschieden werden kann. Die Liste besteht grundsätzlich aus zwei Ebenen. Die `data`- und die `label`-Ebene. Beim Einfügen eines neuen Elements kann man beide definieren. Hierzu ein Beispiel:

```
List.addItem({Label:"Benutzername", data:"Thomas"})
```

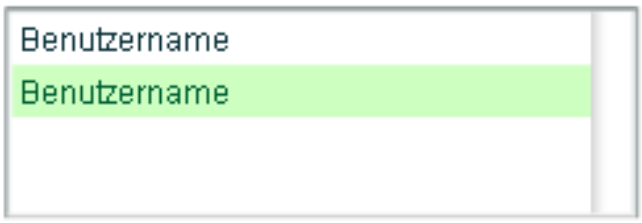
Hier wird als `Label`, also der Text, der in der Liste angezeigt wird, die Bezeichnung `Benutzername` angegeben und als `data`, also der Wert, der im Hintergrund steht, ist der eigentliche Name zu finden. Somit kann man angeben, was in der Liste stehen soll, und aber auch den einzelnen Elementen Werte zuordnen, mit denen man später weiterarbeiten kann.

Es besteht aber auch die Möglichkeit selbst Eigenschaften hinzuzufügen. Dies funktioniert prinzipiell wie das Hinzufügen der `data`-Eigenschaft, nur dass man statt dem Wort `data` einfach den Namen der neuen Eigenschaft schreibt. Das folgende Beispiel funktioniert wie das vorherige, jedoch kommt hierzu noch eine eigene Eigenschaft namens `alter`.

```
List.addItem({Label:”Benutzername”, alter:”19”, data:”Thomas”})
```

Somit kann auf die Eigenschaft `alter` später zugegriffen werden. Der Zugriff auf Eigenschaften ist zum Beispiel in einem `Listener` (siehe *Kapitel 2.4.5, Listener*) möglich. So kann man mit dem ActionScript-Code `evt.target.selectedItem.alter` auf die `alter`-Eigenschaft des aktuellen Elements (`selectedItem`) der Liste (`evt.target`) zugreifen.

Abbildung 2.4–g Aussehen der List-Komponente



Diese Abbildung ist nach den beiden Code-Fragmenten dieses Abschnittes entstanden. Da die `Label`-Schichten der beiden `addItem`-Methoden gleich waren, ist auch die Anzeige in der Liste gleich. Jedoch verbirgt sich hinter dem markierten Element noch die Eigenschaft `alter`, die den Wert `19` aufweist.

Die `List`-Komponente fand in unserer Diplomarbeit nur in einem Bereich Anwendung. Sie diente zur Auflistung aller verfügbaren Benutzer beim Einladen in eine Konferenz. Mithilfe der `CellRenderer`-API (siehe *Kapitel 2.4.8, Das CellRenderer-API {CSI}*) werden auch die Videos der einzelnen Benutzer in der Liste angezeigt.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `List`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 319ff)

2.4.6.5 Die DataGrid-Komponente

Diese Komponente ist eine Unterklasse der `List`-Komponente und hat somit dieselben Voraussetzungen und Eigenschaften wie ihre Vater-Klasse (siehe *Kapitel 2.4.6.4, Die List-Komponente*). Der wesentliche Unterschied zwischen diesen beiden Klassen ist die Tatsache, dass die `DataGrid`-Komponente auch durch Spalten teilbar ist. Auch diese können, genauso wie die einzelnen Zellen, per Programmierung angesprochen werden. Auch die horizontalen und vertikalen Bildlaufleisten übernimmt die `DataGrid`-Komponente von der `List`-Komponente.

Abbildung 2.4–h Aussehen der DataGrid-Komponente

Spalte1	Spalte2	Spalte3
Wert1.1	Wert1.2	Wert1.3
Wert2.1	Wert2.2	Wert2.3
Wert3.1	Wert3.2	Wert3.3
Wert4.1	Wert4.2	Wert4.3
Wert5.1	Wert5.2	Wert5.3
Wert6.1	Wert6.2	Wert6.3

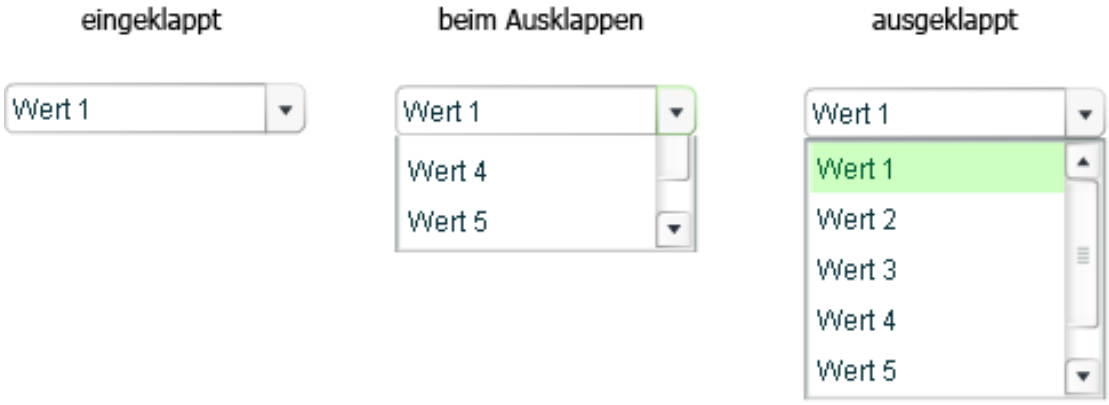
In unserer Diplomarbeit wurde diese Komponente in vielen verschiedenen Teilbereichen verwendet. Zur Auflistung der Konferenzen des markierten Benutzers, der Konferenzteilnehmer, der Nachrichten im Posteingang und der verfügbaren Dateien in einer Konferenz.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `DataGrid`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 171ff)

2.4.6.6 Die ComboBox-Komponente

Mit dieser Komponente kann man dem Benutzer leicht einen Punkt aus vielen auswählen lassen. Die `ComboBox`-Komponente ist eine ausklappbare Liste und ihr stehen damit auch die Eigenschaften und Voraussetzungen der `List`-Komponente zur Verfügung. So sieht die `ComboBox`-Komponente aus:

Abbildung 2.4–i Aussehen der ComboBox-Komponente



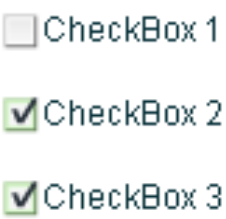
Auch diese Komponente findet in vielen Teilen von V.watch Verwendung. Einige davon sind: Das Navigieren zwischen den Seiten in der Übersicht, das Auswählen des Empfängers einer neuen privaten Nachricht oder das Sortieren der Videos in der Übersicht.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `ComboBox`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 106ff)

2.4.6.7 Die CheckBox-Komponente

Diese Komponente ist vor allem aus dem Web-Bereich bereits bekannt. Bei der `CheckBox`-Komponente handelt es sich um ein kleines Kästchen, das der Benutzer an- oder abhaken kann. Somit kann er Einstellungen oder Abfragen in einem Formular abarbeiten. Ob ein Kästchen selektiert ist oder nicht, kann man per Programmcode abfragen und verändern. Im Gegensatz zur nächsten Komponente, der `RadioButton`-Komponente (siehe *Kapitel 2.4.6.8, Die RadioButton-Komponente*), kann hier eine Mehrfachauswahl von Kästchen stattfinden, wie man auch in der folgenden Abbildung sehen kann.

Abbildung 2.4–j Aussehen der CheckBox-Komponente



CheckBoxen wurden in unserer Diplomarbeit in zwei Bereichen eingesetzt. Zuerst auf dem Login-Bildschirm, um einzustellen, ob die Benutzerdaten gespeichert werden sollen, und im Textchat der Konferenz, um einige Einstellungen zu de- oder aktivieren.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der CheckBox-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 98ff)

2.4.6.8 Die RadioButton-Komponente

Ähnlich der CheckBox-Komponente, ist auch die RadioButton-Komponente aus dem Web sehr bekannt. Sie ist auch dazu da, um es dem Benutzer zu ermöglichen, Abfragen mit eindeutigen Antworten schnell und einfach zu beantworten. Eine solche Frage wäre, welches Geschlecht der Benutzer oder die Benutzerin hat. Trotz ihrer Ähnlichkeit zu ComboBoxen, haben RadioButtons einen markanten Unterschied. Sobald Fragen, die eine eindeutige Antwort erwarten, in einem Formular auftauchen, muss man dem Benutzer auch diese klare Beantwortung möglich machen. Hierzu kann eine so genannte RadioButton-Gruppe verwendet werden. In diesem Fall wären dies zwei Komponenteninstanzen, die in der Eigenschaft groupName den gleichen Wert eingetragen haben. Somit kann immer nur einer dieser Gruppe ausgewählt werden. Diese Prozedur funktioniert natürlich mit beliebig vielen Instanzen je Gruppe. So sieht eine solche Gruppe aus:

Abbildung 2.4–k Aussehen einer RadioButton-Gruppe

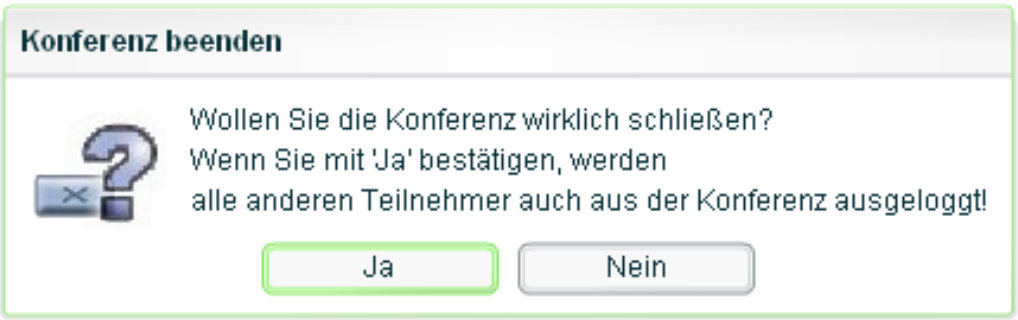
Trotz ihrer vertrauten Funktionsweise hat die `RadioButton`-Komponente nur einmal Platz in unserer Diplomarbeit gefunden. Zum Festlegen der Priorität einer neuen, privaten Nachricht muss der Benutzer eine solche Komponente verwenden.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `RadioButton`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 472ff)

2.4.6.9 Die Alert-Komponente

Diese Komponente ist mit einer Status- oder Fehlermeldung unter verschiedenen Betriebssystemen zu vergleichen. Es handelt sich bei der `Alert`-Komponente um ein Pop-Up-Fenster[↗], das sich automatisch in die Mitte des Bildschirmes platziert und eine Meldung ausgibt. Außerdem kann man diesem Fenster noch Schaltflächen und Icons[↗] zuordnen. Es gibt vier verschiedene Arten von Schaltflächen („Ja", „Nein", „Abbrechen" und „OK"), die beliebig benannt und auch, wie normale `Button`-Komponenten, `Listenern` zugeordnet werden können. Außerdem enthält das `Alert`-Fenster noch eine Titelleiste, in der eine Überschrift platziert werden kann.

Abbildung 2.4–I Aussehen der Alert-Komponente



Die `Alert`-Komponente hat bei V.watch das komplette Anfragenmanagement unterstützt. Wie auch in *Abbildung 2.4–I* zu sehen ist, wird der Benutzer in unserer Diplomarbeit auf viele verschiedene Dinge hingewiesen und kann gleich direkt damit interagieren.

Sei es bei der Eingabe von falschen Benutzerdaten beim Anmelden im System, bei fehlenden Eingaben beim Registrieren oder nachdem man aus einem Meeting geworfen wurde, überall wird diese `Alert`-Komponente verwendet.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `Alert`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 69ff)

2.4.6.10 Die NumericStepper-Komponente

Diese Art der Komponenten ist eine etwas kleine, jedoch hilfreiche, Unterstützung für den Entwickler. Es handelt sich um eine `TextInput`-Komponente, die mit zwei `Button`-Komponenten kombiniert ist. Um sich dies genauer und besser vorstellen zu können, ist eine Abbildung hilfreich.

Abbildung 2.4–m Aussehen der NumericStepper-Komponente

Wie hier zu sehen ist, ist diese Komponente wichtig, um dem Benutzer leicht angeben zu lassen, welche Anzahl er von etwas benötigt. Man kann mithilfe der Hinauf- und Hinunter-Pfeile den Wert in dem Textfeld nach oben oder nach unten korrigieren. Außerdem kann man per Programmierung ein Maximum und ein Minimum für diesen Wert angeben.

Auch der `NumericStepper` hat in V.watch nur eine Funktion übernommen. Im Posteingang für private Nachrichten kann der Benutzer angeben, wie viele Nachrichten er darin angezeigt haben will. Dabei ist die Anzahl aller Nachrichten dieses Benutzers das Maximum der `NumericStepper`-Komponente.

Nähere Informationen zu den einzelnen Eigenschaften, Methoden und Ereignissen der `NumericStepper`-Komponente sind im „Komponenten verwenden"-Handbuch zu finden. ([ADOB2003a], S. 445ff)

2.4.7 Vorteile/Nachteile

Nach den wichtigsten Informationen zu Komponenten, die diesem Kapitel entnommen werden konnten, kann eine Liste an Vor- und Nachteilen erstellt werden. Ob Komponenten in Projekten verwendet werden sollen oder nicht, liegt einzig und allein in der freien Entscheidung bzw. kann an die folgende Auflistung angelegt werden.

2.4.7.1 Vorteile

Benutzerfreundlichkeit:

Aufgrund der Ähnlichkeit zu bekannten Elementen (aus Web und anderen Anwendungen) ist es für den Benutzer einfach mit Programmen zu arbeiten, in denen Komponenten verwendet wurden. Es ist kein zusätzlicher Lernaufwand nötig und Benutzer fühlen sich heimisch. Nähere Informationen dazu finden Sie in *Kapitel 7.6, Usability*.

Handhabung:

Durch die einfache Implementierung, die schnelle Bearbeitung und die Vielzahl an Methoden der Komponenten, ist es dem Programmierer eine große Hilfe, wenn er Komponenten verwendet. Er erspart sich viel Zeit, und auch der Programmcode wird durch vorgefertigte Methoden wesentlich kürzer.

Einsatzmöglichkeiten:

Durch die große Anzahl an bestehenden Komponenten kann man sie beinahe in jedem Projekt verwenden. Es gibt unzählige Vereinfachungen, die sich nicht nur client-seitig etabliert haben. Es gibt auch für den Flash-Media-Server einige Komponenten (siehe *Kapitel 3.11, FMS2-Komponenten*).

Eigene Komponenten:

Sollten die bestehenden Komponenten nicht ausreichen und will man einige Programmteile in einem oder mehreren Projekten immer wieder verwenden, ist es hilfreich selbst Komponenten zu erstellen.

Wie dies funktioniert, und auch nähere Informationen dazu, können im Buch „Komponenten verwenden" ([ADOB2003a], S. 703ff) nachgelesen werden.

Anpassungsfähigkeit:

Obwohl die Komponenten vorgefertigte Elemente sind, kann man sie genau den Vorstellungen entsprechend anpassen. Somit ist eine nahtlose Einarbeitung in das eigene Projekt (z.B. Web-Seite) gewährleistet. Durch Veränderungen von Stilen (Farben, Schriftarten etc.) und auch Skins (Erscheinungsbilder) kann dies durchgeführt werden.

2.4.7.2 Nachteile

Komponenten in hinzu geladenen MovieClips:

Sollte man einen `MovieClip`, in dem sich Komponenten befinden, mit der Funktion `attachMovie()` nach dem Start des Programms hinzu laden, können diese Komponenten nicht einfach angesprochen und verändert werden. Es gibt verschiedene Umwege, wie man diese danach trotzdem verwenden kann (siehe *Kapitel 8.5.1, Komponenten in hinzu geladenen MovieClips*).

Dateigröße:

Verwendet man nun Komponenten in einer Flash-Datei, so wird diese deutlich größer. Auch wenn nach dem Exportieren, also dem Erstellen einer SWF-Datei, diese Größe wieder rapide sinkt, ist die Größe der Flash-Datei ziemlich störend.

Hier ein Beispiel:

- Eine leere FLA-Datei: 32 KB
- Die gleiche FLA-Datei mit einer `Button`-Komponente darin: 192 KB

Verwendet man nun immer mehr Komponenten, so erhöht sich auch die Dateigröße kontinuierlich.

2.4.8 Das CellRenderer-API {CSI}

Flash-Komponenten sind an sich schon sehr mächtig – `CellRenderer` bietet dazu noch ein paar Möglichkeiten mehr zur Anpassung und Personalisierung. Dabei handelt es sich um ein API, das beschreibt, wie in listenbasierten Komponenten – dazu gehören `List`, `DataGrid`, `Tree`, `Menu` und `ComboBox` – beliebige Elemente platziert werden können. Möglich sind beispielsweise Textfelder, `MovieClips` und andere Komponenten. Da man ja alle möglichen Elemente in `MovieClips` einbinden kann, steht einem mit dieser API alles offen.

Alle Informationen zum `CellRenderer` API sind online in den Adobe LiveDocs [ADOB2005d] verfügbar und ebendort entnommen.

2.4.8.1 Was ist ein API?

API ist eine englische Abkürzung und steht für „Application Programming Interface". Es definiert also Schnittstellen, um an die eigenen Programme anknüpfen zu können und vorhandene Funktionen zu ergänzen.

2.4.8.2 CellRenderer-API verwenden

Um an die Schnittstellen dieses API anknüpfen zu können, muss eine Klasse geschrieben werden, die die Klasse `mx.core.UIComponent` erweitert. Wichtig dabei ist, dass die Klasse in einer eigenen Datei mit der Endung .as (ActionScript 2.0 Klasse) abgespeichert werden muss.

Die benötigten Klassen müssen natürlich zu allererst importiert werden:

```
import mx.controls.DataGrid;
import mx.core.UIComponent;
```

Diese neue Klasse kann beliebig viele Attribute haben, die folgenden zwei sollten aber übernommen werden:

```
var owner:MovieClip;
private var ListOwner:DataGrid;
```

`owner` ist eine Referenz auf die Zeile, die jene Zelle enthält, in der sich der `CellRenderer` befindet – `ListOwner` die listenbasierte Komponente in der sich alles abspielt. Der Variablentyp variiert natürlich je nach verwendeter Komponente.

Neben diesen Eigenschaften gibt es einige Methoden, die verwendet werden können und sollten. Besonders wichtig sind diese drei:

Tabelle 2.4-a Wichtige Methoden des CellRenderer API ([ADOB2005d])

Methode	Beschreibung
CellRenderer.setSize()	Legt die Breite und Höhe einer Zelle fest (es kann auch `size()` verwendet werden).
CellRenderer.setValue()	Legt den in der Zelle anzuzeigenden Inhalt fest.
CellRenderer.createChildren()	Erzeugt die gewünschten Objekte. Wird aufgerufen, sobald eine Zelle gezeichnet wird.

Diese Methoden werden immer dann aufgerufen, wenn sich in der Zelle der listenbasierten Komponente etwas ändert, welcher der `CellRenderer` zugewiesen wurde. Solche Änderungen sind z.B. wenn der Cursor über eine Zeile/Zelle bewegt, eine Zeile/Zelle markiert, die Komponente neu sortiert oder wenn eine Zelle bearbeitet wird. Durch all diese Aktionen wird die Komponente auf der Bühne neu gezeichnet – und damit wird auch der `CellRenderer` erneut angewandt.

Des Weiteren gibt es noch zwei Methoden, mit denen man dynamisch innerhalb der oben genannten Methoden gewisse Werte auslesen und gegebenenfalls darauf reagieren kann:

Tabelle 2.4-b Weitere Methoden des CellRenderer API ([ADOB2005d])

Methode	Beschreibung
CellRenderer.getCellIndex()	Gibt ein Objekt mit zwei Feldern zurück, `columnIndex` und `itemIndex`, die die Position der Zelle angeben.
CellRenderer.getDataLabel()	Gibt einen String zurück, der den Namen des CellRenderer-Datenfelds enthält.

Eingebunden wird ein `CellRenderer` mit folgenden Schritten:

1. leeren `MovieClip` in der Bibliothek erstellen
2. Verknüpfungseigenschaften dieses `MovieClips` öffnen
3. im Feld „AS 2.0-Klasse" den Namen der `CellRenderer`-Klasse eintragen (die .as-Datei wird so automatisch inkludiert)
4. im Feld „Bezeichner" einen gewünschten Exportnamen eintragen

Abschließend muss der vorher erzeugte `MovieClip` nur noch der `CellRenderer`-Eigenschaft der Komponente bzw. einer einzelnen Spalte zugewiesen werden:

```
_root.Dgmy_DataGrid.getColumnAt(2).CellRenderer = "movieClipName";
```

2.4.8.3 Anwendung in V.watch

In unserem Programm benötigen wir das `CellRenderer`-API an vier Stellen. In der Teilnehmerliste in Konferenzen soll in jeder Zeile neben dem Namen des Benutzers und dessen Rolle ein Icon zum Stummschalten angezeigt werden. Für Moderatoren gibt es daneben noch ein Icon zum Hinauswerfen des jeweiligen Teilnehmers. Außerdem soll, wenn ein Moderator neue Benutzer einladen will, neben jedem Benutzernamen ein kleines Videofenster mit dem passenden Stream[↗] angezeigt werden. In der Liste der in einer Konferenz frei gegebenen Dateien befinden sich neben jedem Eintrag zwei Icons zum Runterladen und Löschen der jeweiligen Datei.

Für diese Zwecke haben wir zwei Klassen geschrieben:

- `imageCellRenderer` für Icons und
- `videoCellRenderer` für Videofenster

2.4.8.4 imageCellRenderer

Dieser `CellRenderer` wird beiden Spalten im Teilnehmerlisten-`DataGrid` und im Dateilisten-`DataGrid` zugewiesen, in denen Icons angezeigt werden sollen.

In der Methode `createChildren()` werden vier `MovieClips` aus der Bibliothek dynamisch eingebunden, die jeweils eines der vier Icons[↗] beinhalten. Ausgerichtet werden alle Clips in der Methode `size()`.

Schlussendlich wird in der Methode `setValue()` überprüft, welches `DataGrid` und welcher Spaltenindex aktuell ist – je nachdem wird dann ein `MovieClip` aus-, die anderen eingeblendet.

Abbildung 2.4–n zeigt die Teilnehmerliste aus der Sicht eines Moderators. Es werden Icons zum Stummschalten und Kicken von anderen Teilnehmern angezeigt.

Abbildung 2.4–n Teilnehmerliste mit Icons in einer Konferenz

2.4.8.5 videoCellRenderer

Wie bereits erwähnt, wird dieser `CellRenderer` beim Einladen von neuen Teilnehmern in ein Meeting verwendet. Er soll ein Videofeld und daneben ein Textfeld mit dem Benutzernamen anzeigen.

Im Konstruktor der Klasse wird zu allererst das Attribut `videoStreams` als neues Array deklariert – darin werden die benötigten `NetStream`-Instanzen gespeichert.

In `createChildren()` wird ein `MovieClip` aus der Bibliothek eingebunden, der ein Videofeld und ein Textfeld enthält. Dieser wird in der Methode `size()` genau in der Mitte der Zelle platziert.

Ob ein Benutzer einen Videostream zur Verfügung stellt, wird in der Methode `setValue()` überprüft. Ist das der Fall, wird das Video gestreamt und im Videofeld dargestellt. Daneben wird der Name des Benutzers aus einem globalen Array ins Textfeld geschrieben. Ist kein Stream verfügbar, wird stattdessen ein Standard-Ersatzbild angezeigt.

3 ADOBE FLASH MEDIA SERVER 2{HAR}

Der Flash Media Server 2, kurz FMS2, ist eine Kombination von herkömmlichen Streaming-Media-Funktionen und einer besonders flexiblen Entwicklungsumgebung, die es ermöglichen innovative interaktive Medienanwendungen für eine sehr große Zielgruppe zu entwickeln und bereitzustellen. Mit dieser Kombination können eine Vielzahl von Online-Erlebnissen geschaffen werden, einschließlich herkömmlicher Medien-anwendungen, wie z.B. Video on Demand, Live-Web-Übertragungen und MP3-Streaming, sowie Rich-Media-Kommunikationsanwendungen, wie z.B. Video-Blogging, Video-Messaging und Multimedia-Chat-Umgebungen. Und auf diese Weise ist der FMS2 für nahezu jede Zielgruppe verwendbar.

Der FMS2 erlaubt Kommunikationen nach den Prinzipien 1-zu-1, 1-zu-viele, viele-zu-1 und viele-zu-viele und das Ganze in Echtzeit mit Applikationen, die mit dem Autorensystem Macromedia Flash 8 erstellt wurden. Die Applikationen werden in ActionScript programmiert, einer Skript-Sprache die auf demselben Standard wie JavaScript basiert, nämlich ECMA-262. Der Server kommuniziert mit den Macromedia Flash-Playern über das ADOBE-proprietäre Real-Time Messaging Protocol (RTMP), ein unverschlüsseltes TCP/IP-Protokoll, das für Hochgeschwindigkeitsübertragungen von Audio, Video und Daten entwickelt wurde. Für die Administration des Servers, der Applikationen, der Streams, der `SharedObjects` und der Clients stellt FMS2 dem Administrator eine Management-Konsole zur Verfügung, mit der diese Aufgaben sehr einfach zu erledigen sind. ([ADOB2006c],[ADOB2005d])

3.1 Stärken

Eine der Stärken des FMS2 ist die Möglichkeit des Streamings von Live-Videos über den Server und deren Aufzeichnung. Vom Computer unterstützte Kamera- und Mikrofonquellen können zum Server hochgeladen und dort in Echtzeit auf der Festplatte des Servers gespeichert werden. Diese Streams können von den Benutzern natürlich auch in Echtzeit, also live, auf die Client-Rechner geladen und wiedergegeben werden.

Des Weiteren unterstützt der Flash Media Server 2 die Kommunikation zwischen mehreren Benutzern, das bedeutet, dass er vor allem Möglichkeiten für die Realisierung verschiedenster Videokommunikationsanwendungen bietet.

Außerdem wird die maximale Übertragungsgeschwindigkeit automatisch erkannt und der Server stellt das Video daraufhin in der geeigneten Bitrate bereit. Dies wird auch Bandbreitenkontrolle genannt.

Eine weitere Stärke ist die automatische Player-Erkennung, denn der Server erkennt die vom Benutzer verwendete Flash-Player-Version und kann somit die Medieninhalte im richtigen Format bereitstellen.

Und auch aus Sicht der Ressourcen-Auslastung bietet der Flash Media Server eine Möglichkeit zur Optimierung. Denn es ist möglich Origin-Server und Edge-Server (siehe *Kapitel 3.8, EDGE- und ORIGIN-Server*) zu verwenden. Durch die Benutzung mehrerer Edge-Server kann ein automatischer Lastausgleich für große Medienanwendungen mit vielen Client-Rechnern geschaffen werden.

([ADOB2006c])

3.2 Schwächen

Eine der Schwächen sind die Systemvoraussetzungen des FMS2. Die Betriebssystemanforderungen der Server-Software sind einerseits Microsoft Windows 2000 Server oder Windows 2003 Server, und andererseits Linux Red Hat Enterprise Version 3.0 und Linux Red Hat Enterprise Version 4.0. FMS2 lässt sich auf keinen weiteren Systemen installieren.

Außerdem sind die Anschaffungskosten relativ hoch. Die für Linux knappe 11 MB oder für Windows rund 7 MB große Installationsdatei inklusive einer Benutzerlizenz des FMS2 kostet € 5.710,80.

Auch die veraltete Script-Sprache erweist sich als negative Eigenschaft. Die Programmierung am FMS2 unterstützt nämlich ausschließlich ActionScript-Syntax der Version 1.0, hat jedoch einen erweiterten Funktions- und Klassen-Pool, der einige nützliche Möglichkeiten bietet.

Auch aus der Sicht des Benutzer-Handlings gibt es Schwächen, so vermisst man als Programmierer beispielsweise eine einfache Möglichkeit Clients in Gruppen zu verteilen. Würde der FMS2 dies unterstützen, dann würden die FMS2-Applikations-Entwickler sich um einiges leichter tun, wenn es zum Beispiel um das Handling eines Meetings oder Ähnliches geht.

([ADOB2006c])

3.3 Einsatzmöglichkeiten des FMS2

Eine Einsatzmöglichkeit des FMS2 ist Video-on-Demand (siehe *Kapitel 3.4.1.2, Video on Demand*), das bedeutet die Freigabe von Streams für Flash mit erweiterten Videobereitstellungsfunktionen wie Instant On, also sofortiger Verfügbarkeit, weitreichende Interaktivität und Unterstützung von Wiedergabelisten.

Videoaufzeichnungen auf dem Server ermöglichen Rich-Media-Kommunikationsanwendungen wie Video-Blogging und Video-Messaging.

Durch Live-Audio-/-Videoaufnahmen mit dem Flash Player können Videos über eine benutzerdefinierte Flash-Benutzeroberfläche live übertragen werden.

Durch Audio-, Video- und Textinhalte für zahlreiche Anwender mit minimaler Wartezeit werden Multimedia-Kommunikationsanwendungen wie Videozusammenarbeit in nahezu Echtzeit ermöglicht.

([ADOB2006c])

3.4 Abgrenzung zu anderen Produkten

Doch der Flash Media Server ist nicht die einzige Flash- und Streaming-unterstützende Server-Lösung am Markt. In diesem Kapitel werden andere Server-Produkte, die diese Forderungen erfüllen, aufgezählt und deren Stärken sowie Schwächen und bevorzugte Anwendungsbereiche angeführt.

3.4.1 Begrifflichkeiten und Definitionen

In den Produktvorstellungen werden zahlreiche Fremdwörter verwendet, die besonders für den Laien unverständlich sind. Um hier Klarheiten zu schaffen, werden die wichtigsten dieser Ausdrücke im Nachfolgenden erklärt.

3.4.1.1 Streaming Media

Streaming Media ist ein Sammelbegriff für Streaming Audio und Streaming Video. Es bezeichnet aus einem Computernetzwerk empfangene und gleichzeitig wiedergegebene Audio- und Videodaten. Den Vorgang der Übertragung selbst nennt man Streaming. Gestreamte Daten werden als Livestreams bezeichnet, wenn sie heruntergeladen werden, während laufend noch neue Daten desselben Datenstroms hochgeladen werden. Streaming Media bildet damit das Internet-Äquivalent zu Broadcasting-Technologien wie Hörfunk oder Fernsehen. Programmformate sind beispielsweise Internetradio und Video on Demand. ([INTE2006], [KUBL2002], [TECH2005a])

3.4.1.2 Video on Demand

Der Begriff „Video on Demand" bedeutet auf Deutsch „Videoabruf" und ist ein Service, der es Teilnehmern ermöglicht, zu jeder beliebigen Zeit im Internet angebotene Filme abzurufen und abzuspielen. Auf der Anbieterseite benötigt es einen Streaming-Server, auf dem die Videos gespeichert und für das Streaming bereitgestellt werden, auf der Client-Seite lediglich einen beliebigen Player. Ein weiteres Merkmal ist die Buffer-Zeit, also die Zeit, die vergeht, bis der heruntergeladene Datenstrom wiedergegeben wird. Je kleiner dieser Buffer gehalten wird, desto früher beginnt der Player die Daten wiederzugeben.

Ist der Buffer bei zu langsamem Internet zu klein, so kann es passieren, dass die Wiedergabe auf einmal stoppt und gewartet werden muss bis der Buffer wieder gefüllt ist.

Das Gleiche gilt für „Audio on Demand", also zum Beispiel MP3-Streaming.

([FOCU2006], [VIDE2006])

3.4.1.3 Live-Streaming

Live-Webübertragung ist das Gegenstück zu Video on Demand. Die Video- oder Audio-Daten sind hier nicht bereits lange vor Abruf auf dem Streaming-Server gespeichert, sondern werden kontinuierlich empfangen und nur zwischengespeichert. Die Teilnehmer streamen[↗] den zwischengespeicherten Datenstrom mit dem Player automatisch und kontinuierlich auf ihren Rechner und geben diesen wieder. Da nur eine bestimmte Länge des Datenmaterials am Server zwischengespeichert wird, hält sich auch die sogenannte Latenz, also die Verzögerung, in Grenzen. Beim Live-Streaming wird der Buffer so klein wie möglich gehalten, was natürlich eine gewisse Internetbandbreite erfordert. Bei eher langsamer Internetverbindung ist also von Live-Streaming-Anwendungen abzuraten. ([WEBR2006], [LRZ2007])

3.4.1.4 Rich-Media

Unter Rich-Media versteht man die Anreicherung von Online-Inhalten durch Audio und Video. Vorreiterrolle auf diesem Gebiet hat Adobe Flash. ([ATMI2007])

3.4.1.5 Video-Blog

Ein Video-Blog, das Kunstwort dafür Vlog, ist eine Website, die periodisch neue Einträge als Video enthält. Video-Blogs sind überwiegend eigene Werke der Personen, die die Vlogs online gestellt haben. Das ist eigentlich der Punkt, der Vlogs von einer Sammlung von Web-Videos (siehe *Kapitel 3.4.1.2, Video on Demand*) unterscheidet. ([WHAT2006])

3.4.1.6 Video-Messaging

Unter Video-Messaging versteht man das Hinterlassen von Nachrichten, verpackt in kurze Videosequenzen. Diese Video-Messages sind dann ähnlich wie die Vlogs anzusehen, mit dem Unterschied, dass Video-Messages nur für die gewünschte Zielperson sichtbar sind, und nicht wie die Video-Blogs für die breite Öffentlichkeit. ([TECH2005b])

3.4.1.7 Multimedia-Chat-Umgebung

Multimedia-Chat-Umgebungen sind vergleichbar mit normalen textbasierten Chats, mit dem Unterschied, dass Audio und Video hinzukommen. Das bedeutet die Chat-Teilnehmer können einander nach wie vor textbasiert schreiben, aber auch per Audio und Video kommunizieren, und das alles zur selben Zeit und live mittels Live-Streaming. ([COMP2007])

3.4.2 Electro-Server 3

Der Electro-Server 3 ist ein Socket-Server[↗]. Er wurde entworfen, um Multi-Player-Spiele und Chat-Anwendungen zu erstellen, ist mittlerweile jedoch für jegliche Art von Multi-User-Anwendung anwendbar. Der Server hat einige Features, die es den Entwicklern leicht machen Multi-Player-Spiele und Chats zu erstellen. Da der Electro-Server eine Java-Applikation ist, kann er auf jedem Rechner laufen, auf dem eine Java Runtime Environment (JRE), auch bekannt als Java Virtuelle Maschine, unterstützt wird. Neben Windows, Mac, und Linux läuft er also auch unter Unix. ([ELEC2006])

3.4.2.1 Stärken

Der Electro-Server 3 hat die folgenden Vorteile, gegenüber anderen Server-Produkten:

- Räume

  Wie beinahe alle Socket-Server, so unterstützt auch der Electro-Server mehrere Räume, in denen zur gleichen Zeit verschiedene Spiele, Chats oder andere Anwendungen abgehalten werden können.

- Raum-Variablen

  Für jeden Raum können am Server Variablen gespeichert werden. Das erlaubt ein einfaches Abspeichern von Informationen an einer zentralen Stelle. Informationen können zum Beispiel sein, ob der jeweilige Raum offen ist, oder gesperrt.

- Automatische User-Nummerierung

  Da man den Benutzern in jedem Raum automatisch eine eindeutige Nummer zuweisen lassen kann, kann man auch unterscheiden, welche Rolle diese Benutzer in zum Beispiel einem Chat, an dem nur 3 Benutzer teilnehmen dürfen, haben, also wer Teilnehmer, und wer nur Zuseher ist.

- Objekt-Transfer

  Zwischen den Clients können ActionScript-Objekte gesendet werden. Das macht es einfacher, Daten, besonders komplexe Datenstrukturen, zwischen einigen, mehreren oder allen Benutzer-Applikationen auszutauschen.

- Serverseitige Plugins

  Um die Funktionalitäten des Electro-Servers zu erweitern, ist es möglich Code in Java und vor allem ActionScript zu schreiben, auf den Server zu laden und während der Applikationen zu nutzen.

([ELEC2006])

3.4.2.2 Schwächen

Der Electro-Server 3 bringt jedoch auch einige Schwächen mit sich:

- Wenig Funktionalität

  kleine Applikationen wie kleine Chats oder Spiele sind aufgrund der unterstützten Funktionalitäten zwar schnell und mit geringem Aufwand und geringer Komplexität erstellt. Will man jedoch größere Anwendungen erstellen, so muss man sich die Funktionalitäten selbst programmieren.

Zwar ist eine API[↗] mitgeliefert, die Entwickler dabei unterstützen soll, trotzdem ist der Mehraufwand im Vergleich zu anderen Server-Produkten gigantisch. Außerdem sollte der Entwickler zusätzlich Java programmieren können.

- keine Streaming-Unterstützung
  standardmäßig gibt es keine Unterstützung von Streams. Da man jedoch mit Java zusätzliche Plugins schreiben kann, und da Java Streaming unterstützen würde, könnte man rein theoretisch den Electro-Server mit einem selbstgeschriebenen Streaming-Plugin ausstatten.

- Preis
  für die begrenzte Funktionalität hat der Electro-Server jedoch einen stolzen Preis. Ein 50-Benutzer-Lizenz-Paket kostet 300,- US-Dollar, ein 1000-Benutzer-Lizenz-Paket bereits 1.600,- US-Dollar und das Unlimitierte-Benutzer-Lizenz-Paket bekommt man um 3.000,- US-Dollar.

([ELEC2006])

3.4.2.3 Einsatzmöglichkeiten des Electro-Servers 3

Der Electro-Server 3 wurde bereits bei seiner Entwicklung auf die folgenden Einsatzgebiete zugespitzt und ist deshalb für Applikationen, die nicht in diese Bereiche fallen, nicht die ideale Lösung:

- Multi-User Chat-Applikationen
- Multi-Player-Spiele

([ELEC2006])

3.4.3 Unity2 Multiuser Server

Unity 2 wurde vom Java-Experten Derek Clayton und dem weltbekannten Flash-Entwickler Colin Moock entwickelt.

Es ist ein wohl überlegtes theoretisches Framework für die Erstellung von Multiuser-Applikationen wie beispielsweise Chats. Die Theorie hinter Unity 2 ist überraschend einfach, denn alle Multiuser-Anwendungen enthalten Benutzer, die miteinander kommunizieren, Daten austauschen und gegenseitig Funktionen aufrufen. Auch der Unity2-Server läuft auf zahlreichen Plattformen, da er lediglich Java 1.3 benötigt. ([UNIT2006])

3.4.3.1 Stärken

Der Unity2 Multiuser Server bietet den Entwicklern folgende besondere Stärken:

- Raumbasierte Architektur
  In jedem Raum befindet sich eine Gruppe an Clients, die untereinander kommunizieren können, ohne die Benutzer anderer Gruppen/Räume zu stören oder zu bemerken. Die Benutzer können jedoch mehreren Räumen gleichzeitig beiwohnen. Mehrere Räume können in Namensräume, das sind verschiedene Räume, die zu einer Gruppe von Räumen zusammengefasst sind, separiert werden. Zusätzlich nimmt der Server mittels des Raum-Objektes, das auch am Client verfügbar ist, den Entwicklern einige Lasten von den Schultern. So werden zum Beispiel bei der Erstellung eines neuen Raums, bzw. bei Löschung eines bestehenden Raums, die Raum-Listen und deren Teilnehmer-Listen bei allen Clients automatisch synchronisiert.

- Administrator-Oberfläche
  Zum Managen der Clients, Räume und Namensräume wird ein web-basiertes Server-Administrations-Tool mitgeliefert, das sehr verständlich und einfach zu bedienen ist.

- Remote-Procedure-Call
  Wie auch der FMS2 unterstützt auch der Unity2-Server das Aufrufen von Funktionen und Ausführen von Code auf den Rechnern anderer User oder am Server.

- Variablilität

  Mit dem Unity2-Server kann man jede Art von Multi-User-Anwendung erstellen, nicht nur Spiele oder Chats. Und außerdem kann der Unity2-Server mit jeder Client-Technologie die persistente TCP/IP-Sockets[↗] unterstützen, also zum Beispiel Java oder C++, genutzt werden.

([UNIT2006])

3.4.3.2 Schwächen

Der Unity2 Multiuser Server hat jedoch auch einige Nachteile:

- Keine Streaming-Unterstützung

  Der Unity2-Server unterstützt neben File-Sharing weder Audio- noch Video-Streaming.

- Java am Server

  Um serverseitige Erweiterungen zu programmieren, muss man beim Unity2-Server Java beherrschen.

- Limitiert auf einen Server

  Zwar können bis zu 2.000 Clients mit einem Server arbeiten, so sind Unity2-Applikationen jedoch nur auf eine Maschine beschränkt, im Gegensatz zum FMS2, der mehrere Server-Maschinen und somit bessere Lastenaufteilung unterstützt.

([UNIT2006])

3.4.3.3 Einsatzmöglichkeiten des Unity2-Servers

Der Unity2-Server ist besonders für die folgenden Einsatzbereiche zu empfehlen:

- Multi-User Avatar-Chat-Applikationen
- Multi-Player-Spiele

([UNIT2006])

3.4.4 Oregano Multiuser Server

Der Oregano-Server ist ein Multi-User-Server für Flash Clients. Er ist gratis und als Open-Source veröffentlicht. Da er in Java programmiert ist, läuft er ebenfalls auf vielen Plattformen, wie Linux, Windows oder Mac OS. Oregano ist besonders nützlich für Applikationen, die nur geringe Verzögerungen erlauben, wie beispielsweise Echtzeit-Chat-Systeme. ([OREG2006])

3.4.4.1 Stärken

Die Stärken des Oregano-Servers liegen vor allem in den folgenden Punkten:

- Gruppen- und Benutzer-Management
  Jede Interaktion am Oregano-Server findet in Gruppen statt. In der unlimitierten Anzahl an Gruppen können mit einigen wenigen Zeilen in ActionScript einfache Multi-User-Chats erstellt werden.

- Serialisierung
  Es ist möglich jeden in ActionScript erstellbaren Datentyp zu übertragen. Diese werden dann automatisch serialisiert und in den passenden Java-Datentyp konvertiert, um zwischen den Clients oder zwischen Server und Clients versendet werden zu können.

- Datenbank-Verbindungen
  Oregano-Server nutzt JDBC für Datenbank-Zugriffe, das bedeutet jede Datenbank, die JDBC unterstützt kann implementiert werden. Die Client-Anwendungen können mit dem Datenbank-Objekt des Oregano-Clients in ActionScript die Datenbank nutzen.

- Clustering
  Oregano-Server-Anwendungen können auch auf mehreren Servern laufen. Mit nur einigen neuen Zeilen in der Konfigurationsdatei unterstützt der Oregano-Server auch Server-Cluster.

([OREG2006])

3.4.4.2 Schwächen

Auch der Oregano-Server hat Schwächen. Diese sind nachfolgend angeführt:

- Nur Flash-Clients
  Der Oregano-Server kann ausschließlich mit Flash-Clients kommunizieren und Daten austauschen.
- Serverseitig nur Java
  Der Oregano-Server ist in Java programmiert und kann leider ausschließlich mit fortgeschrittenem Java serverseitig verändert oder adaptiert werden.

([OREG2006])

3.4.4.3 Einsatzmöglichkeiten des Oregano-Servers

Der Oregano-Server eignet sich aufgrund seiner Stärken besonders für Anwendungen für viele Benutzer aus folgenden Bereichen:

- Echtzeit-Chat-Systeme
- Multi-Player-Spiele
- Shared-Whiteboard-Anwendungen

([OREG2006])

3.4.5 Red5 Open Source Flash Server

Red5 ist ein Open-Source Flash-Server, der in Java programmiert ist und noch in seinen Kinderschuhen steckt. Er ist das Open-Source-Äquivalent zum Flash-Media-Server 2 und benutzt ebenfalls das Real Time Messaging Protocoll (RTMP), zwar nicht das originale, aber ein nahezu perfekt nachgeahmtes. Red5 läuft sowohl unter Windows und Linux als auch unter Mac OS X. ([OSFL2005], [OSFL2007])

3.4.5.1 Stärken

Der Red5-Server zeichnet sich vor allem durch seine folgenden Stärken aus:

- Audio(MP3)- und Video(FLV)-Streaming
  als einzige Server-Lösung, neben dem FMS2 natürlich, unterstützt der Red5 Audio- und Video-Streaming.
- Streams am Client speichern
  die Red5-Technologie unterstützt außerdem das clientseitige Abspeichern von Streams. Bisher jedoch leider nur im FLV-Format.
- `SharedObjects`
  auch `SharedObjects` werden unterstützt
- Remoting
  was beim FMS2 Remote-Procedure-Call heißt, nennt sich beim Red5-Server lediglich Remoting. Es bedeutet das Aufrufen und Ausführen von Funktionen auf anderen Clients oder am Server.

([OSFL2005], [OSFL2007])

3.4.5.2 Schwächen

Als nachgeahmter FMS2 zeigt er jedoch auch einige Schwächen, die nachfolgend beschrieben sind:

- Bisher nur Testversionen
  da es bisher noch keine Veröffentlichung einer Version 1.x gegeben hat, sondern lediglich Testversionen, die aktuellste ist Red5 v.06 RC2, kann man Red5 noch nicht wirklich 100%ig einsetzen.
- Nachgeahmtes RTMP
  da die Open-Source-Flash-Community das Adobe-proprietäre Real Time Messaging Protokoll bisher nur nachahmen konnten, sind aufgrund von Übersetzungsprobleme auftretende Laufzeit-Fehler durchaus möglich.

- Differenzen bei Programmiersprachen
  leider können für den FMS2 entwickelte Applikationen nicht eins zu eins übernommen werden und funktionieren mit dem Red5 nicht. Man muss nahezu alles umprogrammieren, da die komplette Struktur anders ist. Der FMS2 ist Ereignis-orientiert aufgebaut, der Red5 Objekt-orientiert.

([OSFL2005], [OSFL2007])

3.4.5.3 Einsatzmöglichkeiten des Red5-Server

Prinzipiell ist der Red5-Server in denselben Bereichen wie der FMS2 einsetzbar:

- Video-on-Demand
- Video-Blogging & Video-Messaging
- Live-Video-Übertragungen
- Live-Video-Konferenzen

([OSFL2005], [OSFL2007])

3.4.6 Konklusion

Die Gegenüberstellung der fünf Server-Produkte zeigt die deutlichen Unterschiede.

Tabelle 3.4-a Vergleichsergebnisse der Server-Produkte

	FMS2	Electro	Unity2	Oregano	Red5
Flash-Unterstützung	+	+	+	+	+
Streaming-Unterstützung	+	-	-	-	+
Funktionalitätsumfang	+	-	-	-	+
Plattformunabhängigkeit des Servers	~	+	+	+	~
Preis	-	-	~	+	+

Angefangen mit dem Electro-Server, der schon bei der Entwicklung hauptsächlich auf Multi-Player-Spiele und Chats zugespitzt wurde. Für seinen

Funktionalitätsumfang ist er jedoch überteuert. Denn muss man sich weitere Funktionalitäten ohnehin umständlich selber programmieren.

Zwar ist dies beim Electro-Server auch in ActionScript möglich, was ihn wiederum interessant erscheinen lässt, doch um zum Beispiel komplett neue, nicht im ursprünglichen System vorgesehene Möglichkeiten hinzuzufügen, muss man doch auf Java zurückgreifen, doch dies wird dann sehr schnell sehr komplex und es ist daher eher davon abzuraten.

Der Unity2-Server hat ein ähnliches Angebot an zur Verfügung stehenden Funktionalitäten wie der Electro-Server. Zusätzlich unterstützt Unity2 jedoch das sogenannte Remote-Procedure-Call (siehe *Kapitel 4.7, RPC {HAR}*) und liefert eine Administrator-Oberfläche, mit der zumindest die von Grund auf unterstützten Features administriert werden können. Außerdem ist er sehr variabel und kann nicht nur mit Flash, sondern auch mit Java oder C++ genutzt werden. Auch hier können zwar Erweiterungen programmiert werden, dies ist jedoch ausschließlich in Java möglich.

Beim Oregano-Server läuft die komplette Kommunikation serialisiert und über Java ab. Außerdem wird JDBC unterstützt, und das bedeutet, dass mit Datenbanken gearbeitet werden kann. Er ist auch einer der wenigen Server-Lösungen, die das Zusammenschließen mehrere Server zu einem Server-Cluster unterstützen. Die auf Java basierende Kommunikation ist jedoch leider dermaßen auf Flash spezialisiert, dass keine anderen Technologien mit dem Oregano-Server arbeiten können. Das Erstellen von Erweiterungen ist wiederum nur in sehr fortgeschrittenem Java möglich.

Mit dem Red5 Open Source Flash Server hat sich die OpenSource-Flash-Community viel vorgenommen. Denn da sich Adobe weigert die Spezifikation des Real Time Messagin Protokolls zu veröffentlichen, mussten sie dieses in mühevoller Arbeit nachahmen so gut es ging. Das ist auch gleich eines der Probleme, mit denen man beim Red5-Server konfrontiert wird. Da der Server nämlich noch in seinen Kinderschuhen steckt, ist auch die Fehlerquote bei der Kommunikation zwischen Flash-Player und Red5-Server über das

nachprogrammierte RTMP sehr groß, da es immer wieder zu unerwarteten Komplikationen kommt. Diese Probleme sind zwar in der aktuellsten Version 0.6 bereits größtenteils behoben, selten aber doch noch vorhanden.

Außer, dass die Programmierstruktur und die Server-Architektur ein wenig anders ist als der Flash Media Server 2, ist er diesem jedoch sehr ähnlich und die wesentlichen Features werden vom Red5 bereits unterstützt. Sobald er in einer stabilen Version verfügbar ist, kann man den Red5 bestimmt als fähige Alternative zum Flash Media Server 2 in Betracht ziehen.

Es blieb also lediglich der Flash Media Server 2 der Firma Adobe über, der sämtliche Voraussetzungen erfüllte, um unser Projekt damit durchführen zu können. Aber glücklicherweise gibt es diesen in einer Entwickler-Version, die gratis zum Download auf der Adobe-Website verfügbar ist. Diese ist zwar auf maximal zehn gleichzeitige Verbindungen eingeschränkt, kostet jedoch keine € 5.710,80 und ist somit besonders für ein Schüler-Projekt die ideale Lösung.

3.5 RTMP, RTMPT, RTMPS

Standardmäßig kommuniziert der Flash-Player mit dem Flash Media Server 2 über das so genannte „Real Time Messaging Protokoll", einem Adobe-proprietären unverschlüsselten TCP/IP-Protokoll, das für Hochgeschwindigkeitsübertragungen von Audio, Video und Daten entwickelt wurde und den Port[↗] 1935 benutzt.

So ein Verbindungsaufbau würde folgendermaßen aussehen:

```
nc.connect ("rtmp://myserver.com/myApplication");
```

Sehr oft wird dieses Protokoll von Firewalls und/oder Proxy-Servern blockiert, die meistens nur HTTP[↗] und HTTPS, also die Ports[↗] 80 und 443 zulassen. Aus diesem Grund gibt es die Möglichkeiten RTMP[↗] über diese beiden standardisierten Protokolle zu tunneln.

Wird es über HTTP[↗], also Port[↗] 80, getunnelt, so wird es RTMPT genannt und ein derartiger Verbindungsaufbau würde so geschrieben werden:

```
nc.connect ("rtmpt://myserver.com/myApplication");
```

Dasselbe über HTTPS, also Port[↗] 443, wird als RTMPS bezeichnet und würde so aussehen:

```
nc.connect ("rtmps://myserver.com/myApplication");
```

Oft reicht das „Tunneling" jedoch nicht aus und man muss auf spezielle Ports[↗] zurückgreifen. Dies funktioniert mit allen drei Protokollen und wird ganz einfach folgendermaßen geschrieben, indem nach der Server-Adresse ein Doppelpunkt und der gewünschte Port[↗] hinzugefügt werden:

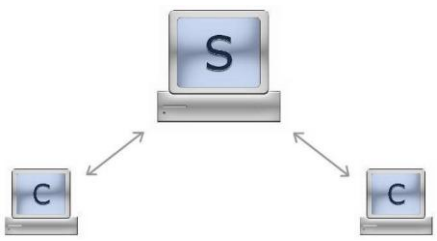
```
nc.connect ("rtmp://myserver.com:PORT/myApplication");
nc.connect ("rtmpt://myserver.com:PORT/myApplication");
nc.connect ("rtmps://myserver.com:PORT/myApplication");
```

Besonders erwähnenswert ist noch, dass HTTP[↗]- und HTTPS-Tunneling Auswirkungen auf die Performance von Echtzeit-Audio- und -Video-Übertragungen haben. ([ADOB2002])

3.6 Server-Client-Kommunikationswege

Sowohl vom Client zum Server, als auch umgekehrt, wird das Real Time Messaging Protokoll für sämtlichen Datenverkehr verwendet. Sobald die Verbindung zwischen Client und Server hergestellt ist, kann der Client auf die freigegebenen Ressourcen des Servers zugreifen, also `SharedObjects` auslesen und ändern, eigene Streams hoch- und Streams anderer Clients herunterladen, und Funktionen via Remote-Procedure-Call aufrufen. Die Clients können sogar auf anderen Clients Funktionen aufrufen. Doch sämtliche Tätigkeiten erfolgen über den Server, auf dem der FMS2 und die Applikationen laufen. Auch, wenn ein Client auf einem anderen Client etwas ausführen möchte, so führt kein Weg am Server vorbei. Denn der Server agiert als Kommunikations-Kanal zwischen den Clients. ([ADOB2005c] S. 21ff)

Abbildung 3.6–a Server-Client-Kommunikationswege

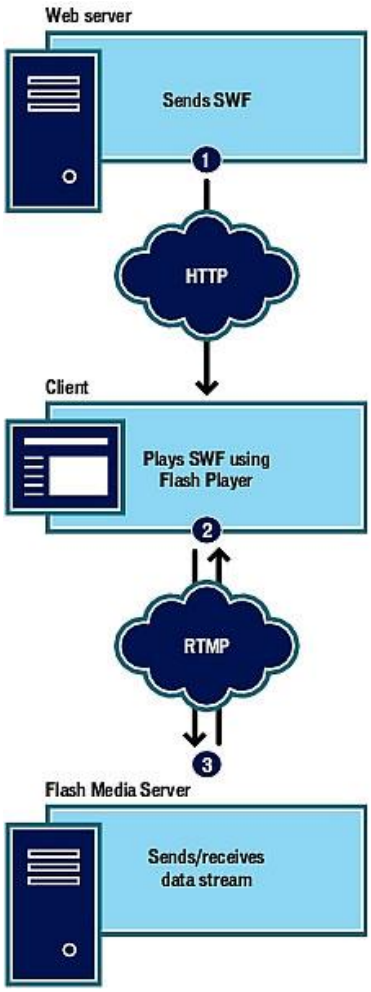


3.7 FMS-Architektur

Prinzipiell gibt es zwei Möglichkeiten für einen Client die clientseitige Applikation zu bekommen. Erstens kann er die Applikation selbst auf dem lokalen Rechner gespeichert haben und ruft sie auf. Und zweitens kann er die Applikation über den Browser bekommen und nutzen.

Der Webserver, zu dem sich der Client über den Browser verbinden muss, liefert dem Flash Player die Applikation über HTTP[↗], und sobald diese vollständig übertragen wurde, kann die Verbindungsherstellung und danach der Datenverkehr über die bestehende RTMP-Verbindung zwischen Client-Applikation und Server beginnen. ([ADOB2005c] S. 22)

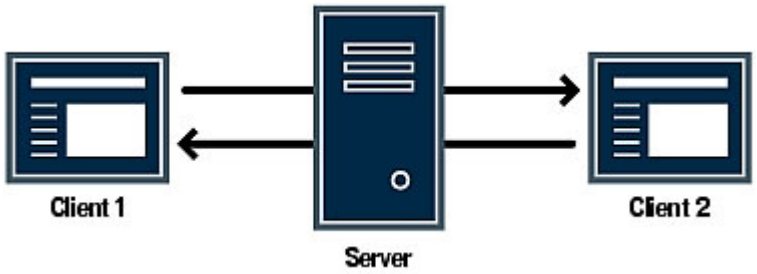
Abbildung 3.7–a Flash Media Server 2-Architektur ([ADOB2005c] S. 22)



Der FMS unterstützt natürlich nicht nur einen Client pro Applikation, sondern lässt viele Verbindungen zu ein und derselben Applikation zu.

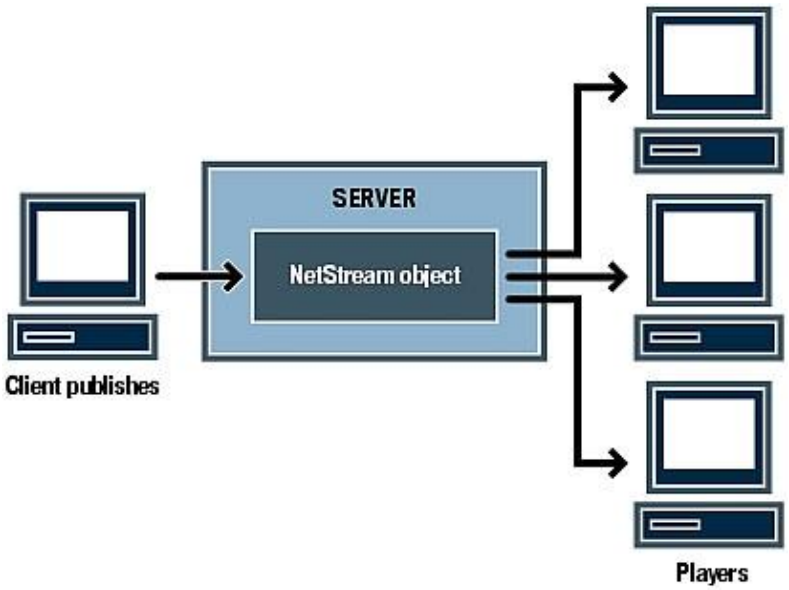
Der Server agiert hierbei als Kommunikations-Kanal zwischen den verbundenen Usern. ([ADOB2005c] S. 23)

Abbildung 3.7–b FMS2 als Kommunikationskanal ([ADOB2005c] S. 23)



Die verbundenen Clients wollen jedoch auch Daten austauschen, besonders Audio- und Video-Streams. Hierbei verwenden sie den FMS2 als Zwischenspeicher und Veröffentlichungsplattform. Jeder Client lädt seine Streams hoch und der Server lässt alle anderen Clients darauf zugreifen und die Streams wieder herunterladen. Nähere Informationen zu `NetStreams` sind in *Kapitel 4.3, NetStream-Klasse {CSI}* zu finden. ([ADOB2005c] S. 23f)

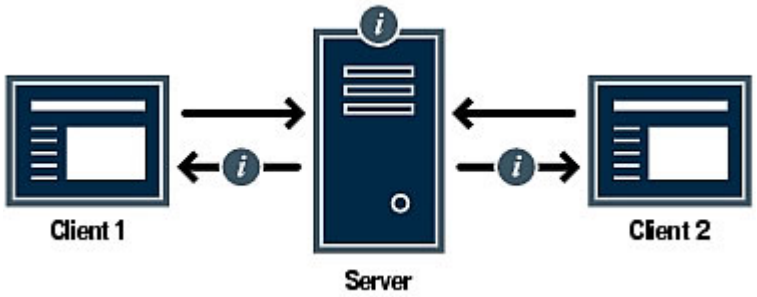
Abbildung 3.7–c Verteilen von NetStreams ([ADOB2005c] S. 24)



Neben den Streams ist die nächste Möglichkeit des Datenaustausches das Remote `SharedObject`. Der FMS stellt die `SharedObjects` zur Verfügung und alle Clients können die Daten auslesen und ändern.

Das Synchronisieren wird wiederum vom FMS übernommen und sämtliche Clients, die zum sich ändernden `SharedObject` verbunden sind, werden benachrichtigt und ihre Daten werden aktualisiert. Nähere Informationen zu `SharedObjects` sind in Kapitel *4.5, Client SharedObject-Klasse {CSI}* zu finden. ([ADOB2005c] S. 25)

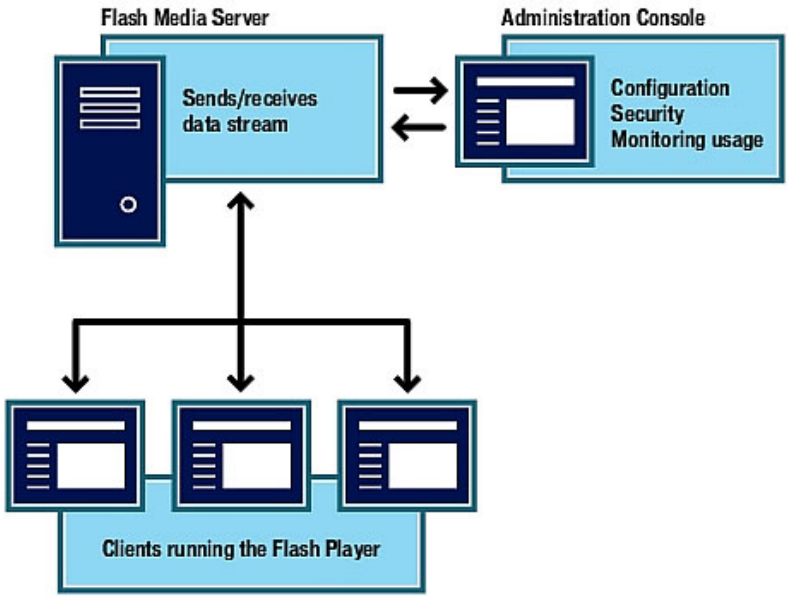
Abbildung 3.7–d Austausch von SharedObjects ([ADOB2005c] S. 25)



Und die letzte Form Daten auszutauschen ist der Remote-Procedure-Call. Nähere Informationen zu RPC sind in Kapitel *4.7, RPC {HAR}* zu finden. ([ADOB2005c] S. 31ff)

Als Administrator hat man die Aufgabe, den FMS2 zu konfigurieren, die Applikationen zu überwachen bzw. bei auftretenden Problemen zu erkennen, wo der Fehler entstanden sein könnte. Diese Aufgaben werden einem durch die Management-Konsole vereinfacht. Der Administrator kann sich mit der Management-Konsole während des laufenden Betriebs diverser Applikationen zum FMS2 verbinden, und kontrollieren, ob alles so läuft, wie es soll, in die Daten der Applikationen und Clients einsehen, oder beispielsweise neue Applikationen erstellen. Nähere Informationen zur Management-Konsole sind in Kapitel *3.10.6, Management-Konsole* zu finden. ([ADOB2005c] S. 29)

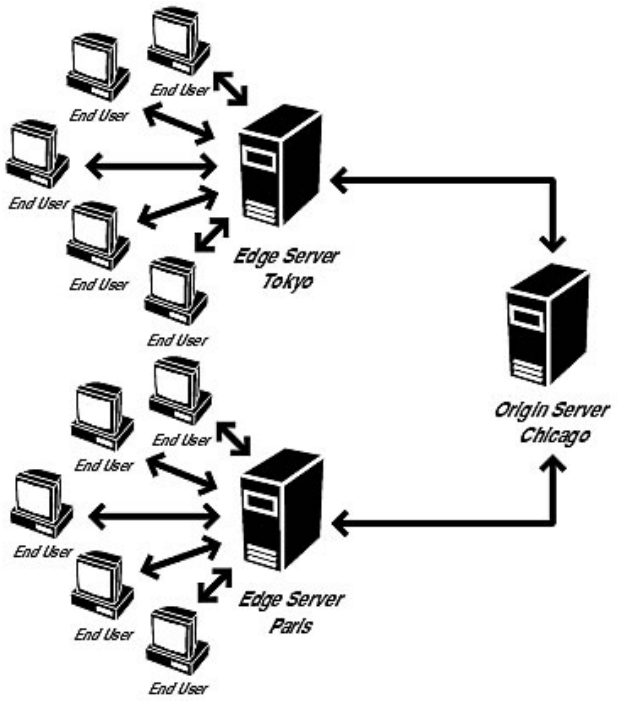
Abbildung 3.7–e Management-Workflow ([ADOB2005c] S. 29)



3.8 EDGE- und ORIGIN-Server

Beim Vorgänger des FMS2, also dem Flash-Communication-Server, war es nur möglich, dass sich die Clients direkt zum Server, auf dem die Applikation gelaufen ist, verbunden haben. Mit dem FMS2 jedoch wurde das Konzept der Remote-Applikationsausführung in die Tat umgesetzt. Die Server können nämlich Applikationen lokal als „Origin"-Server oder remote als „Edge"-Server ausführen. Edge-Server sind keine andere Art des FMS2, sie sind lediglich anders konfiguriert, nämlich um Applikationen remote[↗] auszuführen.

Abbildung 3.8–a wenige Edge-Server statt vielen Clients kommunizieren mit der Applikation ([ADOB2005h] S. 7)



Mit Edge-Servern wird die Sicherheit gesteigert, denn die Applikationen, die auf dem Origin-Server laufen, sind nicht mehr direkt mit dem Internet verbunden.

Außerdem werden die System-Ressourcen des Servers und seine Bandbreite effizienter genutzt, denn ein Großteil des Datenverkehrs passiert zwischen den Clients und den Edge-Servern.

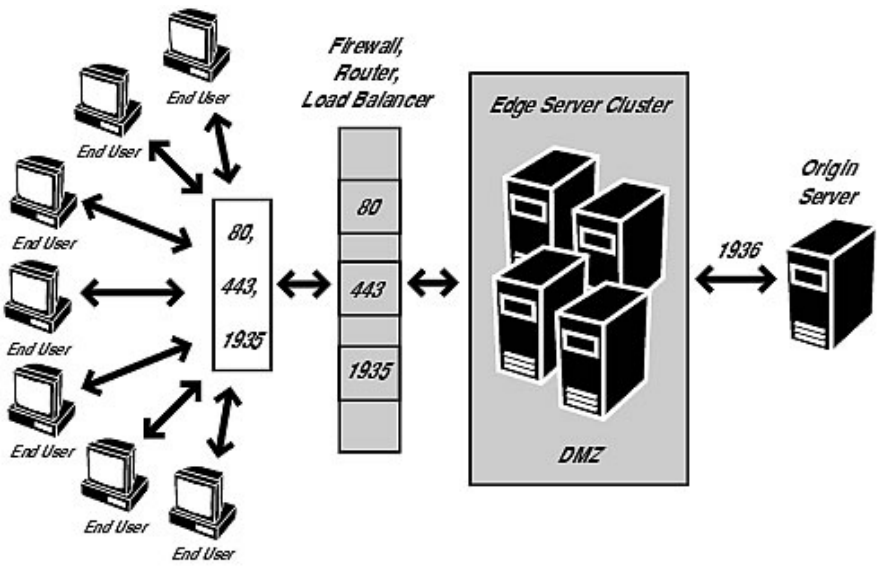
Die Edge-Server sammeln die Anfragen der Clients, fassen diese zusammen und bündeln sie und schicken sie dann an den Origin-Server. Dieser beantwortet die Anfragen und schickt die Ergebnisse zurück. Die Edge-Server verteilen die geforderten Daten dann auf die Clients. Es werden also auch die Anzahl der Verbindungen zur Applikation verringert, von vielen Clients auf einige wenige Edge-Server.

Neben der Verringerung der Anzahl an Verbindungen und dem Sammeln von Anfragen werden Daten auf Edge-Servern auch zwischengespeichert.

Dies hat den Vorteil, dass, wenn zu einem späteren Zeitpunkt nach Daten verlangt wird, die schon einmal angefordert und zwischengespeichert wurden, keine weitere Anfrage an den Origin-Server geschickt werden muss, sondern einfach die zwischengespeicherten Daten an den Client gesendet werden. Dies erleichtert wiederum die Ressourcen des Origin-Servers und seine Netzauslastung.

Um die Sicherheit eines Unternehmens zu fördern, gibt es die Möglichkeit auf Applikationsebene unbefugtes Eindringen in das interne Netzwerk zu verhindern, indem ein Cluster von Edge-Servern in die DMZ[↗] gestellt wird. Zu diesen Edge-Server verbinden sich die Clients wie gehabt über einen der drei Ports[↗] 1935, 80 oder 443, um durch die Firewall und den Router zu kommen. Werden die Authorisierungen der Clients von den Edge-Servern bestätigt, so leiten die Edge-Server über einen anderen Port[↗], zum Beispiel Port[↗] 1936, die Anfragen und Daten weiter. Neben der Steigerung der Sicherheit können so auch Internet-Kosten gespart werden.

Abbildung 3.8–b Edge-Server-Cluster in DMZ ([ADOB2005h] S. 10)



Nähere Informationen zur Konfiguration und Erstellung eines derartigen Edge-Origin-Server-Netzes sind online in den LiveDocs verfügbar.

([ADOB2005h] S.5ff)

3.9 Serverseitiges Actionscript

Um Applikationen mit dem FMS2 zu entwickeln, ist es notwendig am Server selbst zu programmieren. Dies wird mit der Server-Ausführung der Programmiersprache ActionScript getan. Doch dieses serverseitige ActionScript hat einige Eigenarten, die besonders für Flash-Entwickler, die ActionScript 2.0 gewöhnt sind, anfänglich zu Schwierigkeiten führen können. Nichts desto trotz, lassen sich mit dieser Skriptsprache äußerst effiziente und flexible Anwendungen erstellen. Denn man kann zum Beispiel die Verbindungen der Clients kontrollieren, auf Ereignisse reagieren, herausfinden was welcher Client in welcher Applikation gerade macht oder bisher getan hat oder mit anderen Servern kommunizieren. ([ADOB2005g] S. 5ff)

Das serverseitige ActionScript basiert auf der ECMA-262-Spezifikation für das ECMAScript 1.5 und nutzt daher die JavaScript-Syntax. Es unterstützt sowohl globale Methoden und Klassen als auch die Erstellung von Rich-Media-Applikationen. ([ADOB2005g] S. 5ff)

Einige dieser Klassen wären zum Beispiel die `Application`-Klasse, auf der jede serverseitige Applikation aufbaut; die `Client`-Klasse, mit der jeder zum Server verbundene Benutzer referenziert und identifiziert wird; die `File`-Klasse, die es dem Server ermöglicht mit Ordnern und Dateien zu arbeiten, die auf der Festplatte lokal abgespeichert sind bzw. werden sollen; aber auch die `Stream`- und `SharedObject`-Klassen, mit denen der Daten-Austausch zwischen den Clients und den Servern durchgeführt wird. Da ein Server-Skript selbst auch als Client einer Applikation funktionieren kann, unterstützt er selbstverständlich auch all jene Klassen, die dazu benötigt werden, wie zum Beispiel `NetConnection` oder `XML`. Natürlich kann man auch eigene Klassen, Eigenschaften und Methoden erstellen. ([ADOB2005g] S. 5ff)

Mit dem serverseitigen ActionScript ist man also ideal ausgerüstet um Applikationen zu erstellen, die von einer einfachen Chat-Anwendung bis hin zu einer höchst-komplexen Videoüberwachungs-, Sicherheits- und Kommunikations-Software reichen. ([ADOB2005g] S. 5ff)

3.10 Server-Applikationen

In den nachfolgenden Kapiteln wird von Grund auf erläutert, warum man Applikationen am FMS2 überhaupt braucht, wie man sie erstellt und betreut und wie sie arbeiten. Denn ohne diese serverseitigen Programme würden clientseitige Anwendungen zwar problemlos funktionieren, jedoch nur solange diese nicht auf externe Ressourcen angewiesen sind.

3.10.1 Aufbau von Server-Applikationen

Applikationen, die mit dem Flash Media Server 2 zusammenarbeiten, setzen natürlich auch Skripte seitens des Servers voraus, um die Kommunikation in und die Arbeit mit diesen Systemen zu ermöglichen. Hierfür bietet der Flash Media Server 2 die Möglichkeit an, die Programmteile, die Server-seitig laufen sollen, selbst zu erstellen.

Zwar gibt es für die Erstellung derartiger Skripte Referenzen, die die einzelnen möglichen Befehle erläutern, es ist jedoch nicht nur als Anfänger, sondern auch als fortgeschrittener Flash-Programmierer besonders zu Beginn äußerst schwierig zu durchschauen. Auch uns erging es nicht besser, denn selbst nach dem Durcharbeiten der kompletten FMS2-Dokumentation waren grundlegende Funktionalitäten nach wie vor unklar und unverständlich für uns. Durch langwieriges Probieren und Testen haben wir schlussendlich die Arbeitsweise des FMS2 verstanden und konnten mit der Umsetzung unserer Arbeiten beginnen.

3.10.1.1 Was sind Applikationen?

Alles, was mit dem FMS2 geschieht, wie er welche Ressourcen, welche Daten, Streams, Clients verarbeitet, behandelt etc. basiert auf Applikationen. Jedes Programm, das für den FMS2 entwickelt wird, ist eine Applikation.

Im System-Ordner des Servers, standardmäßig „`C:\Programme \Macromedia\Flash Media Server 2\`“, befindet sich ein Unterordner `Applications`, in dem sämtliche Applikationen und deren Daten gespeichert werden. Für jede Applikation, die erstellt werden soll, muss ein eigener Ordner erzeugt werden.

Der Name dieses Ordners ist dann der eigentliche Name der Applikation, mit dem im System gearbeitet wird. Hierhinein wiederum werden sämtliche Skript-Dateien gespeichert. Es muss für das Funktionieren einer Applikation zumindest eine Skriptdatei mit der Endung „.asc" oder „.js" und entweder dem zuvor angesprochenen Namen der Applikation oder der Bezeichnung „main", also „main.asc", vorhanden sein (nähere Informationen zu den verschiedenen unterstützten Dateiformaten werden im *Kapitel 3.10.1.2, Unterstützte Dateiformate* gebracht). Um das Programm überhaupt zum Laufen zu bringen, muss man eine Instanz[↗] der Applikation erzeugen. Dieser Instanz[↗] kann ein beliebiger Name gegeben werden, oder es wird der Standard-Name `_defInst_` automatisch vergeben. Es können mehrere Instanzen[↗] derselben Applikation erzeugt werden.

Die Client-Programme verbinden sich immer zu einer speziellen Instanz[↗] einer Applikation, deshalb wird der Instanz-Name vergeben, um mehrere unterscheiden zu können. Die Instanzen[↗] laufen unabhängig voneinander und haben keinerlei Zugriff auf die Daten der anderen Instanzen[↗], als wären sie komplett unterschiedliche Programme auf jeweils verschiedenen Servern. ([ADOB2005c] S.15ff)

3.10.1.2 Unterstützte Dateiformate

Diese Skript-Dateien beinhalten die Befehle und den Code, den der Programmierer entwickelt und können verschiedenen Datei-Typen entsprechen:

- ASC und JS:
  Server-Skript-Dateien, die mit dem Flash ActionScript-Editor oder einem JavaScript-Editor geschrieben wurden und im Applikations-Ordner gespeichert werden.
- ASE:
  Server-Skript-Dateien wie „.asc" oder „.js", die jedoch als binäre Dateien vorkompiliert wurden um das Laden der Dateien zu beschleunigen.

- FAR:
  Ein Paket, in das sämtliche Server-Skript-Dateien der Typen „.asc", „.js" und auch „.ase" gepackt werden. Besonders hilfreich und praktisch für bessere Handhabung, wenn die Server-Skripts auf mehrere Dateien verteilt sind.

Weitere unterstützte Dateiformate, die aber keinen Programmcode enthalten, sind:

- FLV und IDX:
  Aufgenommene Streams und deren Index-Dateien. Werden im Unterordner „streams" in jedem Applikations-Ordner gespeichert.
- SOL, SOR und FSO:

  `SharedObject`-Dateien. Eine „SOL"-Datei ist eine dauerhaft gespeicherte Datei am Client. „FSO"-Dateien sind nur am Server dauerhaft gespeichert. „SOR"-Dateien sind am Client gespeichert und haben ein passendes Abbild am Server, nämlich eine „FSO"-Datei.

([ADOB2005c] S.18ff)

3.10.2 Application-Klasse

Die `Application`-Klasse ist seit der ersten Version des Flash Communication Servers verfügbar und ist eigentlich das Objekt, ohne das keine Server-Applikation funktionieren kann. Sie bietet sämtliche Informationen einer Applikationsinstanz vom Zeitpunkt ihrer Erzeugung bis sie vom System wieder entfernt wird. Jede Instanz[↗] einer FMS-Applikation hat ein Objekt `Application`, das eine einzelne Instanz[↗] der `Application`-Klasse ist, und automatisch erstellt wird. Dieses Objekt ist eine Sammlung von Streams, `SharedObjects` und Clients (also verbundenen Benutzern). Die Methoden und Eigenschaften können mit folgender Syntax angesprochen bzw. aufgerufen werden:

```
Application.methodOrPropertyOrHandler
```

Hierbei ist zu beachten, dass `methodOrProbertyOrHandler` einen Platzhalter für die Eigenschaften und Methoden des Objekts `Application` darstellt. ([ADOB2005g] S.15f)

3.10.2.1 wichtige Methoden

Die `Application`-Klasse besitzt eine Fülle an Methoden, die wichtigsten davon sind nachfolgend zu finden:

Tabelle 3.10-a Methoden der Application-Klasse ([ADOB2005g] S.16ff)

Methode	Beschreibung
Application. clearSharedObjects(SOName)	Löschen sämtlicher `SharedObjects` der aktuellen Instanz. „SOName" stellt den eindeutigen Namen des zu löschenden `SharedObjects` dar.
Application.clearStreams (StreamName)	Löschen sämtlicher Streams der aktuellen Instanz. „StreamName" stellt den eindeutigen Namen des zu löschenden Streams dar.
Application.gc()	Aufrufen des „Garbage-Collectors", der nicht mehr benutzte Speicherbereiche löscht und wieder verfügbar macht.
Application.getStats()	Ausgeben einer Netzwerk-Statistik der aktuellen Instanz.
Application.shutdown()	Schließen und Löschen der aktuellen Instanz.

3.10.2.2 wichtige Eigenschaften

Die `Application`-Klasse besitzt außerdem einige wichtige Eigenschaften, die für die Erstellung von Programmen unverzichtbar sind:

Tabelle 3.10-b 1 Eigenschaften der `Application`*-Klasse ([ADOB2005g] S.16ff)*

Eigenschaft	Beschreibung
Application.allowDebug	Zulassen oder Sperren des Debug-Modus des Administrators.
Application.clients	Eine Liste sämtlicher im Moment verbundener Clients.
Application.config	Zugriff auf die Konfigurationsdatei der Applikation „`Application.xml`".
Application.hostname	Der Name des „Hosts" bzw. der Name des „Virtuellen Hosts.
Application.name	Name der aktuellen Applikationsinstanz.
Application.server	Plattform und Version des Servers.

Besonders wichtig für den Umgang mit Clients und die Verarbeitung ihrer Daten ist die Eigenschaft `clients`. Hierbei handelt es sich um eine Liste sämtlicher im Moment am Server angemeldeter und akzeptierter Clients. Sie kann wie ein Array behandelt werden. Man kann also mit der Eigenschaft `.length` die Anzahl der Clients auslesen oder über die eckigen Klammern „`[]`" auf die einzelnen Elemente zugreifen, also mit „`[0]`" zum Beispiel auf das 1. Element.

Um die gesamte Liste durchzugehen und beispielsweise den Namen aller Clients auszugeben wäre folgender Code notwendig:

```
for(i=0; i<Application.clients.length; i++){
      trace(Application.clients[i].name);
}
```

Hierzu ist zu sagen, dass die Funktion `trace` eine Ausgabe im Live-Log der Management-Konsole auslöst.

Weiters ist zu sagen, dass die Server-Skripts durch `Application.clients` Zugriff auf die `Client`-Objekte der einzelnen Clients bekommen und deren Methoden und Eigenschaften nutzen und aufrufen können (näheres zur `Client`-Klasse in *Kapitel 3.10.3, Client-Klasse*).

3.10.3 Client-Klasse

Die `Client`-Klasse ist seit der ersten Version des Flash Communication Server verfügbar und bietet direkten Zugriff auf die Verbindung eines Clients zur Applikation. Mit jeder neuen Verbindung eines Benutzers zur Applikation erzeugt der Server ein neues `Client`-Objekt, welches, sobald die Verbindung beendet wird, auch wieder gelöscht wird. Jeder Benutzer hat also ein eindeutiges und einzigartiges `Client`-Objekt für jede Applikation mit der er verbunden ist.

Am Server sind diese `Client`-Objekte wie bereits in *Kapitel 3.10.3.2, wichtige Eigenschaften* über die Eigenschaft `client` des `Application`-Objekts, die eine Liste aller in dieser Applikation verfügbaren Clients beinhaltet.

Jeder einzelne Eintrag in dieser Liste dient als direkte Referenz auf das `Client`-Objekt des jeweiligen Benutzers. Diese Schreibweise bietet den Server-Skripts also die Möglichkeit mit jedem einzelnen Client auch einzeln zu kommunizieren.

Anhand des `Client`-Objekts und der Methoden und Eigenschaften, die es bietet, können die Server-Skripts diverse Informationen, wie zum Beispiel IP-Adresse, auslesen und Funktionen aufrufen, die am Client selbst ausgeführt werden.

Diese spezielle Art des Funktionsaufrufs nennt sich „Remote-Procedure-Call". Das bedeutet, dass der Client eine Funktion aus den Server-Skripts aufruft, die dann am Server auch ausgeführt wird. Das gleiche funktioniert natürlich auch umgekehrt, also dass die Server-Skripts eine Funktion des Client-Programms aufrufen und ausführen. Näheres zu RPC sind in *Kapitel 4.7, RPC {HAR}* zu finden.

([ADOB2005g] S.47ff)

3.10.3.1 wichtige Methoden

Die `Client`-Klasse besitzt einige wichtige Methoden, die mit einer kurzen Erklärung nachfolgend zu finden sind:

Tabelle 3.10-c Methoden der Client-Klasse ([ADOB2005g] S.49ff)

Methode	Beschreibung
Client.call(methodName [, resultObj [, p1, ..., pN]])	Mit dieser Methode wird eine Funktion am Flash-Client oder an einem anderen Server aufgerufen und ausgeführt.

Der erste Parameter ist verpflichtend. Er gibt den Namen der Funktion an, die ausgeführt werden soll. Optional kann man ein Rückgabe-Objekt angeben, in das die Rückgabewerte der aufgerufenen Funktion gespeichert werden. Und dann gibt es auch noch die Möglichkeit, wiederum optional, Übergabeparameter an die Funktion mitzugeben. Dies können beliebig viele sein. Wenn man Übergabeparameter mitschicken will, aber keine Rückgabewerte empfangen will, so muss man zumindest `null` anstelle eines Rückgabe-Objekts hinschreiben, ansonsten kommt es zu einem Fehler.

Client.getBandwithLimit (iDirection)	Diese Methode gibt die maximale Bandbreite, die für diese Verbindung nutzbar ist, als Zahl zurück.

Der Parameter gibt die Richtung des Bandbreitenlimits an. Der Wert 0 steht für das Bandbreitenlimit der Übertragung vom Client zum Server, der Wert 1 für die umgekehrte Richtung.

Client.getStats ();	Diese Methode gibt eine Netzwerkstatistik des jeweiligen Clients zurück.

Anhand dieser Statistik kann beispielsweise die Qualität der Video- und Audio-Streams vom Server zwangsweise herabgesetzt werden, indem dieser den Clients über Remote-Procedure-Call die Anweisungen dazu gibt.

Beispiel:

```
var stats = new Object();
stats = Client.getStats ();
```

In diesem Beispiel wird die Rückgabe im Objekt „stats" gespeichert. Nun kann auf dessen Eigenschaften zugegriffen und die Details zur Netzwerkstatistik ausgelesen werden.

Diese Zeile gibt die Gesamtzahl an empfangenen Bytes im LiveLog der Management-Konsole aus:

```
trace("empfangene Bytes: " + stats.bytes_in);
```

Gesamtzahl an gesendeten Bytes:

```
stats.bytes_out
```

Anzahl an empfangenen RTMP-Nachrichten:

```
stats.msg_in
```

Anzahl an gesendeten RTMP-Nachrichten:

```
stats.msg_out
```

Anzahl an verworfenen RTMP-Nachrichten:

```
stats.msg_dropped
```

Ping-Roundtrip-Time[↗] in Millisekunden, die angibt, wie viel Zeit ein Ping-Signal benötigt, der vom Server gesendet, vom Client empfangen, wieder zurückgesendet und vom Server empfangen wird:

```
stats.ping_rtt
```

Client.ping()	Die Methode sendet ein Ping-Signal an den Client und wartet auf dessen Antwort.

Wenn der Client antwortet, dann wird `true` zurückgegeben, andernfalls `false`. Wird vor allem benutzt um zu überprüfen, ob die Client-Verbindung überhaupt noch aktiv ist.

Client.setBandwithLimit(iServerToClient, iClientToServer)	Die Methode setzt die Obergrenze für die Bandbreite des Clients für die Verbindung Client-zu-Server, Server-zu-Client oder für beides.

Der erste Parameter gibt die Bandbreitenobergrenze vom Server zum Client in Bytes pro Sekunde an. Wird hier „0" geschrieben, ändert sich nichts an den Einstellungen.

Beim zweiten Parameter ändert sich nichts, außer dass es sich hierbei um die Verbindung vom Client zum Server handelt.

3.10.3.2 wichtige Eigenschaften

Die `Client`-Klasse bietet außerdem einige wichtige Eigenschaften, die für die Erstellung von Programmen sehr hilfreich sind:

Tabelle 3.10-d Eigenschaften der Client-Klasse ([ADOB2005g] S.49ff)

Eigenschaft	Beschreibung
Client.agent	Informationen zur Version und Plattform des Flash-Clients
Client.ip	IP-Adresse des Flash-Clients als Zeichenkette
Client.protocol	Protokoll, mit dem sich der Client zum Server verbunden hat. Diese Zeichenkette kann folgende 3 Werte annehmen: ***rtmp*** RTMP über eine persistente Verbindung ***rtmpt*** RTMP getunnelt über HTTP[↗] ***rtmps*** RTMP über eine SSL-Verbindung
Client.referrer	URL der aufgerufenen SWF-Datei
Client.secure	Gibt „true" oder „false" zurück, je nachdem ob die Internetverbindung sicher (true) oder nicht (false) ist
Client.uri	URI, die der Client eingegeben hat, um sich zu dieser Instanz der Applikation zu verbinden

3.10.4 Event Modell

Wesentlich bei der Programmierung und für das Verständnis des Flash Media Servers ist das „Event Modell". Denn auf diesem Modell baut die Funktionsweise des FMS2 auf. Auf verschiedene Ereignisse, wie zum Beispiel beim Start der Applikation, oder wenn sich Clients verbinden wollen, reagieren die Server-Skripts, indem die den einzelnen „Events" zugeteilten Funktionen aufgerufen werden.

Diese Funktionen sind vom Entwickler selbst zu erstellen und können auf zwei Arten den Ereignissen zugewiesen werden.

```
Application.onAppStart = function () {
};
```

In diesem Code-Beispiel ist zu sehen, wie dem Event `onAppStart`, also dem Applikationsstart, eine so genannte anonyme Funktion zugewiesen wird. Anonyme Funktionen haben keinen Funktionsnamen. Das hat den Vorteil, dass man sich keinen Namen überlegen muss, sondern gleich direkt die Funktionalität programmieren und dem Event zuteilen kann. Andererseits kann man die Funktion jedoch aus dem restlichen Programm nicht mehr aufrufen, sie ist also an das Event gebunden.

```
Application.onAppStart = startUpFunction;
function startUpFunction () {
};
```

Dieses Beispiel unterscheidet sich zum vorherigen lediglich dadurch, dass hier dem Ereignis eine Funktion mit Funktionsnamen zugeteilt wird. Das hat den Vorteil, dass die Funktion auch vom Serverskript generell, und nicht nur durch auftreten des Events, aufgerufen werden kann.

([ADOB2005g] S.17ff)

3.10.4.1 Die Events

Die `Application`-Klasse verfügt über sieben verschiedene Events:

Tabelle 3.10-e Events der `Application`*-Klasse ([ADOB2005g] S.17ff)*

Event	Aufruf bei / Beschreibung
Application.onAppStart	Wird aufgerufen, wenn der Server die Applikation lädt.

Dieses Event wird verwendet, um globale Variablen zu setzen und das System für die problemlose Benützung vorzubereiten.

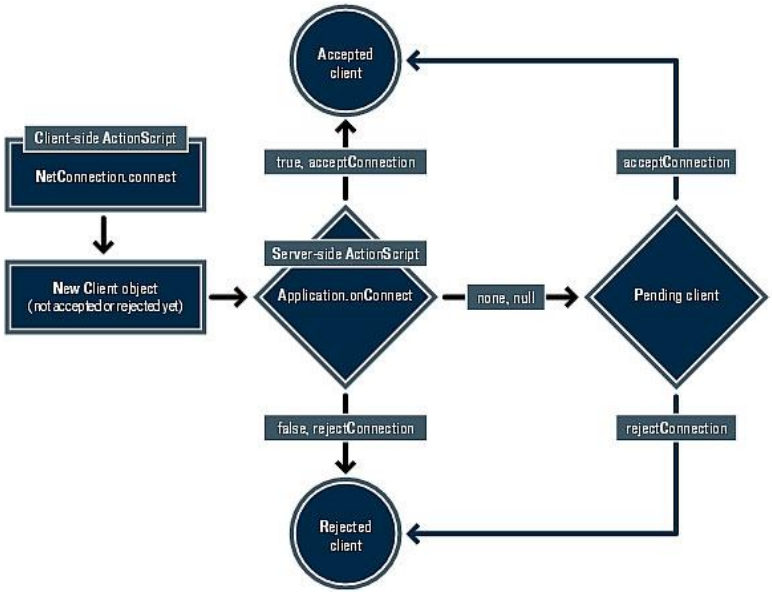
Application.onAppStop = function(infObj) { };	Wird aufgerufen, wenn der Server die Applikation löscht.

Es wird verwendet um das System aufzuräumen, also Variablen und Daten in Files oder `ShareObjects` abzuspeichern, und die Applikation „sauber" zu schließen. Parameter dieses Events ist ein Informationsobjekt, das beschreibt warum die Applikation gestoppt wird.

Application.onConnect = function (clientObj [, p1, ..., pN]) { };	Aufruf beim versuchten Verbindungsaufbau eines Clients zur Server-Applikation.

Die nachfolgende Abbildung veranschaulicht den Aufruf des `onConnect`-Events, sobald am Client `NetConnection.connect` ausgeführt wird.

Abbildung 3.10–a Verbindungsaufbau eines Clients ([ADOB2005g] S. 31)



Man kann dem Event-Handler eine Funktion zuweisen, in der man beispielsweise die Authentifizierung der Benutzer überprüft. Hierzu kann man die optionalen Parameter `(p1` bis `pN)` dazu verwenden, Anmeldeinformationen an den Server mitzuschicken. Wenn keine Funktion definiert ist, werden die Verbindungen automatisch akzeptiert. Sobald der Server eine neue Verbindung zulässt, wird das `Application.clients`-Objekt aktualisiert, in dem sämtliche mit der Applikation verbundenen Clients eingetragen und referenziert sind.

Application.onDisconnect = function (clientObj) { };	Wird aufgerufen, wenn die Verbindung eines Clients zur Applikation beendet wurde.

Dieses Ereignis wird verwendet, um Daten des Clients abzuspeichern oder beispielsweise andere Benutzer zu benachrichtigen. Der Parameter ist das Client-Objekt, das auf jenen Client referenziert, der soeben die Verbindung beendet hat.

Application.onStatus = function (infoObj) { };	Wird aufgerufen, sobald im laufenden Betrieb ein Fehler auftritt.

Parameter dieses Events ist ein Informationsobjekt, das Details zum aufgetretenen Fehler, wie Fehler-Code oder –Beschreibung, enthält.

Application.onConnectAccept = function (clientObj [, p1, …, pN]) { } ;	Wird aufgerufen, sobald ein Client, der eine FMS2-Komponente benutzt, sich erfolgreich zur Applikation verbunden hat.

Wird ausschließlich in Verbindung mit Flash-Media-Server-Komponenten verwendet. Die Parameter sind gleich wie beim Event `onConnect`.

Application.onConnectReject = function (clientObj [, p1, ..., pN]) { };	Wird aufgerufen, sobald der Versuch von einem Client, der derartige Komponenten benutzt, eine Verbindung zu einer Applikation aufzubauen fehlgeschlagen ist.

Wird ausschließlich in Verbindung mit Flash-Media-Server-Komponenten verwendet. Die Parameter sind gleich wie beim Event „`onConnect`".

Neben den Events der `Application`-Klasse gibt es auch noch bei anderen Klassen Events, die benutzt werden können. Zum Beispiel das `onSync`-Ereignis bei `SharedObjects`, welches bei der Änderung eines `Shared-Objects` aufgerufen wird.

Erwähnenswert ist auch das `__resolve`-Event der Klasse `Client`:

```
Client.__resolve = function (propName) {
      return „property_value";
};
```

Dieses Event wird aufgerufen, wann immer auf eine nicht definierte Eigenschaft der Klasse `Client` zugegriffen wird.

Der Parameter gibt den Namen der Eigenschaft an, auf die zugegriffen wurde, die es jedoch nicht gibt. Die Rückgabe der anonymen Funktion ist das, was dann als Ergebnis des zuvor missglückten Ausleseversuches weiterverwendet wird. Die Server-Skripts können also nicht definierte Eigenschaften abfangen und so tun, als würde es sie geben.

3.10.5 Server-Management-ActionScript

Neben dem „normalen" serverseitigen ActionScript gibt es noch das sogenannte Server-Management-ActionScript, das für das Management der Applikationen, der Clients und der Streams verwendet wird. Um das Server-Management-ActionScript verwenden zu können, benötigt man vor allem administrativen Zugriff auf den Flash Media Server. Um diesen zu erlangen muss bereits beim Verbindungsaufbau ein kleines, aber wesentliches Detail geändert werden. Generell kommuniziert der FMS über den Port[↗] 1935. Um sich zum Administrationsserver verbinden zu können muss der Port[↗] 1111 benutzt werden.

([ADOB2005i] S.5ff)

3.10.5.1 Verbindungsaufbau einer Admin-Verbindung

Die Syntax eines derartigen Verbindungsaufbaues ist folgendermaßen aufgebaut:

```
nc = new NetConnection ();
nc.connect ("rtmp://localhost:1111/admin", "Vwatch_Admin",
"pwVwatch_Admin");
```

Die nachfolgende Tabelle zeigt die einzelnen Segmente des Code-Beispiels und deren Bedeutungen auf.

Tabelle 3.10-f Verbindungsaufbau-Parameter für Server-Management-Skript

Parameter	im Beispiel	Beschreibung
targetURI:String	`rtmp://localhost:1111/admin`	„localhost" bedeutet, dass die Verbindung zum lokalen Rechner aufgebaut wird; „:1111" ist der Port für die Administrator-Verbindung; „admin" ist eine Standard-Applikation, die auf dem Administrationsserver läuft und sämtliche Administrator-Funktionen unterstützt
adminID:String	`"Vwatch_Admin"`	Benutzername eines Benutzers mit Administrator-Rechten
adminPW:String	`"pwVwatch_Admin"`	Benutzerpasswords des Benutzers mit Administrator-Rechten

Wenn man sich zum Admin-Server eines bestimmten virtuellen Hosts verbinden möchte, so muss man beachten, dass der Name des virtuellen Hosts zur URI[↗] gehört, und der Port[↗] erst danach geschrieben werden darf.

Das Ganze würde also, wenn man sich beispielsweise zum virtuellen Host `myVHost` verbinden wollen würde, folgendermaßen aussehen:

```
nc.connect ("rtmp://localhost/myVHost:1111/admin", "Vwatch_Admin",
"pwVwatch_Admin");
```

([ADOB2005i] S.5ff)

3.10.5.2 Aufruf einer Server-Management-Funktion

Der erfolgreiche Verbindungsaufbau zum Administrationsserver ist die Voraussetzung dafür, Server-Management-Funktionen aufzurufen. Alle diese Funktionen gemeinsam können skriptmäßig all das, was die Management-Konsole mit einem sehr benutzerfreundlichen GUI[↗] kann. Doch oft kommt es vor, dass für komplexere Applikationen auch gewisse Management-Funktionen notwendig sind, und hierzu verwendet man dann die Management-Funktionen in den Server-Skripts wie folgt:

```
nc.call ("getLiveStreams", new readStreams(), "vwatch");
```

Die nachfolgende Tabelle zeigt die einzelnen Segmente des Code-Beispiels und deren Bedeutung auf.

Tabelle 3.10-g Funktionsaufruf-Parameter für Server-Management-Skript

Parameter	im Beispiel	Beschreibung
methodName: String	`getLiveStreams`	Gibt den Namen der Server-Management-Funktion an, der aufgerufen werden soll
callbackHandle: Function	`new readStreams()`	„Callback-Handler", die angegebene Funktion wird aufgerufen sobald die Server-Management-Funktion beendet wird, und das Ergebnis wird als Übergabeparameter mitgegeben
parameter: String	`"vwatch"`	Ein Übergabeparameter an die Funktion „getLiveStreams", der den Namen der Applikation angibt, von der eine Liste der Streams ausgelesen werden soll

Doch damit dieses Code-Stück funktionieren kann, muss die Funktion, die als „Callback-Handler" angegeben wurde auch definiert werden. Die Ergebnis-daten der Server-Management-Funktion sind in einem Informationsobjekt verfügbar, das von jeder Server-Management-Funktion zurückgegeben wird.

Die zum oberen Beispiel passende Funktion könnte folgendermaßen aussehen:

```
function readStreams () {
      this.onResult = function (info) {
            trace(info.data);
      };
};
```

Hierbei wird mit dem funktionsinternen Event `onResult`, das erst aufgerufen wird, sobald das Ergebnis vollständig übertragen worden ist, eine anonyme Funktion aufgerufen, in der die Ergebnis-Daten einfach in das LiveLog der Management-Konsole ausgegeben werden.

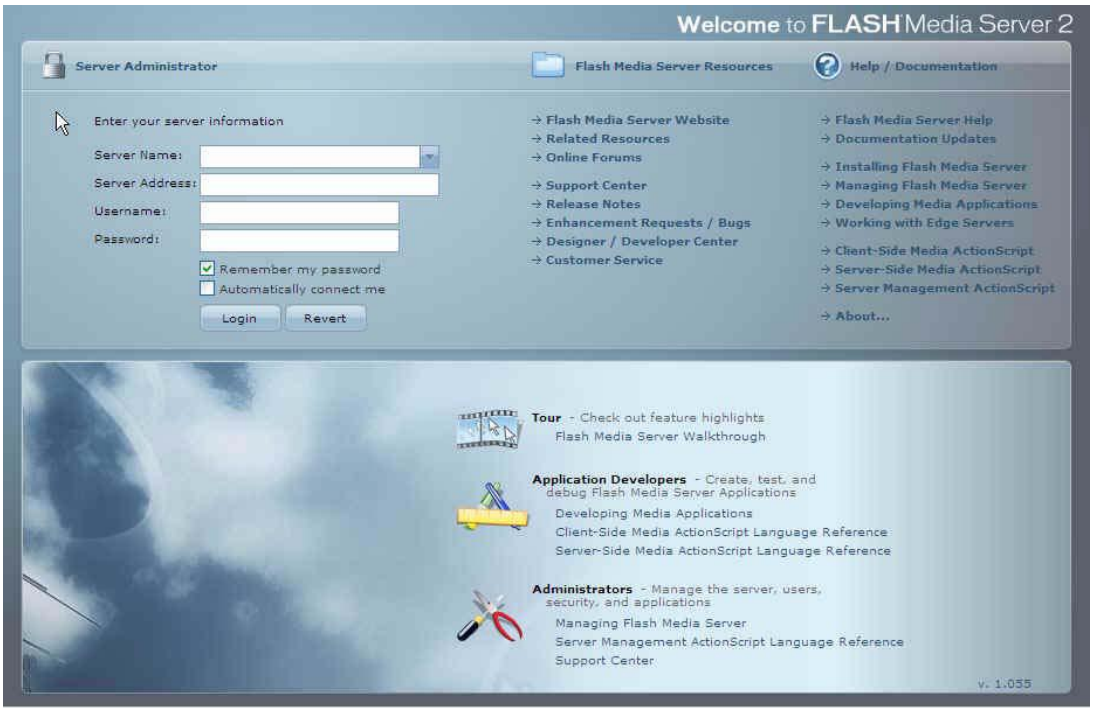
([ADOB2005i] S.6ff)

3.10.6 Management-Konsole

Die Management-Konsole des FMS2 stellt ein hilfreiches und funktionalitäts-reiches Werkzeug für System-Administratoren dar. Über das lediglich in englischer Sprache verfügbare GUI[↗] ist ein intuitives Managen des Servers, seiner Applikationen und verbundener Clients, sowie der Streams und `SharedObject`-Inhalten möglich und das Debuggen von Applikationen wird durch den LiveLog vereinfacht. Außerdem verfügt die Management-Konsole über eine Sammlung von Links zu FMS-Ressourcen, die Entwicklern helfen können Probleme zu bewältigen.

Nach dem Start der Management-Konsole befindet man sich auf dem Anmeldebildschirm. Mit den korrekten Login-Daten, die einerseits bei der Installation des FMS2 eingegeben wurden, andererseits aber auch von Administratoren erstellt und geändert werden können, verbindet man sich zu einem bestimmten Server.

Abbildung 3.10–b Anmeldebildschirm der Management-Konsole ([ADOB2005j] S. 17)



Sobald man erfolgreich eingeloggt ist, ändert sich die Oberfläche und ein neues Menü erscheint mit den Punkten „View Applications", „Manage Users" und „Manage Servers". Deren Funktionalitäten werden in den nachfolgenden Kapiteln näher beschrieben.

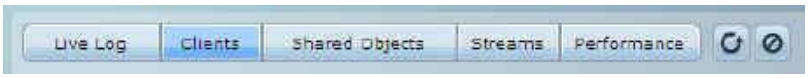
Abbildung 3.10–c Hauptnavigation der Management-Konsole ([ADOB2005j] S. 18)

([ADOB2005j] S. 16ff)

3.10.6.1 Applikationen anzeigen und managen

Der Hauptmenü-Punkt „View Applications" zeigt Informationen über die am Server laufenden Applikationen inklusive aller dazugehörigen Daten, wie Clients, Streams oder `SharedObjects`. Mit dem Betätigen dieser Schaltfläche erscheint ein neues Untermenü, das folgendermaßen aussieht und im Prinzip die Informationen der Applikationen auch gleichzeitig in fünf Bereiche einteilt.

Abbildung 3.10–d Untermenü des Applikationen-Bereiches ([ADOB2005j] S. 20)

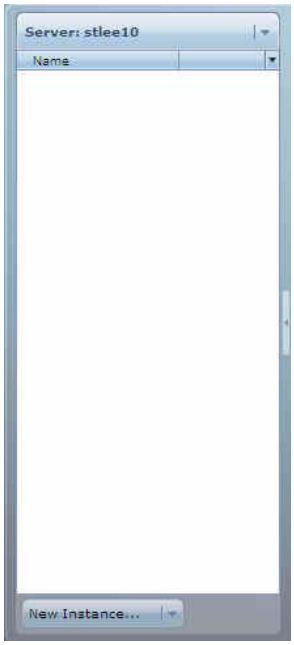


Neben den fünf Unternavigationspunkten gibt es auf der rechten Seite dieser Leiste noch zwei Schaltflächen. Der erste `Button` dient zum Neustarten der aktuellen Applikationsinstanz, der zweite zum Löschen, also zum Beenden, einer Instanz[↗].

Außerdem wird auf der linken Seite der Management-Konsole ein Bereich angezeigt, der die Instanzen[↗] der Applikationen anzeigt, und mit dem man auch neue Instanzen[↗] hinzufügen kann. Des Weiteren kann in diesem Bereich mit der ganz oben angezeigten Combo-Box zwischen den einzelnen Servern und virtuellen Hosts gewechselt werden, um deren Applikationen managen zu können. Die AuswahlListe ganz unten dient dazu, neue Instanzen[↗] zu erzeugen.

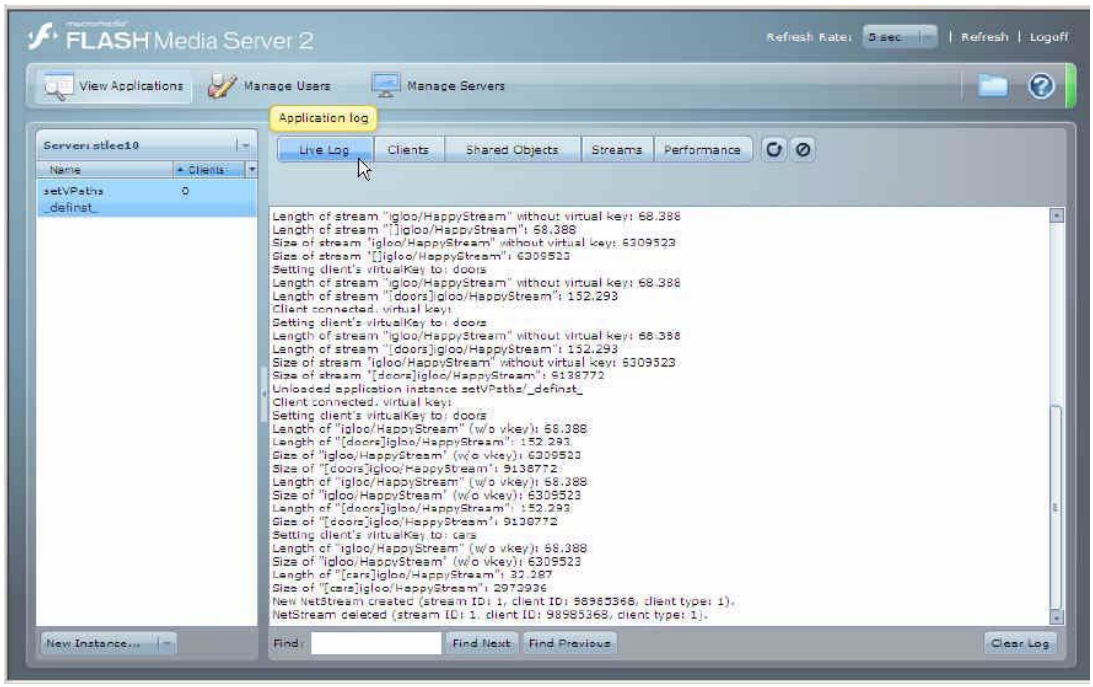
Man klickt auf die `ComboBox` und eine Liste sämtlicher auf diesem Server verfügbaren Applikationen wird angezeigt. Wählt man eine Applikation aus, so muss noch ein eindeutiger Instanzname vergeben werden und danach ist die Instanz[↗] der Applikation angelegt und gestartet.

Abbildung 3.10–e Liste laufender Applikationen ([ADOB2005j] S. 21)



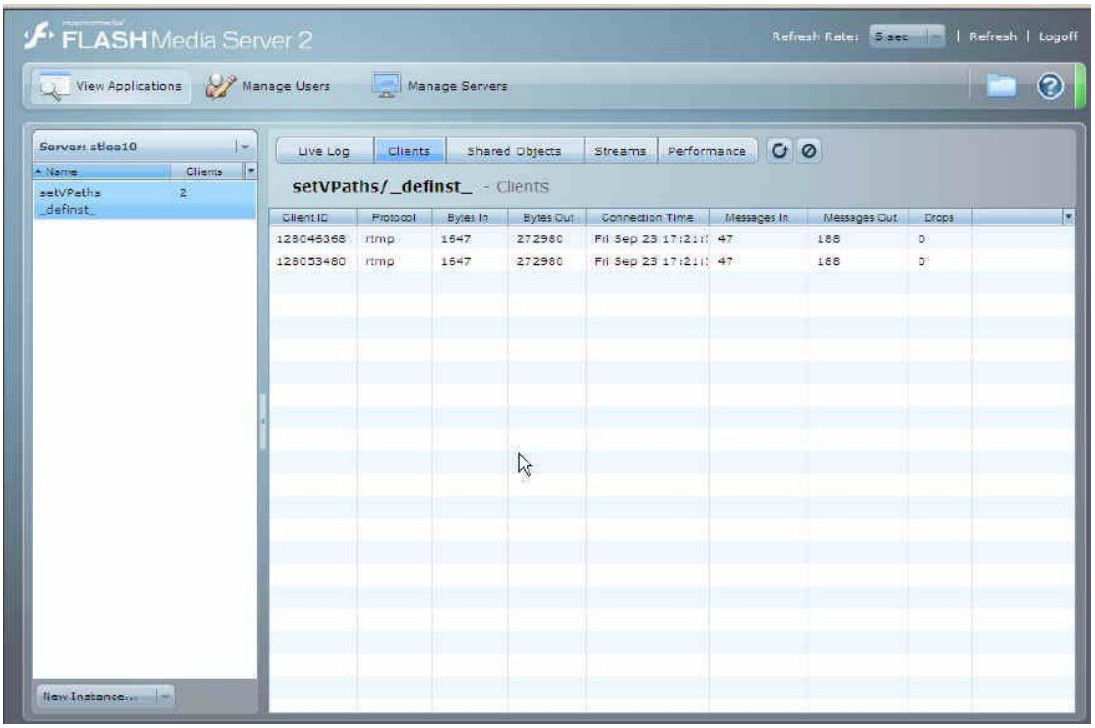
Der „Live Log" einer Applikation zeigt alle im Server-Skript mit dem Befehl `trace()` ausgegebenen Daten. Es kann beinahe jeder Datentyp auf diese Art und Weise ausgegeben werden, sogar mehrdimensionale Arrays. Mit der Such-Funktion wird es dem Administrator ermöglicht nach bestimmten Zeichenketten, oder Teilen von Wörtern, zu suchen, und der „Clear Log"-`Button` löscht die Anzeige.

Abbildung 3.10–f LiveLog der Management-Konsole ([ADOB2005j] S. 22)



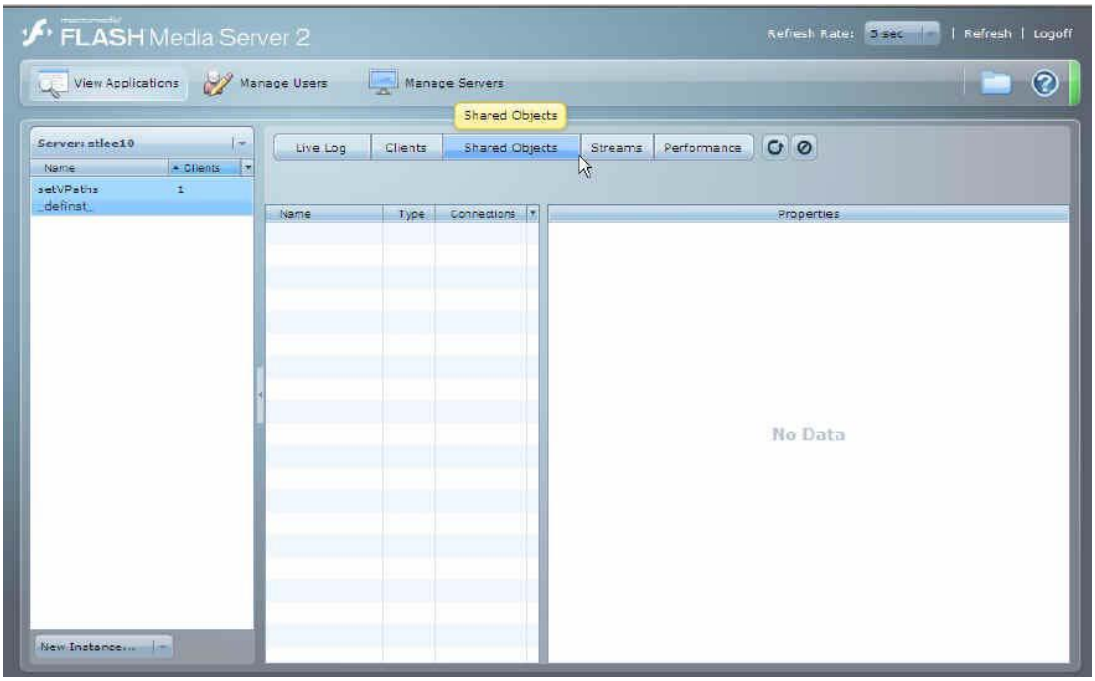
Der „Clients"-Bereich Listet sämtliche, mit der aktuellen Instanz[↗] verbundenen, Clients auf. Neben einer eindeutigen Client-ID, dem Verbindungsprotokoll und der Zeit des Verbindungsbeginns ist auch noch der Datentransfer seit Verbindungsanfrage abzulesen.

Abbildung 3.10–g Liste verbundener Clients zur aktuellen Applikation ([ADOB2005j] S. 23)



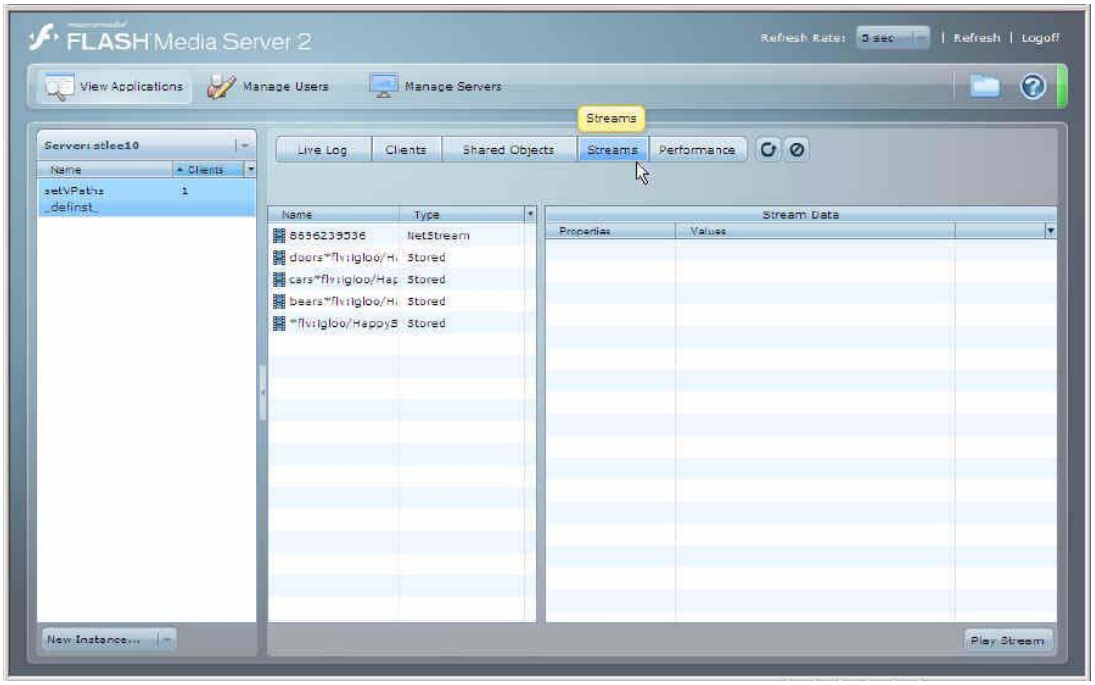
Unter dem Punkt „Shared Objects" werden alle aktiven `SharedObjects` der Applikation aufgelistet. Neben Name und Typ wird auch die Anzahl der aktuellen Verbindungen auf das jeweilige `SharedObject` angezeigt. Wählt man eines aus der Liste aus, so wird sein gesamter Inhalt dargestellt.

Abbildung 3.10–h Liste der SharedObjects der aktuellen Applikation ([ADOB2005j] S. 24)



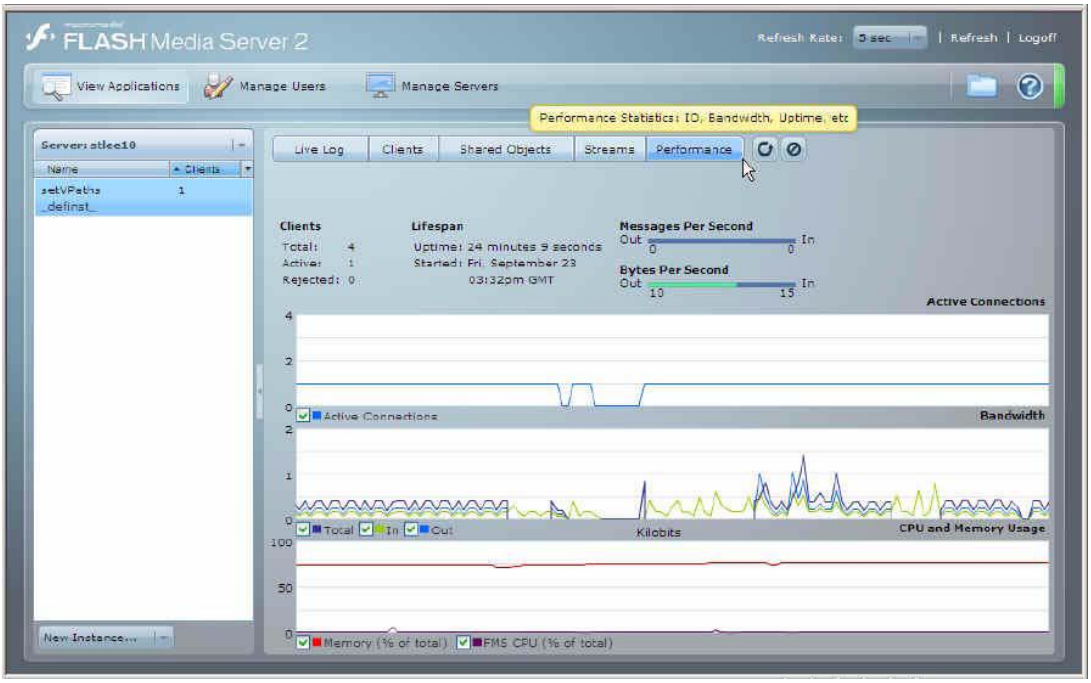
Ähnlich wie bei den „SharedObjects" ist auch der Bereich für die „Streams" aufgebaut. Eine Liste aller aktiven Streams wird mit Namen und Typ angezeigt. Wählt man einen davon aus, so kann man einerseits seine Eigenschaften einsehen, andererseits aber auch den Stream wiedergeben mit dem Betätigen der Schaltfläche „Play Stream".

Abbildung 3.10–i Liste der Streams der aktuellen Applikation ([ADOB2005j] S. 25)



Im „Performance"-Bereich werden die laufenden Informationen des Servers und der Applikation angezeigt. Diese Informationen sind die Anzahl der aktiven Clients, die verstrichene Zeit seit dem Start der Applikation, die Anzahl der Nachrichten, die empfangen oder gesendet wurden, und die Menge der empfangenen und gesendeten Daten in Bytes. Zusätzlich wird grafisch die Anzahl der aktiven Verbindungen, die Netzauslastung und sowohl CPU- als auch Speicherauslastung der Server-Hardware veranschaulicht.

Abbildung 3.10–j Performance der aktuellen Applikation ([ADOB2005j] S. 26)

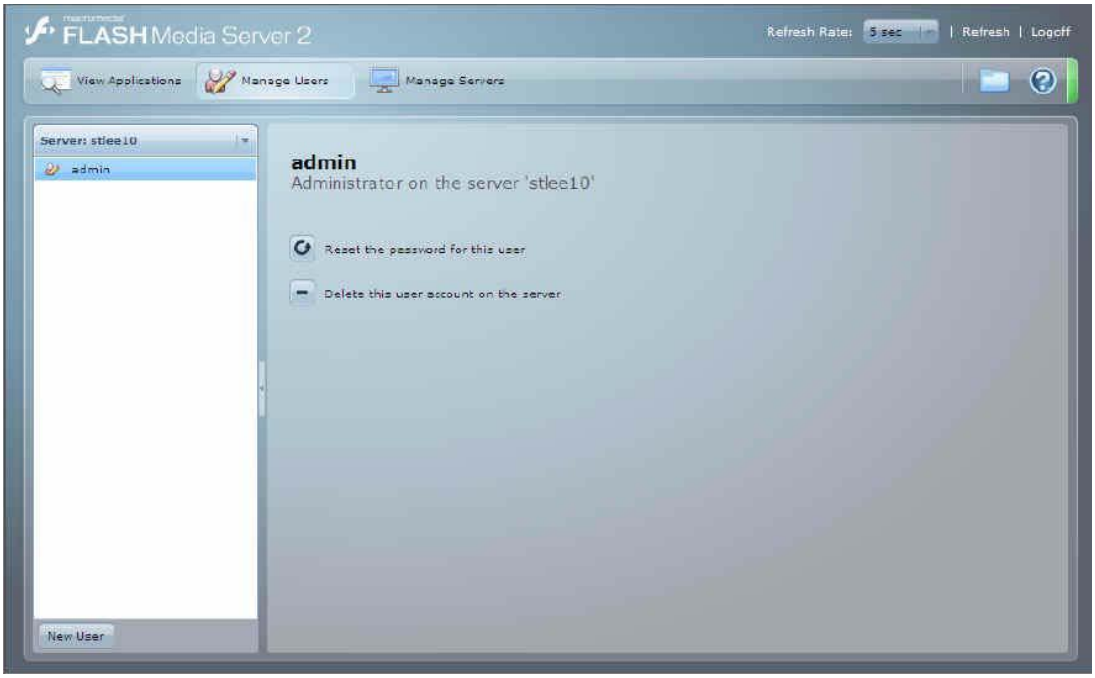


([ADOB2005j] S. 19ff)

3.10.6.2 Administrative Benutzer managen

Durch Betätigen der Schaltfläche „Manage Users" im Hauptmenü kommt man zum Bereich der Benutzerverwaltung der Administratoren. In der Liste auf der linken Seite werden die Benutzer angezeigt. Wenn einer dieser Benutzer markiert ist, so kann entweder sein Passwort neu gesetzt oder der Account gelöscht werden. Unterhalb der Liste befindet sich ein `Button` mit der Aufschrift „New User", mit dem man einen neuen administrativen Benutzer anlegen kann.

Abbildung 3.10–k Verwaltungsbereich der administrativen Benutzer ([ADOB2005j] S. 27)



([ADOB2005j] S. 27)

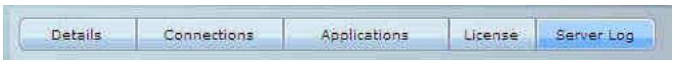
3.10.6.3 Server managen

Die dritte Schaltfläche des Hauptmenüs „Manage Server" lässt den Bereich zur Serververwaltung anzeigen. Auf der linken Seite werden alle Server und virtuellen Hosts aufgelistet, auf die der Administrator Zugriff hat. Mit der Menüleiste über der Server-Liste kann ein neuer Server hinzugefügt, die Login-Informationen eines Servers geändert und ein Server aus der Liste gelöscht werden. Mit jener Menüleiste unterhalb der Server-Liste kann die Management-Konsole zum ausgewählten Server verbunden werden und mit einem Ping-Request überprüft werden, ob dieser noch läuft und wie ausgelastet die Verbindung ist. Außerdem kann der ausgewählte Server neu gestartet oder gestoppt und alle unbenutzten Server auf einmal aus der Liste entfernt werden.

Abbildung 3.10–l Liste der verfügbaren Server ([ADOB2005j] S. 18)

Zusätzlich gibt es auch noch ein Untermenü für das Server-Management.

Abbildung 3.10–m Untermenü der Serververwaltung ([ADOB2005j] S. 29)

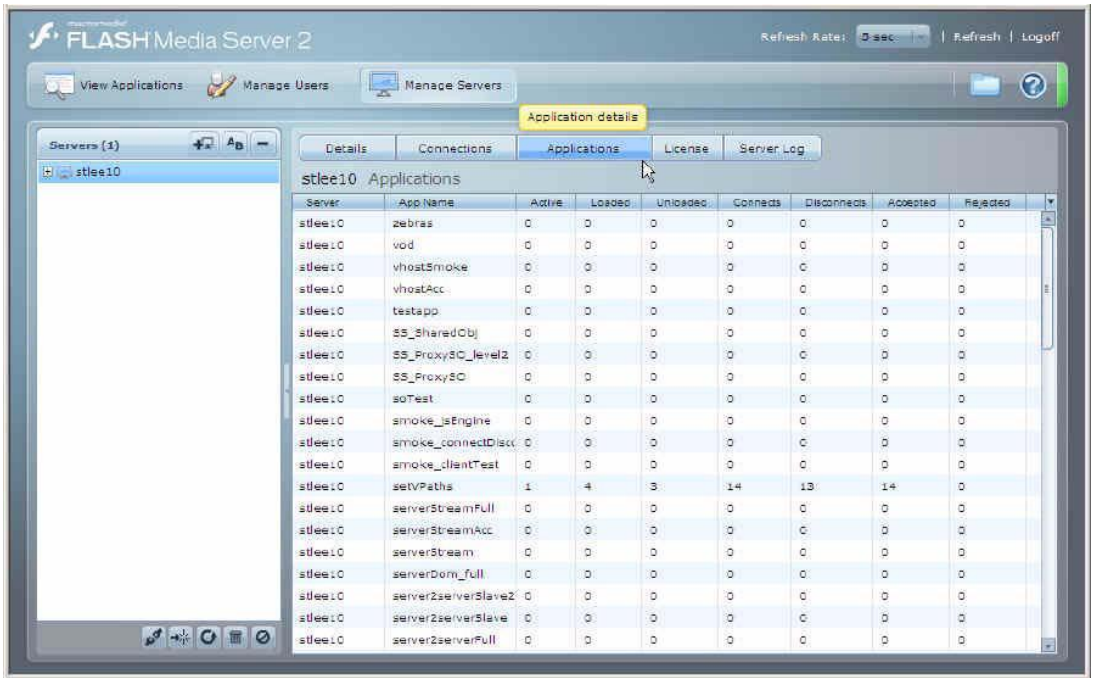


Der „Details"-Bereich des Server-Managements zeigt dieselben Informationen wie der „Performance"-Bereich des `Application`-Managements. Lediglich die Anordnung und Darstellung einzelner Daten sind verschieden.

Unter dem Punkt „Connections" des Server-Managements werden dieselben Informationen gezeigt wie im „Clients"-Bereich des `Application`-Managements, jedoch mit einem Unterschied: Hier werden sämtliche Verbindungen zum Server und nicht nur zu einer bestimmten Applikation angezeigt.

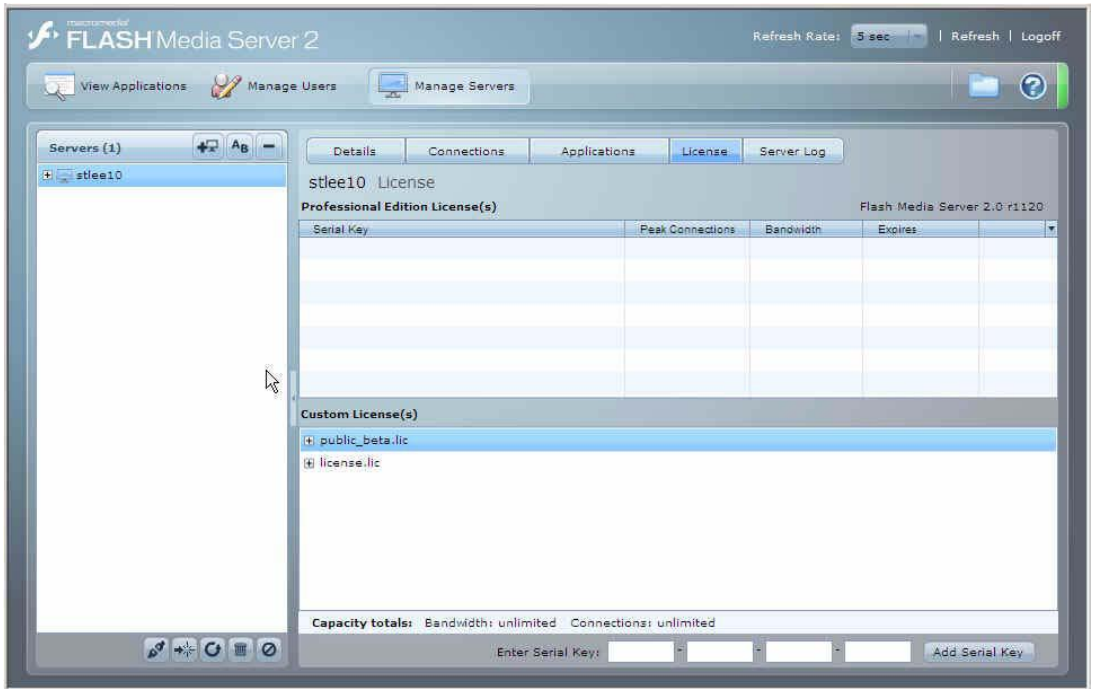
Der Bereich „`Application`s" zeigt eine Liste sämtlicher auf dem Server verfügbaren Applikationen an. Neben dem Namen, werden auch die Anzahl der laufenden Instanzen, der verbundenen Clients und der bisherigen Verbindungsanfragen angezeigt.

Abbildung 3.10–n Detailliste sämtlicher Applikationen ([ADOB2005j] S. 32)



Der „License"-Bereich zeigt alle Lizenzen, die auf dem Server abgelegt sind, an. Zusätzlich zu jeder Lizenz kann sich der Administrator anzeigen lassen, welche Beschränkungen die Lizenz mit sich bringt, also wie viele gleichzeitige Client-Verbindungen erlaubt sind, oder wie viel Bandbreite zugelassen wird. Natürlich wird auch das Ablaufdatum der Lizenz angezeigt.

Abbildung 3.10–o Lizenzinformationen des Servers ([ADOB2005j] S. 33)



Der Bereich „Server Log" ist genauso aufgebaut wie der „Live Log"-Bereich des `Application`-Managements. Es bietet ebenfalls die Such-Funktionalität und die Möglichkeit des Löschens der Anzeige. Lediglich einen Unterschied gibt es: Der Server-Log zeigt nicht die `trace()`-Ausgaben der Server-Skripts an, sondern auftretende Fehler und normale Operationen, wie zum Beispiel das Laden oder das Neustarten einer Applikation.

([ADOB2005j] S. 29ff)

3.11 FMS2-Komponenten

Adobe stellt für die Zusammenarbeit mit dem Flash Media Server 2 auch eine Reihe einfach zu bedienender FMS2-Komponenten zur Verfügung. Mit einfachen Mausklicks und kaum Programmierarbeit kann man mit Hilfe dieser Komponenten Audio- oder Video-Konferenzen, Whiteboard-Applikationen oder Chats erstellen. Auch serverseitig muss nicht viel programmiert werden. Einzig die Datei „component.asc" muss in den Server-Skripts inkludiert werden und schon funktionieren die Komponenten. Diese Datei ist standardmäßig in der Skript-Bibliothek ab dem Zeitpunkt der Installation vorhanden und kann von jedem Server-Skript geladen werden, ohne eine Pfadangabe benutzen zu müssen.

Einige dieser Komponenten sind zum Beispiel die „AudioConference"-, „VideoConference"- oder die „Whiteboard"-Komponente. Mit der sogenannten „SimpleConnect"-Komponente wird das Verbinden zum und Anmelden am Server übernommen und die Verbindung wird den restlichen Komponenten zur Verfügung gestellt.

([ADOB2005k] S. 5ff)

Wir haben unser Projekt jedoch ohne FMS2-Komponenten entwickelt, da wir schon während der Planung unser Programm so konzipiert haben, dass wir die Komponenten in ihrer Ursprungsform nicht verwenden hätten können. Denn die funktionalen Anforderungen, die wir an die Komponenten stellten, hätten von ihnen nicht erfüllt werden können.

Rein theoretisch wäre es möglich gewesen die FMS2-Komponenten zu adaptieren und für unsere Bedürfnisse umzubauen. Wir haben uns jedoch dazu entschieden, diese Arbeit einzusparen und ohne derartige Komponenten auszukommen.

4 CLIENT-SERVER-KOMMUNIKATION {CSI}

V.watch ist eine Client-Server-Applikation, da eine geographisch weit reichende Kommunikation der Benutzer untereinander nur so möglich ist. Der Server ist die zentrale Organisationseinheit und stellt wichtige Ressourcen zur Verfügung. Dieses System ermöglicht einen hohen Interaktionsgehalt zwischen den Komponenten Benutzer, Client- und Serverapplikation.

Der Inhalt dieses Kapitels behandelt die Grundlagen und Werkzeuge der Client-Server-Kommunikation in unserem Programm.

4.1 Application-Klasse {HAR}

Grundlage jeder Applikation ist die `Application`-Klasse, denn alles baut darauf auf, auch die Kommunikation mit den Clients. Allgemeine Informationen über die `Application`-Klasse sind in *Kapitel 3.10.2, Application-Klasse,* zu finden. Alle Informationen zur `Application`-Klasse sind online in den Adobe LiveDocs [ADOB2005e] verfügbar und ebendort entnommen.

4.1.1 Verbindungsanfragen behandeln

Nun kommt das zuvor erwähnte Event `onConnect` der `Application`-Klasse ins Spiel. Dieses wird von jedem Client, der versucht eine Verbindung aufzubauen, am Server aufgerufen. Der Server hat nun zwei Möglichkeiten um auf diese Verbindungsanfragen zu antworten.

Anfrage ablehnen:

```
Application.rejectConnection(clientObject, errorObject);
```

Hierbei wird die Verbindung abgelehnt. Dem Client wird keine Verbindung erlaubt, er bekommt also viel mehr ein Fehler-Objekt als eine Fehlermeldung als Antwort auf die Anfrage zurück.

Tabelle 4.1-a Verbindungsanfragen ablehnen – Parameter

Parameter	Beschreibung
clientObject	Referenz auf das Client-Objekt jenes Clients, der die Verbindungsanfrage geschickt hat
errorObject	Fehler-Objekt, das beispielsweise den Grund für das Ablehnen beinhalten kann

Anfrage annehmen:

```
Application.acceptConnection(clientObject);
```

Mit dieser Code-Zeile wird die Verbindung akzeptiert und aufgebaut. Der Client hat ab diesem Zeitpunkt Zugriff auf sämtliche Ressourcen des Servers. Dieser wiederum hat die Kontrolle über sämtliche Clients.

Tabelle 4.1-b Verbindungsanfragen annehmen – Parameter

Parameter	Beschreibung
clientObject	Referenz auf das Client-Objekt jenes Clients, der die Verbindungsanfrage geschickt hat

In den Server-Skripts des V.watch-Systems wird, bevor die Verbindung akzeptiert wird, überprüft, ob der Client registriert und berechtigt ist sich anzumelden. Des weiteren wird das übergebene Passwort mit dem im System gespeicherten verglichen und dann gibt es zwei Möglichkeiten für den Typ eines Users. Einerseits kann er ein ganz normaler V.watch-Benutzer sein, dann wird lediglich nachgeschlagen, ob vielleicht schon jemand mit seinen Benutzerdaten angemeldet ist. Ist der Benutzer bereits im System eingeloggt, wird seine Verbindung gesperrt, ansonsten wird sie akzeptiert. Die andere Möglichkeit ist, dass die Anfrage als `addCameraUser` gesendet wird. Was das zu bedeuten hat und wie die Server-Skripts damit umgehen wird in *Kapitel 8.2, Mehrere Kameras zeitgleich verwenden {HAR}* genauer erläutert.

4.1.2 Ende von Verbindungen

Aber nicht nur der Verbindungsaufbau, sondern auch die Beendigung einer Verbindung zwischen Client und Server muss natürlich behandelt und vor allem bemerkt werden. Hierzu stellt der Flash Media Server ein weiteres Event namens `onDisconnect` zur Verfügung.

Neben dem Client hat natürlich auch der Server die Möglichkeit selbst die Verbindung zu einzelnen Clients zu beenden. Dies ist notwendig, wenn der Administrator Clients, die das System unnötig belasten oder sogar überlasten, entfernen möchte.

```
Application.clients[7].disconnect();
```

Die Funktion `disconnect` wird hierbei für jenen Client aufgerufen, der im bereits angesprochenen globalen Array, in dem alle Clients gespeichert sind, den Index `7` besitzt. Es wird also die Verbindung zum 8. Client, der sich in dieser Session verbunden hat, beendet.

Da sich im Laufe einer Benutzer-Session jede Menge an Informationsmaterial für jeden Client zusammensammelt, müssen diese Daten beim Verlassen verarbeitet werden. Falls der User zum Zeitpunkt des Beendens Meetings, Großansichten oder eine Slideshow offen haben sollte, werden diese ordnungsgemäß von den Server-Skripts geschlossen. Außerdem werden sämtliche Eintragungen des Clients in den diversen globalen Variablen und `SharedObjects` entfernt und schlussendlich dann die Verbindung beendet.

4.2 NetConnection-Klasse {CSI}

Die ActionScript-Klasse `NetConnection` ist seit der ActionScript Version 1.0 in Flash implementiert und wird von Flash Playern ab der Version 7 unterstützt. Sie ist die Basis für die Kommunikation des Client-Programms mit dem Flash Media Server. Die Kommunikation mit dem Flash Communication Server, wie der Flash Media Server bis zur Version 1.5 hieß, per `NetConnection`-Klasse, ist bereits ab dem Flash Player 6 möglich.

Obwohl diese Klasse eine große Bedeutung für die Client-Server-Kommunikation hat, hat sie nur drei Methoden, zwei Eigenschaften sowie einen Statushandler.

Alle Informationen zur `NetConnection`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] und [ADOB2005e] verfügbar und ebendort entnommen.

4.2.1 Verbindung herstellen

Die wichtigste Methode ist jene, mit der eine Verbindung hergestellt wird:

```
public connect(targetURI:String) : Boolean
```

Eine `NetConnection`-Instanz kann neben der Verbindungsherstellung zu einem Server auch dazu genutzt werden, lokale Verbindungen herzustellen, um FLV-Dateien zu streamen. Zu diesem Zweck wird als Parameter `null` übergeben.

Ansonsten wird als Parameter der URI[↗] der Serverapplikation angegeben. In unserem Fall ergibt das folgenden Befehl:

```
serverConnection.connect("rtmp://81.223.149.231/vwatch",
clientID:String, clientPW:String, clientType:String,
clientLocation:String);
```

`serverConnection` ist dabei der Name der `NetConnection`-Instanz.

Wie man sieht, werden von uns zusätzlich zur Serveradresse noch einige andere Parameter übergeben. Diese Werte werden beim Verbindungsaufbau automatisch mitgesendet und können am Server ausgelesen werden.

Tabelle 4.2-a NetConnection-Parameter für Anmeldung an V.watch

Parameter	Beschreibung
targetURI:String	81.223.149.231 ist die Schul-externe IP-Adresse unseres Servers. Hinter dieser Adresse wird der Applikations-Name angegeben.
clientID:String	Username des Benutzers
clientPW:String	SHA1-Hash[↗] des Benutzerpassworts
clientType:String	Art der Anmeldung (Hauptbenutzer oder zusätzliche Kamera von V.addCam)
clientLocation:String	Standort (wird vom Benutzer angegeben)

Die Methode `connect` hat zwar einen Rückgabewert, der angibt, ob der Befehl korrekt ausgeführt wurde; dieser gibt allerdings keinerlei Auskunft darüber, ob der Verbindungsaufbau auch wirklich erfolgreich war.

Aus diesem Grund sollte der Handler `NetConnection.onStatus` verwendet werden. Dieser wird immer dann aufgerufen, wenn sich am Status der `NetConnection` irgendetwas ändert.

Mögliche Status-Veränderungen sind:

Tabelle 4.2-b Statuscodes der NetConnection-Klasse ([ADOB2005e])

Code	Art	Beschreibung
NetConnection.Call.Failed	Error	Ein RPC (Remote Procedure Call) ist fehlgeschlagen.
NetConnection.Connect.AppShutdown	Error	Die Server-Applikation wurde beendet.
NetConnection.Connect.Closed	Status	Die Verbindung wurde erfolgreich geschlossen.
NetConnection.Connect.Failed	Error	Es konnte keine Verbindung zum Server hergestellt werden.
NetConnection.Connect.Rejected	Error	Verbindungsanfrage wurde vom Server abgelehnt.
NetConnection.Connect.Success	Status	Verbindungsaufbau war auf allen Stufen erfolgreich.

Im Handler kann dann die Eigenschaft `NetConnection.isConnected` überprüft werden, um herauszufinden, ob wirklich eine Verbindung zum Server besteht.

4.2.2 Verbindung beenden

Die Server-Verbindung kann aus drei Gründen getrennt werden:

- Server schließt die Verbindung
- Client schließt die Verbindung
- Netzwerkfehler

Soll die Verbindung vom Clientprogramm aus beendet werden, so wird folgender Befehl verwendet:

```
NetConnection.close();
```

Damit werden alle Dienste, die mit dem Flash Media Server in Zusammenhang stehen, beendet und die Verbindung geschlossen.

Zum Abschluss ruft Flash den Statushandler automatisch mit dem Code `NetConnection.Connect.Closed` auf.

4.3 NetStream-Klasse {CSI}

`NetStream`-Instanzen dienen zum Austausch von Audio- und Videodaten zwischen Flash-Clients und Servern (HTTP[↗] oder FMS). Als Grundlage wird immer eine `NetConnection` benötigt.

Ab der Version 7 des Flash Players wird nicht einmal mehr ein Flash Media Server benötigt, um Flash Video zu streamen.

Die Verfügbarkeit dieser Klasse gleicht jener der `NetConnection`-Klasse.

Mit den Methoden und Eigenschaften einer `NetStream`-Instanz lässt sich das Laden und Abspielen von Dateien verfolgen und Benutzern die Möglichkeit einräumen, die Wiedergabe zu steuern (Start, Stopp, Pause, etc.).

Mit `NetStream` werden Daten half duplex[↗] übertragen, d.h. es kann mit einer Instanz gleichzeitig nur entweder ein Stream[↗] zum Flash Media Server gesendet oder von diesem empfangen werden. Mit mehreren Instanzen ist natürlich auch eine full duplex[↗] Übertragung über eine einzige Serververbindung möglich.

Alle Informationen zur `NetStream`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] und [ADOB2005e] verfügbar und ebendort entnommen.

4.3.1 Streams senden

Das Um und Auf in unserer Applikation ist das Bereitstellen von Video- und Audiodaten der Clients in Echtzeit. Dies geschieht mit jeweils zwei `NetStream`-Objekten.

Der Befehl, mit dem ein Stream[↗] zum Server gesendet werden kann, lautet folgendermaßen:

```
var my_netstream:NetStream = new NetStream(my_netconnection);
```

Als Parameter für den Konstruktor wird eine vorher definierte `NetConnection`-Instanz angegeben. Diese muss mit dem Server verbunden sein, um Streams senden und/oder empfangen zu können.

Der nächste Befehl ist notwendig, um die lokal aufgezeichneten Daten der `NetStream`-Instanz zuzuordnen. Diese Daten können entweder Videodaten von einer angeschlossenen Kamera, die von einem `Camera`-Objekt repräsentiert wird, oder Audiodaten von einem Mikrofon, das mit einem `Microphone`-Objekt verbunden ist, sein.

```
// verknüpft Audiodaten :
NetStream.attachAudio(audioSource :Microphone) ;
// verknüpft Videodaten:
NetStream.attachVideo(videoSource:Camera);
```

Zu diesem Zeitpunkt wird allerdings noch nichts zum Server gesendet. Dazu muss noch ein weiterer Schritt folgen, mit dem die Daten erst wirklich publiziert werden. Als Parameter wird der Name angegeben, mit dem der `NetStream` von nun an am Server und von allen anderen Empfängern identifiziert wird. In unserem Fall setzt sich dieser Name aus dem Usernamen des Clients plus ein entsprechendes Suffix („`_audio`“ oder „`_video`“) zusammen.

```
NetStream.publish(publishName:String);
```

Jeder V.watch-Client sendet, sobald er mit dem Server verbunden ist, zwei Streams zum Server: einen Audio- und einen Video-Stream[↗]. Wird keine Kamera gefunden, so entfällt natürlich der Video-Stream – genauso ist es mit einem Mikrofon und dem entsprechenden Audio-Stream. Sobald das Streaming gestartet wurde, wird am Server per RPC (siehe *Kapitel 4.7, RPC {HAR}*) eine Funktion aufgerufen, die diese neuen Streams registriert und in einem `SharedObject` (siehe *Kapitel 4.5, Client SharedObject-Klasse {CSI}*) verzeichnet.

4.3.2 Streams empfangen

Um Daten vom Server empfangen zu können, werden ebenfalls `NetStream`-Objekte verwendet. Die Handhabung dieser ist in unserem Programm allerdings um einiges umfangreicher als das Senden von Daten, da viel mehr empfangen als gesendet wird.

Allgemein gesehen, erfolgt das Empfangen eines Streams mit einem einzigen Befehl:

```
NetStream.play(publishName:Object);
```

Zusätzlich zum Namen, mit dem der Stream[↗] identifiziert wird, können noch drei weitere Parameter angegeben werden: Startpunkt (in Sekunden), Länge (in Sekunden) und ein weiteres Objekt, das allerdings nur zum Steuern von dynamischen Playlisten erforderlich ist und von uns nicht verwendet wird.

Will man statt FLV-Dateien lieber MP3-Dateien über ein `NetStream`-Objekt wiedergeben, so muss der Parameter der `play`-Methode mit dem Präfix „`mp3:`" oder „`id3:`" erweitert werden.

Grundsätzlich versucht Flash immer zuerst den Live-Stream mit dem angegeben Namen wiederzugeben. Ist ein solcher nicht vorhanden, wird die FLV-Datei mit diesem Namen angesprochen. Falls diese auch nicht vorhanden ist, gibt es keine Fehlermeldung, sondern Flash gibt einen leeren Stream wieder. Der Vorteil dabei ist, dass sobald zu einem späteren Zeitpunkt ein Stream[↗] mit dem entsprechenden Namen verfügbar ist, dieser sofort automatisch abgespielt wird.

Der oben angeführte Befehl reicht allerdings nicht aus, um einen Stream auch abzuspielen – dieser wird bis jetzt nur empfangen. Zum Anzeigen von Videodaten müssen diese erst einem Videofeld auf der Bühne zugewiesen werden. Dies erfolgt mit folgendem Befehl:

```
my_video.attachVideo(my_netstream:Netstream);
```

Audiodaten können theoretisch ohne weiteres abgespielt werden. Um aber die Lautstärke und andere Parameter steuern zu können, sollte der Audio-Stream einem `MovieClip` zugewiesen werden. Die Steuerung erfolgt dann über ein `Sound`-Objekt.

```
Var my_movieclip = new MovieClip();
my_movieclip.attachAudio(my_netstream:Netstream);
```

V.watch empfängt Streams an mehreren Stellen: In der Übersicht werden bis zu acht Videostreams zur gleichen Zeit wiedergegeben. Die Speicherung der entsprechenden `NetStream`-Objekte erfolgt in einem `Array`. Die Videofelder zur Anzeige werden dynamisch auf der Bühne platziert.

Analog dazu erfolgt der Empfang in Konferenzen. Zusätzlich dazu gibt es hier aber ein weiteres Array mit `MovieClips` darin, um auch die Audio-Streams wiederzugeben.

4.3.3 Streaming beenden

Analog zur `NetConnection`-Klasse kann auch hier ganz einfach mit einem Befehl der Up- oder Downstream beendet werden:

```
NetStream.close();
```

4.4 Stream-Handling am Server {HAR}

Damit der Client weiß, welche Streams es am Server gibt und wie diese heißen, müssen die Server-Skripts diese Information zur Verfügung stellen. Zwar gibt es am FMS2 eine Methode, die diese Information ausgibt, diese zählt jedoch zum Server-Management-ActionScript (siehe *Kapitel 3.10.5, Server-Management-ActionScript*) und benötigt eine Administratorverbindung.

Wie diese Verbindung erstellt werden kann, ist in Kapitel *3.10.5.1, Verbindungsaufbau einer Admin-Verbindung* näher beschrieben.

Nach dem Aufbau der Administratorverbindung müssen die Methode `getLiveStreams` aufgerufen und die Ergebnis-Daten ausgelesen werden.

Wie dies syntaktisch auszusehen hat ist in Kapitel *3.10.5.2, Aufruf einer Server-Management-Funktion* erläutert.

Diese Ergebnis-Daten müssen dann anschließend aufbereitet und den Clients zugänglich gemacht werden. Es gibt hierzu zwei Möglichkeiten: Einerseits wäre es möglich mittels Remote-Procedure-Call den Clients die Daten zu übermitteln. Die in diesem Fall jedoch bessere Variante ist ein `Shared-Object` mit den Daten zu bestücken, auf das die Clients dann je nach Bedarf zugreifen können.

Wie die Erstellung eines derartigen `SharedObjects` auszusehen hat, wird in den *Kapiteln 4.6.1, Erstellung eines SharedObjects* und *4.6.2, SharedObjects mit Daten füllen* gezeigt.

4.5 Client SharedObject-Klasse {CSI}

`SharedObjects` dienen zum Speichern von Variablen und Objekten. Es muss zwischen zwei Arten von `SharedObjects` unterschieden werden:

- local `SharedObject`
- remote `SharedObject`

Alle Informationen zur `SharedObject`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] und [ADOB2005e] verfügbar und ebendort entnommen.

4.5.1 Local SharedObject

Lokale `SharedObjects` können am besten mit Cookies[↗] in Browsern verglichen werden, da sie genau gleich funktionieren. Sie sind Textdateien, in denen Daten auf dem Rechner abgespeichert werden können, auf dem der Flash Player gerade ausgeführt wird.

```
Var localSO:SharedObject = new SharedObject();
localSO = SharedObject.getLocal(„userDetails“, „/“);
```

Mit diesem Code wird eine neue `SharedObject`-Instanz erstellt und auf das lokale File mit dem Namen `userDetails` zugegriffen. Der zweite Parameter gibt den Pfad an, in dem sich das File befindet. `SharedObjects`, die im obersten Verzeichnis („/“) gespeichert sind, können von allen SWF-Files auf dem entsprechenden Rechner geöffnet werden.

Falls das File mit dem angegeben Namen bereits existiert, kann nun auf die zuvor gespeicherten Daten zugegriffen werden. Falls nicht, wird mit dem Befehl `getLocal` ein neues File erzeugt.

Die Eigenschaft data enthält alle Werte, die in einem `SharedObject` gespeichert sind. Ihr können Variablen jeglichen Typs zugewiesen werden:

```
localSO.data.serverIP = serverIP;
localSO.data.clientID = clientID;
localSO.data.clientPW = clientPW;
```

Damit die neu hinzugefügten Daten in das File geschrieben werden, ist folgender Befehl notwendig:

```
localSO.flush();
```

Lokale `SharedObjects` sind gut geeignet, um beispielsweise Benutzerdaten oder Spielstatistiken zu speichern.

4.5.2 Remote SharedObject

Diese Art von `SharedObject` wird von Flash Media Server Applikationen erstellt, verwaltet und verteilt. Sie können von allen Clients, die mit der Applikation via `NetConnection` verbunden sind geöffnet, ausgelesen, bearbeitet und erweitert werden.

Um eine Verbindung zu einem remote `SharedObject` herzustellen, wird der Befehl `getRemote` verwendet:

```
var remoteSO:SharedObject = new SharedObject();
remoteSO = SharedObject.getRemote(“regUserSO”,
serverConnection.uri, true);
```

Der erste Parameter gibt den Namen des `SharedObject` an, der zweite die entsprechende Serveradresse und der dritte die Persistenz des Objekts. Der URI[↗] muss dabei genau jener des `NetConnection`-Objekts entsprechen – es empfiehlt sich also, die Eigenschaft `uri` der `NetConnection`-Klasse zu verwenden, um Flüchtigkeitsfehler zu vermeiden und Adressänderungen an nur einer Stelle vornehmen zu müssen.

Beim dritten Parameter kann entweder `true`, `false` oder ein lokaler Pfad angegeben werden. `False` bedeutet, dass das `SharedObject` weder am Client, noch am Server persistent ist, `true` eine dauerhafte Speicherung am Server. Wird ein lokaler Pfad angegeben, so wird eine Kopie des `SharedObject` am Client gespeichert – es ist somit auf beiden Seiten persistent, wird also dauerhaft auch nach Beendigung des SWF-Files oder der Server Applikation gespeichert.

Sobald ein SWF-File zu einem `SharedObject` am Server verbunden ist, kann es auf dessen Daten zugreifen und diese verändern. Um auf Änderungen reagieren zu können, gibt es den Handler `onSync`. Darin kann ausgelesen werden, welches Attribut verändert wurde, was der alte Wert war und wie groß das `SharedObject` nun ist.

In unserem Programm verwenden wir remote `SharedObjects` zu mehreren Zwecken. So werden unter anderem die verzeichneten Streams, die aktuell angemeldeten Benutzer und die Meeting-Daten auf diesem Weg bereitgestellt.

Leider gibt es derzeit anscheinend noch einige Probleme bei der Synchronisation der Zugriffe auf Objekte am Server – deshalb hatten wir im Falle der Meeting-Daten erhebliche Probleme:

Wie alle anderen `SharedObjects`, haben wir auch das Meeting-SO verbunden, sobald die `NetConnection` vom Server akzeptiert wurde. In diesem Fall jedoch geschah es, dass beim Zugriff auf einige Eigenschaften die Daten am Server vom zugreifenden Client überschrieben wurden.

Das kommt daher, dass Flash Player und Flash Media Server automatisch zu große Inkonsistenz zwischen den Datenbeständen erkennen und die Daten synchronisieren. Anscheinend erfolgte diese Synchronisierung aber in dem Sinne falsch, dass die älteren Daten vom Client die aktuelleren Daten auf dem Server überschrieben.

Nachdem wir einige Testfälle definierten und abgearbeitet hatten, werteten wir die Log-Files aus und kamen zu dem Schluss, dass der Fehler nicht bei uns liegt, zumal die Synchronisation bei den anderen `SharedObjects` ja auch einwandfrei funktioniert.

Als Lösung kam also nur eine Variante in Frage: die Verbindung wird nun nicht mehr gleich zu Beginn hergestellt, sondern immer nur dann, wenn Daten benötigt werden. Danach werden die Daten ausgelesen und die Verbindung sofort wieder geschlossen. So kann es nie passieren, dass clientseitige Daten diejenigen am Server überschreiben, da ja nie welche vorhanden sind.

4.6 Server SharedObject-Klasse {HAR}

Wenn man am Flash Media Server 2 ein `SharedObject` erstellt, dann wird dieses zu einem remote[↗] `SharedObject`. Das bedeutet, dass es von den mit dem Server verbundenen Clients ausgelesen und verändert werden kann.

Alle Informationen zu den `Remote SharedObject` sind online in den Adobe LiveDocs [ADOB2005e] verfügbar und eben dort entnommen.

4.6.1 Erstellung eines SharedObjects

Um ein `Remote SharedObject` am Server zu erstellen, ist eine ähnliche Syntax nötig wie am Client für ein lokales `SharedObject`. Ein Unterschied ist jedoch die Variable, in die das `SharedObject` gespeichert wird. Am Server wird es einer Eigenschaft der Applikation zugewiesen. Des Weiteren wird nicht die Methode `getLocal`, sondern eine mit dem Namen `get` aufgerufen, um das `SharedObject` zu erzeugen. Und zu guter Letzt die Parameter.

Der erste Parameter steht für den Namen der `SharedObject`-Eigenschaft, der zweite Parameter gibt an, ob es sich um ein persistentes `SharedObject` handelt oder nicht. Ist dieser Parameter `true`, so ist es ein `SharedObject`, das am Server zusätzlich gespeichert wird. `False` bedeutet, dass das `SharedObject` nach Beendigung der Applikation gelöscht wird.

Ein Beispiel für die Erstellung eines `SharedObjects` ist hier zu sehen:

```
Application.SoregUserSO = SharedObject.get ("regUserSO",true);
Application.SoregUserSO.clear();
Application.SoregUserSO.onSync = function (List) {
      for (var k in List) {
            trace("name = " + List[k].name + ", event = " +
             List[k].code);
      }
};
```

Nach der Erstellung des `SharedObjects` wird es auch noch gelöscht, um sicherzugehen, dass es auch wirklich leer ist. Dies ist eine reine Sicherheitsmaßnahme, denn es könnte ja sein, dass ein `SharedObject` mit demselben Namen am Server bereits gespeichert worden ist, dann würde kein neues erstellt werden, sondern einfach nur zu diesem verbunden werden. Zusätzlich wird auch noch ein Event `onSync` dem `SharedObject` zugewiesen. Dies wird immer dann aufgerufen, wenn von Clients die Inhalte des `SharedObjects` geändert oder gelöscht werden.

4.6.2 SharedObjects mit Daten füllen

Nachdem wir ein `Remote SharedObject` erstellt haben, müssen wir es noch mit Daten füllen. Hierzu muss zu allererst der Zugriff auf das `SharedObject` kurzzeitig gesperrt werden, um Inkonsistenzen zu vermeiden. Danach wird mit der Methode `setProperty` die Eigenschaft des `SharedObjects` mit den neuen Daten befüllt. Der erste Parameter stellt wieder den Namen der Eigenschaft dar, der zweite ist das Datenbündel, das in das `SharedObject` gespeichert werden soll, und kann jedem beliebigen Datentyp entsprechen.

Nachdem die Daten gespeichert wurden, muss die Zugriffssperre natürlich wieder aufgehoben werden.

```
Application.SoregUserSO.lock();
Application.SoregUserSO.setProperty ("regUserSO",
registeredUserArray);
Application.SoregUserSO.unlock();
```

Im Laufe der Server-Skripts wird es wahrscheinlich des Öfteren vorkommen, dass die `SharedObjects` aktualisiert werden, also mit aktualisierten Daten bestückt werden. Hierzu ist es ratsam, zuerst die Daten aus dem `Shared-Object` auszulesen, diese Daten dann zu aktualisieren und danach wiederum zurückzuspeichern.

4.6.3 Daten aus SharedObjects auslesen

Um nun die Daten am Server auch wiederum auslesen zu können wird mit der Methode `getProperty` gearbeitet. Diese hat als einzigen Parameter den Namen der Eigenschaft des `SharedObjects`, die man auslesen möchte. Der Inhalt der `SharedObject`-Eigenschaft wird dann in eine Variable gespeichert, im folgenden Beispiel ist dies die Variable `registeredUserArray`.

```
registeredUserArray = Application.SomeetingsSO.getProperty
("regUserSO");
```

4.7 RPC {HAR}

Wie bereits mehrmals erwähnt, ist es bei der Erstellung einer Software, die den Flash Media Server 2 verwendet, nahezu unumgänglich die sogenannten Remote-Procedure-Calls zu nutzen. Remote-Procedure-Call ist eine besondere Form des Funktionsaufrufes und bedeutet, dass ein Client, der sich an einem beliebig entfernten Ort befindet, eine Funktion aus den Server-Skripts aufruft, die dann am Server auch ausgeführt wird. Das gleiche funktioniert natürlich auch umgekehrt, also dass die Server-Skripts eine Funktion des Client-Programms aufrufen und ausführen. An die aufgerufenen Funktionen können natürlich auch Übergabeparameter mitgeschickt werden.

Außerdem können die Rückgabewerte über das Rückgabeobjekt empfangen und ausgelesen werden. So können mit dieser Funktionalität relativ einfach Daten ausgetauscht und verschiedene Ereignisse aufgerufen werden. Die RPC-Funktionen arbeiten genauso wie die bisher vorgestellten Events, nur dass diese Ereignisse vom direkten Gegenüber, also Server bzw. Client, ausgelöst werden.

Wichtig für die Durchführbarkeit ist, dass die Funktion der Verbindung zum Gegenüber zugeteilt wird. So muss eine Funktion, die am Client definiert ist, und dem Server zur Verfügung stehen soll, dem `NetConnection`-Objekt zugeordnet werden (siehe *Kapitel 4.7.3, Definieren von Client-Remote-Funktionen {CSI}*).

Am Server jedoch gibt es kein `NetConnection`-Objekt, das die Verbindung zum Client beschreibt. Hier muss mittels des Client-Objektes gearbeitet werden. Es gibt nun 2 Möglichkeiten den Clients eine Funktion zuzuweisen:

```
Application.onConnect = function (clientObj) {
clientObj.newFunction = function () {
      return 8;
};
```

Hierbei wird die Funktion `newFunction` immer einem bestimmten Client zugeteilt, und zwar jenem, der sich gerade verbinden möchte.

```
Client.prototype.newFunction = function () {
      return 8;
};
```

Dieselbe Funktion wird mit dem Präfix `Client.prototype` einmal definiert und somit jedem Client zur Verfügung gestellt.

([ADOB2005g] S.52ff), ([ADOB2005c] S. 31ff)

4.7.1 Serverseitiges Aufrufen von Funktionen am Client

Zwar bestehen am Server mehr als nur eine einzige Verbindung zu Clients – ganz anders als beim Client, denn der Client verbindet sich nur zu einem Server, der Server hingegen zu vielen Clients – so wird der Aufruf von Funktionen jedoch sehr ähnlich notiert. Anstelle des `NetConnection`-Objektes wird hier das `Client`-Objekt verwendet. Wie die Methode `call` der Klasse `Client` anzuwenden ist, wird in *Kapitel 3.10.3, Client-Klasse* beschrieben.

```
For (var I = 0; I < Application.clients.length; i++){
      if (Application.clients[i].name == "special_client")  {
      Application.clients[i].call("newFunction",null);
      }
}
```

Da, wie in *Kapitel 3.10.3.2, wichtige Eigenschaften* bereits angesprochen, der Zugriff auf die Client-Objekte der mit dem System verbundenen Clients hauptsächlich über die Eigenschaft `clients` des `Application`-Objektes `Application` möglich ist, muss die über diese Eigenschaft erreichbare Liste an `Client`-Referenzen in einer Schleife durchlaufen werden, um dann bei dem gewünschten Client, im Beispielfall dem Client mit dem Namen `spezial_client`, eine beliebige Funktion aufzurufen.

```
Application.broadcastMsg("newFunction");
```

Um bei allen Clients auf einmal die gleiche Funktion aufzurufen, gibt es eine weitaus schnellere Möglichkeit: Die Methode `Application.broadcastMsg` ist weitaus effizienter als die Liste der Clients durchzulaufen und einzeln die Funktion `newFunction` aufzurufen. Es besteht hierbei die Möglichkeit Übergabeparameter mitzuschicken. Diese werden nach dem Funktionsnamen mit Beistrich getrennt einfach angefügt. Die Möglichkeit Rückgabeobjekte zu empfangen besteht hierbei jedoch nicht.

4.7.2 Serverfunktionen clientseitig aufrufen {CSI}

Sobald eine Verbindung zum Flash Media Server mittels `NetConnection` hergestellt wurde, kann auf freigegebene Funktionen desselben zugegriffen werden.

Ein Aufruf einer solchen Funktion geschieht mit dem `call`-Befehl der `NetConnection`-Klasse:

```
serverConnection.call("functionName", serverReturnObject:Object,
[param1, param2, …]);
```

Besonders wichtig ist der zweite Parameter: Er gibt ein Objekt an, von dem die Rückgabewerte vom Server empfangen werden. Diese Werte sind Eigenschaften des angegebenen Objekts.

So ein Objekt wird am besten mit einer `onResult`-Methode versehen, die aufgerufen wird, sobald ein Wert vom Server eingeht.

```
Var serverReturnObject:Object = new Object();
serverReturnObject.onResult = function(returnValues:Object):Void {
      var localVar1:Number = returnValues.returnedNumber;
      var localVar2:String = returnValues.returnedString;
};
```

4.7.3 Definieren von Client-Remote-Funktionen {CSI}

Um Funktionen, die am Client definiert sind, vom Server aus aufrufen zu können, müssen sie in folgender Art und Weise der `NetConnection` zugewiesen werden:

```
serverConnection.my_function = my_function;
function my_function (param:Number):Void {
      trace (param);
};
```

Natürlich wäre es auch möglich, die Funktion direkt als Eigenschaft der `NetConnection` zu definieren und sie nicht nur zuzuweisen.

Die oben angeführte Schreibweise hat allerdings den Vorteil, dass die Funktion `my_function` so auch im clientseitigen ActionScript aufgerufen werden kann, was von uns des Öfteren benötigt wird.

5 KAMERAS & VIDEO {CSI}

Wozu wir in unserem Projekt und bei unserer Software Kameras benötigen, ist ganz klar: Videoüberwachung ist nutzlos ohne Videos.

Während alte analoge Systeme nur mit herkömmlichen Überwachungskameras kompatibel waren, kann unsere Software mit allen Arten von Kameras arbeiten – sowohl analoge als auch digitale.

5.1 Analoge Kameras

Im Prinzip erzeugt jede Kamera ganz zu Beginn ein analoges Signal – schließlich ist ja auch das Licht, das eingefangen wird, nicht digital. So genannte analoge Videokameras zeichnen die Daten aber auch analog auf – und geben sie als analoges Signal aus.

Von analogen Kameras wird gesprochen, wenn diese das Material auch analog, z.B. auf Bändern, abspeichern.

Es gibt mehrere Arten, ein analoges Videosignal zu übertragen. S-Video und Video sind zwei davon.

5.2 Digitalisierung

Um analoge Videosignale mit einem Computer verarbeiten zu können, müssen diese erst digitalisiert werden. Dieser Vorgang besteht aus Abtastung und Quantisierung.

Bei Audiodaten sind dabei Abtastfrequenz und die Anzahl der Quantisierungsstufen ausschlaggebend. Die Abtastung entspricht bei Videosignalen der Auflösung (DPI), die Quanitisierung der Farbtiefe (z.B. 8 Bit = 256 Stufen).

Wenn von digitalem Video gesprochen wird, sind immer digitalisierte Komponenten (Y, C_R, C_B) gemeint.

Tabelle 5.2-a Digitales Video im Vergleich ([HENN2003] S. 186, S. 189, S. 191)

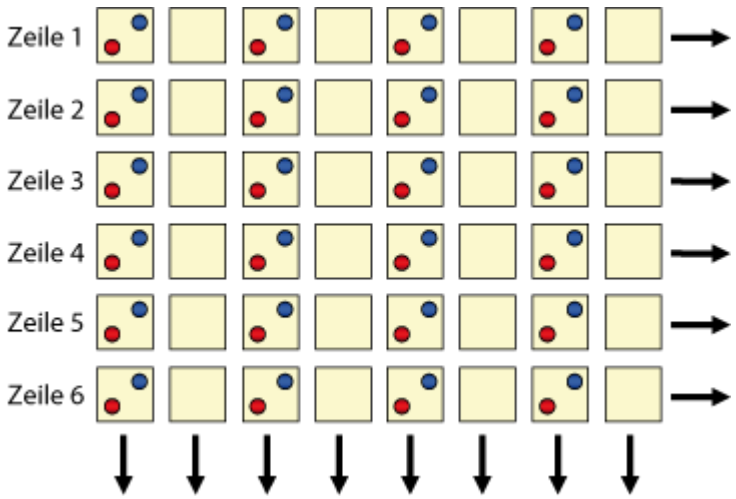
	SDTV	HDTV	Digitale Filmkamera
Auflösung	720 x 576 (0,4 MegaPixel)	1920 x 1080 (2,0 MegaPixel)	4096 x 3072 (12 MegaPixel)
Quantisierung	8 Bit	10 Bit	14 Bit
Sampling	4:2:2 /Y,C_R,C_B	4:2:2 /Y,C_R,C_B	4:4:4/RGB
Datenrate (bei 25 FPS)	125 Mbit/s	1,485 Gbit/s	17,3 Gbit/s

5.2.1 Sampling

In den benutzerorientierten Farbmodellen YC_RC_B[↗] (gleichbedeutend mit YUV) und YIQ wird das so genannte 4:2:2 Sampling (Abtastung) angewandt, was bedeutet, dass C_R & C_B nur halb so oft abgetastet werden, wie die Luminanz. Es werden die Farbinformationen also ungenauer gespeichert als Helligkeitsinformationen.

Hier ein Beispiel anhand einer Grafik:

Abbildung 5.2–a Umschalten 4:2:2 Sampling ([HENN2003] S. 102)



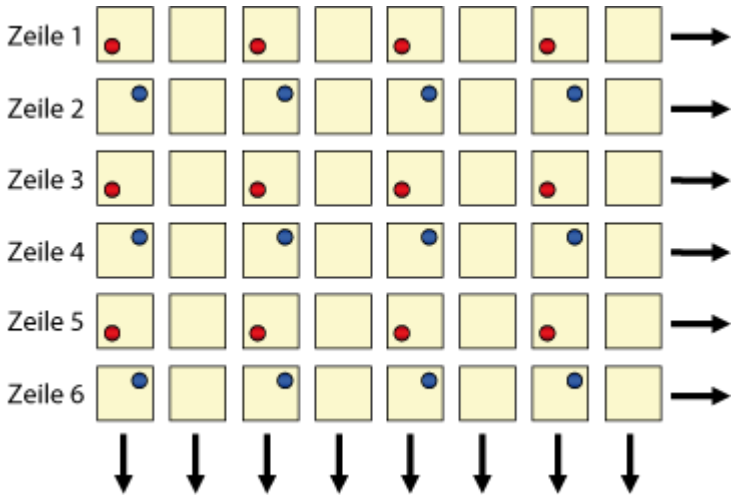
Die gelben Kästen repräsentieren die Luminanz, die in jedem Schritt abgetastet wird. Die roten bzw. blauen Kreise zeigen, dass C_R bzw. C_B nur halb so oft abgetastet werden.

([HENN2003] S. 102f)

5.2.2 Komprimierung / DV

Um der enormen Datenrate von 270 Mbit/s, die ein so quantisiertes Signal mit sich bringt, Herr zu werden, muss komprimiert werden. Das geschieht z.B. im sehr weit verbreiteten DV-Standard von 1994. DV steht für Digital Video und definiert eine 4:2:0 Abtastung für PAL bzw. 4:1:1 für den amerikanischen Standard NTSC. Das 4:2:0 Subsampling-Schema wird außerdem in den JPEG- und MPEG-Datenformaten verwendet.

Abbildung 5.2–b Umschalten 4:2:0 Sampling bei DV ([HENN2003] S. 102)



Das erste Halbbild (die ungeraden Zeilen) hat als Farbinformation nur die C_R Komponente und das zweite Halbbild (die geraden Zeilen) hat nur die C_B Komponente und zwar mit Faktor zwei weniger Abtastwerten als die Luminanz. Weiters erfolgt die Quantisierung nur mit 8 anstatt mit 10 Bit. Dadurch ergibt sich eine Datenrate von 125Mbit/s anstatt von 270Mbit/s. Die einzelnen Bilder werden danach noch mit dem JPG-Verfahren auf ein Fünftel der ursprünglichen Datenmenge komprimiert.

Dadurch ergibt sich die tatsächliche Datenrate von DV: 25Mbit/s.

([HENN2003] S. 103)

5.3 Digitale Kameras / Webcams

Heutzutage sind fast nur mehr digitale Videokameras in Verwendung. Die meisten von ihnen digitalisieren und komprimieren das produzierte Signal direkt – meistens im DV-Format. Für die Übertragung der aufgezeichneten Daten wird häufig die FireWire-Schnittstelle verwendet.

Eine spezielle Untergruppe von digitalen Videokameras sind so genannte Webcams, also Kameras, die speziell für die Verwendung mit PC und Internet entwickelt wurden. Diese sind im Vergleich zu vollwertigen Digitalkameras preisgünstig in der Anschaffung und verwenden einen USB-Anschluss zur Datenübertragung.

Die Bildqualität, vor allem bedingt durch eine ziemlich niedrige Auflösung (meist VGA, also 640x480 Pixel), ist meist nicht mit der von Digicams[↗] zu vergleichen.

5.3.1 Microsoft LifeCam VX-6000 vs. Logitech QuickCam Fusion

In diesem Abschnitt werden die zwei von uns am meisten verwendeten Webcams verglichen.

Tabelle 5.3-a Microsoft LifeCam VX-6000 vs. Logitech QuickCam Fusion

	Microsoft LifeCam VX-6000	Logitech QuickCam Fusion
max. Video-Auflösung	1280 x 960 (1,3 MegaPixel)	640x480 Pixel (VGA)
max. Foto-Auflösung	1,3 MegaPixel (physikalisch) 5 MegaPixel (interpoliert)	1,3 MegaPixel (physikalisch) 4 MegaPixel (interpoliert)
max. Bildrate	30 FPS	30 FPS
Bildsensor	CMOS	CMOS
Scharfstellung	manuell	x
Mikrofon eingebaut	ja	ja
Interface	USB 2.0	USB 2.0
Zoom	3fach digital (nur manuell mit Software)	x
Besonderheiten	71 Grad Weitwinkelobjektiv	„RightSound", „RightLight"
Preis	€ 80	€ 99

Glücklicherweise bekamen wir von unseren Sponsoren die Möglichkeit, einige Exemplare dieser zwei Produkte für unser Projekt zu verwenden und diese ausgiebig zu testen.

Aufgrund der besseren Leistungsmerkmale und des günstigeren Preises bevorzugen wir das Microsoft-Produkt. Die Bildqualität ist, trotz Rauschen, besser als die der Logitech-Kamera. Dieses Defizit macht die QuickCam Fusion aber mit der RightLight-Technologie von Logitech großteils wieder wett. Durch dieses Feature passt sich die QuickCam Fusion an die Helligkeit der Umgebung an und macht es so möglich auch in sehr dunklen Räumen noch etwas zu erkennen.

Ein weiterer Vorteil der Kamera von Microsoft ist, dass sie schneller auf Befehle von Software reagiert. Darauf wurden wir recht früh aufmerksam, da ja die Kamera die Auflösung erhöhen soll, sobald ein Stream[↗] für die Großansicht gebraucht wird. Diese Umschaltung kann bei der Logitech-Kamera durchaus bis zu zwei Sekunden dauern.

5.4 Kamera-Steuerung per ActionScript

In unserem Projekt ist es essenziell, Videomaterial von Kameras mit ActionScript innerhalb von Flash zu verarbeiten. Dabei sind mehrere Punkte zu beachten, die in diesem Kapitel beschrieben werden.

5.4.1 Allgemein

Die `Camera`-Klasse wird dazu verwendet, um den Videofeed, den eine an den Computer angeschlossene Kamera zur Verfügung stellt, in Flash zu verarbeiten. Dieser kann entweder direkt angezeigt oder per `NetStream` zum Flash Media Server übertragen werden. Diese Klasse wurde mit dem Flash Player 6 eingeführt und wird ab ActionScript 1.0 unterstützt.

Alle Informationen zur `Camera`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] verfügbar und ebendort entnommen.

5.4.2 Zugriffsschutz

Was man unbedingt beachten sollte ist, dass Benutzer im Flash Player den Zugriff auf angeschlossene Kameras manuell einschränken können. Es muss also immer damit gerechnet werden, dass per ActionScript kein Zugriff möglich ist. Des Weiteren sollte die Bühnengröße mindestens 215x138 Pixel betragen, da sonst das Einstellungsfenster des Flash Players nicht vollständig angezeigt werden kann.

Die Ereignisprozedur `Camera.onStatus` ist dazu da, um auf mögliche Zugriffsbeschränkungen dynamisch reagieren zu können. Sie wird aufgerufen, wenn der Benutzer den Zugriff auf die Kamera erlaubt oder verweigert.

5.4.3 Videofeed verwenden

Um den Videofeed einer angeschlossenen Kamera in Flash verwenden zu können, muss zu aller erst eine Instanz der `Camera`-Klasse erstellt werden.

```
Var my_cam:Camera = new Camera();
my_cam = Camera.get();
```

Mit der Methode `get()`, die hier ohne Parameter funktioniert, wird dem zuvor erstellten Objekt eine Referenz auf die Standardkamera zugewiesen. Soll eine andere Kamera als die Standardkamera verwendet werden, muss als Parameter der entsprechende Index angegeben werden.

Sehr wichtig ist, dass jede Kamera, die an einen Rechner angeschlossen ist nur einmal verwendet werden kann. Es ist also nicht möglich mit zwei Flash Player Instanzen gleichzeitig auf den Feed einer Kamera zuzugreifen. Auch wenn die Kamera bereits von einem anderen Programm verwendet wird, ist ein Zugriff nicht möglich.

Damit der Videofeed auf der Bühne dargestellt oder per `NetStream` versandt wird, ist je nachdem eine der unten stehenden Zeilen notwendig. Für die Darstellung auf der Bühne muss ein Video-Objekt vorhanden sein – in diesem Fall hat es den Bezeichner `my_video`. Zum Streamen werden eine `NetConnection`- und eine `NetStream`-Instanz benötigt.

```
My_video.attachVideo(my_cam);
my_netstream.attachVideo(my_cam);
```

5.4.4 Auflösung, Bildrate & Aufnahmequalität

Die wichtigsten Parameter, die beim Betrieb einer Kamera beachtet werden müssen, sind Auflösung und Bildrate – beide gemeinsam ergeben die Aufnahmequalität.

Für die Arbeit mit diesen Parametern gibt es einige Eigenschaften und Methoden der `Camera`-Klasse. Nachstehende Tabelle zeigt jene von ihnen, die von uns in V.watch am öftesten verwendet wurden.

Tabelle 5.4-a Methoden und Eigenschaften der `Camera`-Klasse in ActionScript ([ADOB2005d])

Methode/Eigenschaft	Beschreibung
Camera.bandwith:Number [schreibgeschützt]	Eine Ganzzahl, die angibt, welche Bandbreite (in Byte) für den momentan ausgehenden Videofeed maximal zur Verfügung steht. Wert kann mit `Camera.setQuality()` gesetzt werden.
Camera.currentFps:Number [schreibgeschützt]	Die Rate, mit der die Kamera momentan Daten erfasst (in Bildern pro Sekunde).
Camera.fps:Number [schreibgeschützt]	Die maximale Rate, mit der die Kamera Daten erfassen soll (in Bildern pro Sekunde). Wert kann mit `Camera.setMode()` gesetzt werden.
Camera.height:Number [schreibgeschützt]	Die aktuelle Aufnahmebildhöhe in Pixel. Wert kann mit `Camera.setMode()` gesetzt werden.
Camera.quality:Number [schreibgeschützt]	Eine Ganzzahl, die die erforderliche Bildqualität angibt, die sich aus dem Grad der Komprimierung jedes Videobilds ergibt. Wert kann mit `Camera.setQuality()` gesetzt werden.
Camera.width:Number [schreibgeschützt]	Die aktuelle Aufnahmebildbreite in Pixel. Wert kann mit `Camera.setMode()` gesetzt werden.
Camera.setMode ([width:Number], [height:Number], [fps:Number], [favorArea:Boolean])	Wählt den systemeigenen Kameraaufnahmemodus, in dem die ausgewählten Kriterien am besten erfüllt werden. Wenn die Kamera keinen systemeigenen Modus besitzt, in dem alle übergebenen Parameter erfüllt werden, wählt Flash einen Aufnahmemodus, der den gewünschten Kriterien am nächsten kommt. Bei Bedarf wird dazu das Bild auch zugeschnitten, und es werden Bilder übersprungen.
Camera.setQuality ([bandwidth:Number], [quality:Number])	Mit dieser Methode kann festgelegt werden, ob die Ausnutzung der Bandbreite oder die Bildqualität des ausgehenden Videofeeds für die Anwendung Vorrang haben soll.

Mehr Informationen zu den zwei hier angeführten Methoden und deren Anwendung in V.watch sind in *Kapitel 8.3, Netzauslastung vs. Streamqualität {CSI}* und *Kapitel 8.4.2.3, Anwendung der Qualitätslevels am Client {CSI}*zu finden.

5.4.5 Die Kamera als Bewegungsmelder

Flash bietet im Rahmen der `Camera`-Klasse auch die Möglichkeit, Kameras als Bewegungsmelder einzusetzen. Dies geschieht dadurch, dass der Aktivitätswert der Kamera, der zwischen 1 und 100 liegen kann, ausgelesen wird. Zusätzlich kann definiert werden, ab welchem Aktivitätswert der Handler `Camera.onActivity` aufgerufen werden soll. Darin kann auf Bewegungen – wie auch immer – reagiert werden.

Um dem Benutzer unseres Programms eine Rückmeldung zum Aktivitätslevel seiner Kamera zu geben, wird dieser auf Wunsch in einer Grafik rechts unten im Programmfenster angezeigt.

5.5 Videofelder in Flash

Videoobjekte bzw. –felder dienen dazu, um Videos, egal ob diese direkt von einer Kamera oder per `NetStream` ins Programm kommen, anzeigen zu können. Für mehr Komfort ist freilich mit der vorgefertigten `FLVPlayback`-Komponente gesorgt – diese ist aber eher als Player zum Abspielen von Videos gedacht und findet in unserem Programm keine Verwendung.

5.5.1 Erzeugung eines Videofeldes

Um ein Videofeld zu erstellen, genügt ein Rechtsklick auf die Titelleiste der Bibliothek und die Auswahl des Punktes „Neues Video…". Im erscheinenden PopUp muss der Punkt „Video (von ActionScript gesteuert)" als Typ angegeben und ein eindeutiger Bezeichner vergeben werden. Dieses Objekt kann nun aus der Bibliothek beliebig auf die Bühne gezogen werden.

5.5.2 Steuerung per ActionScript

Neben Eigenschaften wie X- und Y-Koordinate, Sichtbarkeit, Breite und Höhe und Rotation, die auch bei `MovieClips` verfügbar sind, haben Video-Objekte noch ein paar andere interessante Eigenschaften.

Unter anderem kann mit der Eigenschaft `Video.smoothing` angegeben werden, ob das angezeigte Video interpoliert, also geglättet werden soll. Mit der Methode `Video.clear()` kann das aktuell angezeigte Video aus dem Objekt entfernt werden, sodass dieses nicht ausgeblendet werden muss.

Die Anzeige eines Videos erfolgt mit folgender Methode:

```
public attachVideo(source:Object): Void
```

Die Quelle des Videos kann entweder ein `Camera`-Objekt oder ein `NetStream`-Objekt sein. Wenn die Anzeige beendet werden soll, kann `null` als Parameter übergeben werden. Dabei bleibt aber das letzte angezeigte Videobild auf der Bühne bestehen.

6 MIKROFONE & AUDIO {CSI}

Mikrofone werden verwendet, um Schallwellen in analoge elektromagnetische Impulse umzuwandeln. Das Signal, welches daraus resultiert, kann entweder direkt wiedergegeben, in analoger Form bearbeitet, gespeichert oder digitalisiert und danach erst weiterverarbeitet werden.

6.1 Mikrofontypen

Für verschiedene Einsatzgebiete gibt es verschiedene Arten von Mikrofonen mit unterschiedlichen Leistungsmerkmalen. Die am öftesten verwendeten Typen sind:

- Kondensatormikrofone (siehe *Kapitel 6.1.1*)
- dynamische Mikrofone (siehe *Kapitel 6.1.2*)
- Piezo-Kristall-Mikrofone (siehe *Kapitel 6.1.3*)
- Elektretkondensatormikrofone (siehe *Kapitel 6.1.4*)

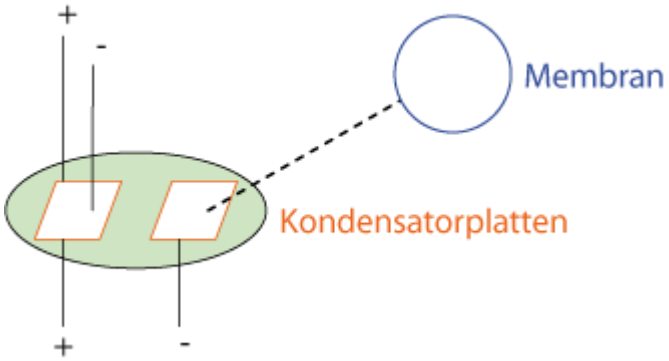
6.1.1 Kondensatormikrofon

Werden Luftmoleküle durch eine Stimme oder irgendeine andere Schallquelle in Bewegung versetzt, so bringt diese partielle Luftdruckveränderung die Membran im Kondensatormikrofon zum Schwingen. Die Membran ist mechanisch mit einer der Platten im Kondensator verbunden und gibt die Schwingung deshalb weiter. Dadurch wird wiederum das Dielektrikum, ein nicht leitendes Material, das die Kondensatorplatten galvanisch trennt, in Bewegung versetzt. Diese Reihe von Bewegungen verändert das magnetische Feld, das zwischen den beiden Kondensatorplatten besteht und den Ladezustand des Kondensators ausmacht. Ändert sich der Ladezustand, so ändert sich natürlich auch die Ausgangsspannung und somit das Signal.

Eine andere mögliche Bauform ist auch, dass die Membran selbst aus einem leitenden Material gefertigt ist und als Elektrode des Kondensators wirkt.

Da ein Kondensator ein passiver Bauteil ist, der zum Arbeiten eine externe Versorgungsspannung benötigt, braucht ein Kondensatormikrofon immer eine so genannte Phantomspannung (48V), um funktionieren zu können. Diese Spannung erzeugt das benötigte Spannungsgefälle zwischen den Kondensatorplatten.

Abbildung 6.1–a Schema Kondensatormikrofon



Dadurch, dass die Membran in Kondensatormikrofonen sehr dünn ist und eine geringe Masse hat, reagiert sie bereits auf kleinste Veränderungen des Schalldrucks. Dieses Verhalten führt zu besonders klaren Signalen, wodurch sich Kondensatormikrofone speziell für den Studiobereich eignen, wo sie auch am häufigsten eingesetzt werden. Je größer die Membran, desto empfindlicher ist sie und desto bessere Aufnahmen sind möglich. Die Empfindlichkeit von Mikrofonen wird generell in Millivolt pro Pascal (mV/Pa) gemessen. In Kondensatormikrofonen für den Studiobereich sind häufig Membrane mit 1 Zoll Durchmesser verbaut, die bis zu 100 mV/Pa liefern.

([BRUN2005] S. 128f)

Im Vergleich zu anderen Mikrofonbautypen sind Kondensatormikrofone recht teuer, was sich durch die Feinheit der Bauteile aber leicht erklären lässt.

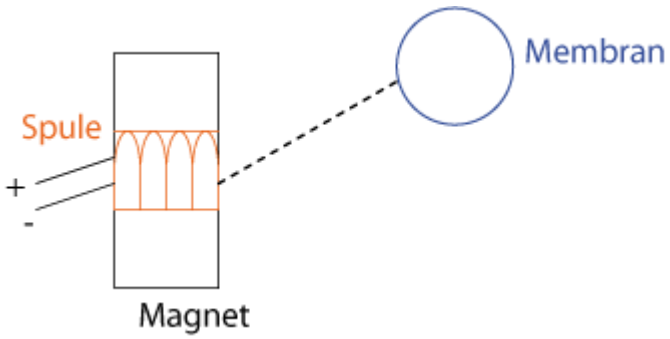
6.1.2 Dynamisches Mikrofon

Bei diesem Bautyp von Mikrofon bewegt die durch Schallwellen schwingende Membran keine Kondensatorplatte, sondern eine Spule, die von einem permanenten Magneten umgeben ist.

Durch diese Bewegung verändert sich die in die Spule indizierte Spannung. Diese Spannungsveränderungen ergeben das benötigte elektromagnetische Signal.

Im Gegensatz zu Kondensatormikrofonen beinhalten dynamische Mikrofone keine Bauteile, die eine Versorgungsspannung benötigen würden. Dadurch können sie ohne Phantomspannung von einem Vorverstärker betrieben werden.

Abbildung 6.1–b Schema dynamisches Mikrofon



Dynamische Mikrofone sind robuster als Kondensatormikrofone. Sie liefern deswegen zwar keinen so klaren Klang, sind aber besser für Live-Anwendungen nutzbar. Ihre Handhabung erleichtert sich durch den Wegfall einer Versorgungsspannung natürlich ungemein, da keine Vorverstärker benötigt werden.

([BRUN2005] S. 128f)

6.1.3 Piezo-Kristall-Mikrofon

In diesem Fall wird durch die Schwingung einer Membran eine Verformung eines so genannten Piezo-Kristalls bewirkt. Durch den daraus resultierenden piezoelektrischen Effekt, wird vom Kristall eine Spannung erzeugt, die als Audiosignal verwendet wird.

Diese Kristalle können aber nur in einem bestimmten Bereich schwingen und sind deshalb nicht für den gesamten, vom Menschen hörbaren, Frequenz-bereich nutzbar. Man verwendet sie aber z.B. als Gitarrentonabnehmer.

6.1.4 Elektretkondensatormikrofon

Elektretkondensatormikrofone funktionieren ähnlich wie die, fast baugleichen, Kondensatormikrofone. Der Unterschied dabei ist, dass sich bei diesem Bautyp kein Dielektrikum, sondern ein Elektret zwischen den Kondensatorplatten befindet. Dieses Material ist von Haus aus geladen – es ist also nur mehr eine sehr geringe Versorgungsspannung nötig, um das Mikrofon zu betreiben.

Aus dem Grund, dass man solche Mikrofone sehr klein und vor allem billig bauen kann und sie nur eine geringe Versorgungsspannung benötigen, werden sie fast ausschließlich in batteriebetriebenen Geräten, wie z.B. Handys, Videokameras und Notebooks, verwendet. Aus diesem Grund sind Elektretkondensatormikrofone auch die meist verbreiteten Mikrofone. Auch in den Webcams Microsoft LiveCam VX-6000 und Logitech QuickCam Fusion, die wir verwendet haben, sind Elektretkondensatormikrofone verbaut. Bei Computern liegt auf dem Mikrofoneingang eine Versorgungsspannung (geringe Gleichspannung mit ca. 2,5 Volt) an. Damit können Elektretkondensatormikrofone (z.B. in Headsets) bereits arbeiten.

Das Problem dabei ist aber, dass das Elektret nach einiger Zeit seine Ladung verliert, weshalb Elektretkondensatormikrofone schon nach ca. 15 Jahren unbrauchbar werden.

([BRUN2005] S. 128f)

6.2 Digitalisierung

Obwohl digitale Systeme aufgrund der weiten Verbreitung von Computern für den privaten Gebrauch immer beliebter werden, sind auch analoge Technologien immer noch oft im Einsatz – eine Kombination von beiden Bereichen ist also oft von Nöten.

Dieser Abschnitt beschreibt den Weg vom analogen zum digitalen Signal im Audibereich.

6.2.1 Analog vs. Digital

Analoge Signale zeichnen sich dadurch aus, dass sie zu jedem Zeitpunkt einen bestimmten Wert aufweisen. Sobald solche Signale digitalisiert werden, ist ein Qualitätsverlust nicht zu vermeiden, da nur zu bestimmten Zeitpunkten abgetastet und gemessen wird. Ein Teil der Information geht also verloren. Dieser Signalverlust wird „Quantisierungsrauschen" genannt.

Dafür bieten analoge Signale aber einige Vorteile bei Speicherung und Bearbeitung. Bei analogen Daten nähern sich Störsignale und ursprüngliches Signal mit der Zeit immer mehr an, bis schlussendlich keine Unterscheidung mehr möglich ist. Störsignale auf digitalen Signalen machen bei Weitem nicht so viel aus, da nur eine Unterscheidung zwischen 0 und 1 von Nöten ist.

([BRUN2005] S. 195)

6.2.2 Samplefrequenz & Messtiefe

Beim Digitalisieren von analogen Audiosignalen gibt es zwei wichtige Parameter: Die Samplefrequenz und die Auflösung oder Messtiefe. Analoge Spannungen werden dabei zeit- und wertdiskret gemacht.

Die Samplefrequenz gibt an, wie oft das analoge Signal abgetastet und gemessen wird. Sie sollte immer mindestens ein bisschen höher als die maximale Signalfrequenz sein, damit der Abtastpunkt immer ein bisschen verschoben wird. Wäre das nicht der Fall und würde ein Signal z.B. ein Sinus-Signal mit 22kHz mit 44kHz abgetastet werden, so wäre es theoretisch möglich, dass der Abtastpunkt immer im Nullpunkt des Signals liegt – das Ergebnis nach der Digitalisierung wäre somit eine Nulllinie.

([BRUN2005] S. 18)

Allgemein wird diese Anforderung als „Nyquist-Shannon Abtasttheorem" bezeichnet und hat folgende Formel:

$$f_{abtast} > 2\,(f_{max} - f_{min})$$

Auflösung bedeutet, wie genau das Signal an jedem Abtastpunkt gemessen und gespeichert werden soll. 16 Bit Messtiefe bzw. Auflösung ergeben also 65536 Stufen, die der abgetastete Wert annehmen kann. ([BRUN2005] S. 18)

Derzeit sind folgende Standards in Verwendung:

Tabelle 6.2-a Audio-Standards bei Digitalisierung ([HENN2003] S. 132)

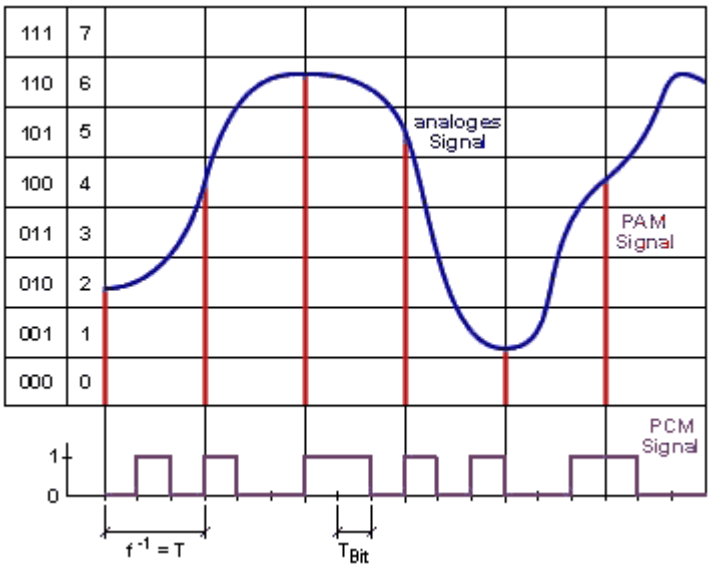
Anwendungsbereich	Samplefrequenz	Bittiefe
Audio CD (Compact Disc)	44,1 kHz	16 Bit
DAT-Tape	48 kHz	16 Bit
digitales Bühnenequipment (Mischpult, Effektprozessor, etc.)	96 kHz	24 Bit

6.2.3 PCM – Pulse Code Modulation

Das Verfahren, bei dem analoge Audiodaten digitalisiert, also abgetastet und gemessen, werden, heißt „Pulse Code Modulation", kurz PCM. PCM-Codierte Daten werden als unkomprimierte WAVE-Dateien gespeichert.

Abbildung 6.2–a zeigt das PCM-Verfahren mit einer Messtiefe von 3 Bit, was 8 verschiedene Werte möglich macht.

Abbildung 6.2–a Pulse Code Modulation
(http://upload.wikimedia.org/wikipedia/de/c/cc/Pcm.png)



Übersteigt die Amplitude des analogen Signals die Grenzen der Quantisierung, so wird von „digitalem Clipping" gesprochen. An den Stellen der Überschreitung wird der höchst mögliche Abtastwert gespeichert.

6.3 Mikrofonsteuerung per ActionScript

Mit der `Microphone`-Klasse können Audiosignale von einem Mikrofon, das an einen Computer, auf dem der Flash Player ausgeführt wird, angeschlossen ist, erfasst werden. Sie kann daher als Audio-Pendant zur `Camera`-Klasse gesehen werden. Die Klasse wird ab Flash Player 6 und ActionScript 1.0 unterstützt.

Alle Informationen zur `Microphone`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] verfügbar und ebendort entnommen.

6.3.1 Zugriffsschutz

Ähnlich wie bei Kameras, kann der Benutzer auch den Zugriff auf Mikrofone beschränken. Reaktionen auf mögliche Beschränkungen können auch hier im `onStatus`-Handler platziert werden. Nähere Informationen hierzu sind in *Kapitel 5.4.2, Zugriffsschutz* zu finden.

6.3.2 Audiodaten verwenden

Um das Signal eines angeschlossenen Mikrofons in Flash verwenden zu können, muss zu aller erst eine Instanz der `Microphone`-Klasse erstellt werden.

```
Var my_mic:Microphone = new Microphone();
my_mic = Microphone.get();
```

Mit der Methode `get()` wird das `Microphone`-Objekt mit dem Standardmikrofon verknüpft und repräsentiert dieses ab sofort. Sind mehrere Mikrofone angeschlossen und soll ein anderes als das erste in der Liste verwendet werden, so muss dessen Index als Parameter der `get()`-Methode angegeben werden.

Im Unterschied zur `Camera`-Klasse können hier aber auch mehrere Objekte auf die Daten eines einzigen Mikrofons zugreifen. Auch mehrere Flash Player Instanzen können sich so ein Gerät teilen.

Damit eine Aufnahme begonnen oder die Audiodaten per `NetStream` versandt werden können, muss die Audioquelle, also das `Microphone`-

Objekt, erst noch mit einem `MovieClip` oder einer `NetStream`-Instanz verknüpft werden.

```
My_mc.attachAudio(my_mic);
my_netstream.attachAudio(my_mic);
```

Der `MovieClip` `my_mc` muss sich dabei keinesfalls auf der Bühne befinden – es kann auch eine mit ActionScript dynamisch erstelltes Instanz sein.

6.3.3 Qualitätseinstellungen

Gegenüber der `Camera`-Klasse sind die Möglichkeiten zur Regelung der Qualität von Audiodaten eher gering. In Wahrheit gibt es nur eine Eigenschaft, die verwendet werden kann, um das Signal wunschgemäß anzupassen: `Microphone.rate`.

Der Wert dieser Eigenschaft gibt an, mit welcher Abtastfrequenz das Mikrofon arbeiten soll. Zulässige Werte sind 5, 8, 11, 22, und 44 kHz – 8 kHz ist der Standardwert, sofern das Mikrofon diesen Wert unterstützt. Ist das nicht der Fall, wird die nächst höhere Frequenz angenommen.

Da die `rate`-Eigenschaft schreibgeschützt ist, muss die Methode `setRate()` benutzt werden, um einen neuen Wert zu setzen:

```
my_mic.setRate(44);
trace (my_mic.rate);
```

Natürlich sollte beachtet werden, dass mit einer hohen Abtastfrequenz auch mehr Bandbreite beim Streaming benötigt wird. In unserem Projekt setzen wir ganz auf Qualität und benutzen eine Abtastfrequenz von 44 kHz. Werden 44 kHz von der Hardware nicht unterstützt, wird der nächste verfügbare Wert darunter verwendet.

Zusätzlich zur Abtastfrequenz steuern wir auch die Pegelhöhe, also die Lautstärke des Signals.

Mit der Methode `setGain()` kann das schreibgeschützte Attribut `gain` der `Microphone`-Klasse gesetzt werden. Dieses kann Werte zwischen 0 und 100 annehmen und gibt an, um welchen Faktor das Mikrofon das aufgenommene

Signal verstärken soll. Ein Wert von 0 bedeutet demnach, dass das Mikrofon überhaupt kein Signal überträgt. 50 ist der Standardwert und bedeutet keine Verstärkung, aber auch keine Abschwächung. Da integrierte Mikrofone, wie sie z.B. in Webcams oft verwendet werden, einen relativ geringen Signalpegel liefern, wird dieser in V.watch mit dem Befehl `setGain(70)` ein wenig angehoben.

6.4 Arbeiten mit Sound in ActionScript

Neben der `Microphone`-Klasse, die zum Interagieren mit Audio-Hardware benötigt wird, gibt es in ActionScript auch die `Sound`-Klasse, mit der Audiodaten in Flash-Filmen gesteuert werden können. Diese Klasse ist seit dem Flash Player 5 mit ActionScript 1.0 nutzbar.

Alle Informationen zur `Sound`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] verfügbar und ebendort entnommen.

Während unserer Arbeit an V.watch haben wir festgestellt, dass die `Sound`-Klasse zwar sehr mächtig, aber eher für die Arbeit mit bereits aufgezeichneten und in der Bibliothek verfügbaren Audiodaten gedacht ist. So lassen sich mit Instanzen der Klasse, unter anderem, Sounds starten und stoppen, die Abspielposition in der Datei ändern, Lautstärken- und Balance-Änderungen durchführen und ID3-Tags[↗] auslesen.

Dadurch, dass wir in unserem Projekt aber ausschließlich mit Live-Audio-Daten arbeiten, war die Verwendung von `Sound`-Objekten nicht notwendig.

7 DESIGN & USABILITY {SEI}

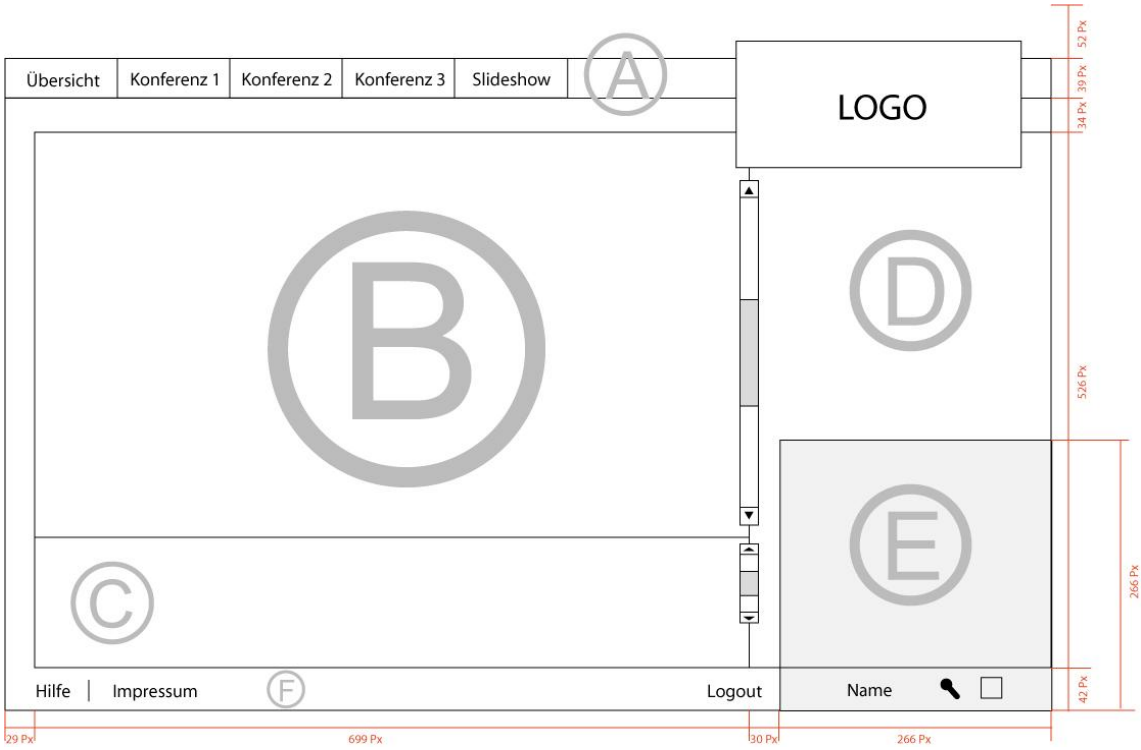
Um V.watch eine ansprechende, beruhigende, schlichte, aber trotzdem funktionell passende Oberfläche zu geben, mussten wir einige Design-vorschläge entwerfen, und uns danach für einen davon entscheiden. Bei diesen Designs haben wir uns eng an das Design der Management Konsole des Flash Media Servers (siehe *Kapitel 3.10.6, Management-Konsole*) gehalten.

7.1 Layout

Im folgenden Punkt werden der Entwurf und die Abweichungen zu den tatsächlich verwendeten Layouts analysiert. Außerdem wird aufgezeigt, warum manche Aspekte des am Ende verwendeten Layouts, besser auf den Benutzer wirken, und warum uns manche Einschränkungen durch Flash, bezüglich der Realisierung unseres Entwurfs, zu Veränderungen zwangen.

Grundsätzlich und auch auf die Entscheidung aufbauend, dass wir ein Reitermenü als Navigation verwenden wollten, kam für uns nur ein Layout in Frage. Also baute sich das Grundlayout, das die Struktur der Oberfläche beschreibt, in seinem ersten Entwurf wie folgt auf:

Abbildung 7.1–a 1. Entwurf des Layouts der V.watch-Programmoberfläche



Wie man der Grafik entnehmen kann, war es geplant, die Struktur des Programms in klare Teile aufzuteilen.

7.1.1 Bereich A

Dieser Bereich ist ein Balken, in dem das Reitermenü platziert werden sollte. Er sollte sich klar von den anderen Teilen, die zusammenhängend unter dem Reitermenü liegen, distanzieren. Durch einen kleinen Zwischenraum zwischen Balken und Hauptteil des Programms, wollten wir diese Distanz herstellen. Die einzige Verbindung der beiden Teile, wäre das Logo, das in der Grafik auf der rechten Seite zu finden ist, gewesen.

7.1.2 Bereich B

Der Teil, den wir als Hauptbereich definiert haben, sollte die Thumbnails[↗] der im System befindlichen Benutzer beinhalten. Außerdem war es im ersten Entwurf geplant eine Bildlaufleiste in diesen Bereich einzubauen, das heißt, dass man mit einem schmalen Balken am rechten Bereichsrand sich im Bereich hinauf und hinunter bewegen kann.

7.1.3 Bereich C

Direkt unter dem Hauptbereich sollte sich der Interaktionsbereich befinden. In ihm sollten `Buttons` und andere Eingabekomponenten angeordnet werden. Somit sollte gewährleistet werden, dass alle Interaktionsmöglichkeiten in einem Bereich zusammengefasst sind.

7.1.4 Bereich D

Im Entwurf des Layouts haben wir diesen Bereich als zweiten Interaktionsbereich definiert. Er sollte in den verschiedenen Reitern, verschiedene Zwecke erfüllen. In der Konferenz sollte er als Platz für die Teilnehmerliste, und in der Übersicht als Feld, indem Alarmmeldungen angezeigt werden sollten, dienen. In der Slideshow und in der Großansicht sollte er Interaktionselemente beinhalten.

7.1.5 Bereich E

Dieser Bereich sollte auf jeder Seite im Programm zu sehen sein. Es war geplant in diesem Bereich das Videofeld des Benutzers zu platzieren. Außerdem sollten darunter auch Informationen zum Benutzer angezeigt werden.

7.1.6 Bereich F

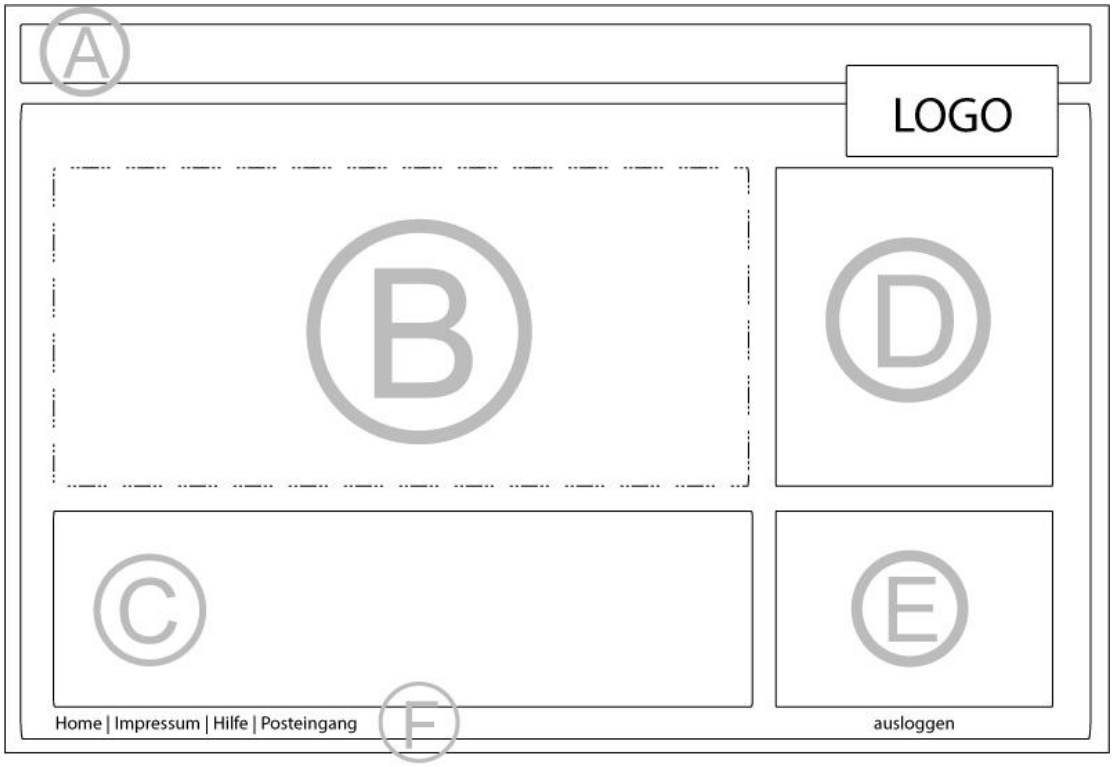
Die Fußzeile, die unter dem Interaktionsbereich C liegen sollte, wurde als Bereich für weniger wichtige Navigationspunkte definiert. Hier sollten unter anderem die Hilfe und das Impressum ihren Platz finden.

7.2 Änderungen im tatsächlich verwendeten Layout

Wie man der folgenden Abbildung entnehmen kann, haben sich einige Dinge, die ursprünglich geplant waren, während des Designprozesses geändert. Gründe für diese Veränderungen waren vor allem Erkenntnisse, die wir im Laufe des Projektes erworben haben, gemeinsame Überlegungen, die uns auf Nachteile des Entwurfes gebracht haben und spätere Unstimmigkeiten mit dem Design.

Im folgenden Abschnitt werden die Bereiche aus *Kapitel 7.1, Layout*, mit den Abweichungen zum tatsächlich verwendeten Layout, erweitert. Außerdem werden Gründe für diese Änderungen näher beschrieben und analysiert.

Abbildung 7.2–a Endgültige Anordnung der V.watch-Programmoberfläche



7.2.1 Ad Bereich A

Das Reitermenü wurde, wie geplant, durchgesetzt und realisiert. Es setzt sich durch einen kleinen Abstand vom Hauptbereich ab und wird einzig und allein durch das Logo mit ihm verbunden.

7.2.2 Ad Bereich B

Der Hauptbereich, in dem wie geplant alle User angezeigt werden, die im System angemeldet sind, hat sich gegenüber dem Entwurf ein wenig verändert. Durch eine unvorteilhafte Handhabung mit Bildlaufleisten in Flash, haben wir uns dazu entschlossen, diese Bereiche durch Seiten, die durchgeblättert werden können, zu ersetzen. Es wurde festgelegt, jeweils acht (in Konferenzen, sechs) kleine Videofenster auf einer Seite zu platzieren.

7.2.3 Ad Bereich C

Dieser Teil des Programms ist der Interaktionsbereich geblieben, da wir gewährleisten wollten, dass alle Möglichkeiten für den Benutzer, etwas zu tun, gemeinsam auf einem Punkt zusammenkommen. In der Übersicht von V.watch ist dieser Teil nochmals in einen schmalen und einen etwas größeren Balken geteilt. Der obere, schmale, Bereich ist zur Navigation zwischen den Thumbnails[↗], und der untere, dickere Bereich für die eigentlichen Interaktionen nötig. In der Konferenz wird dieser, wieder einheitliche, Bereich für den Text-Chat verwendet.

7.2.4 Ad Bereich D

Da die Realisierung des software-basierten Alarmsystems durch eine Hardware-Lösung ersetzt wurde, die außerhalb der Diplomarbeit V.watch entwickelt wurde, bekam der Bereich D in der Übersicht die Aufgabe konferenzbezogene Interaktionsmöglichkeiten in sich zu tragen. Hier sahen wir einen Vorteil darin, dass der Benutzer eine klare Trennung zwischen übersichts- und konferenzbezogenen Elementen hat, und sich somit besser zurechtfinden wird. Auch in der Konferenz selbst sind hier Schaltflächen und die Konferenzteilnehmerliste angebracht. In der Slideshow und in der Großansicht werden wichtige Benutzerdaten darin angezeigt.

7.2.5 Ad Bereich E

Das Video der eigenen Kamera ist nun noch strukturierter von den anderen Bereichen abgetrennt. Es wurde vergrößert, um es somit auffälliger zu gestalten. Außerdem kann sich der Benutzer das eigene Video besser einrichten. Die Interaktionsbereiche für die eigene Kamera, sind im selben Bereich untergebracht, jedoch hinter einer ausfahrbaren Oberfläche. Wie dieser Bereich und auch alle anderen, mit Grafiken und Farbe im Endprodukt aussehen, wird im nächsten Punkt, den Designvorschlägen näher beschrieben.

7.2.6 Ad Bereich F

Prinzipiell hat sich die Fußzeile nur inhaltlich verändert. Im Layout hat sie ihre Position und Aufgabe behalten. Sie befindet sich unterhalb des Interaktionsbereiches und der eigenen Kamera und dient dazu, sekundäre Navigationsbereiche abzudecken. Diese wurden durch einen „Home" und einen „Private Nachrichten" Verweis erweitert. Mit diesen neuen Menüpunkten hat der Benutzer die Möglichkeit zum Willkommensbildschirm zu wechseln, oder seinen Posteingang bzw. das komplette „Private Messaging System" abzurufen.

7.3 Designvorschläge

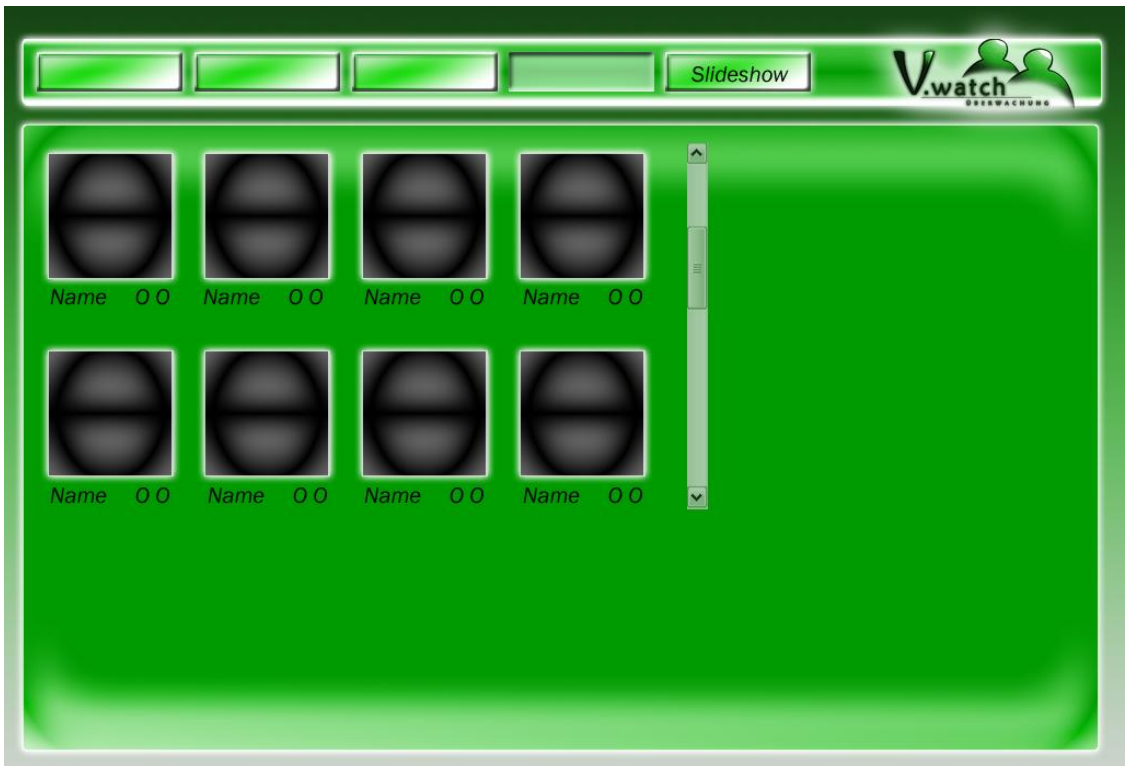
Nachdem wir unseren ersten Layoutentwurf hatten, mussten wir eine passende grafische Oberfläche dazu finden. Im Zuge dieser Suche sind wir auf einige verschiedene Vorschläge dafür gestoßen. Im folgenden Abschnitt werden diese aufgeführt und analysiert. Am Ende dieses Abschnittes werden das aktuelle Design und seine Abgrenzungen zu den nicht genommenen Entwürfen beschrieben.

Außerdem werden alle Designs der Module von V.watch, die ihr Aussehen nicht aus dem Grunddesign ziehen (Login, Registrieren und Überwachungsmodus), anhand von Screenshots[↗] betrachtet.

7.3.1 Vorschlag 1

In diesem ersten Entwurf haben wir einige Experimente mit Glanz und Spiegelungen in Adobe's Photoshop[↗] gemacht. Wie man bemerkt, glänzt der komplette Bildschirm ein wenig zu viel. Dies war auch der Grund, warum wir uns gegen diesen Entwurf entschieden haben. Grundsätzlich hat uns die Anordnung nach der Vorlage des Layouts gut gefallen, wobei auch das Reitermenü gut sichtbar vom Hauptbereich getrennt war. Auch die Thumbnails[↗] waren gut sichtbar, jedoch wirkte dieses Design etwas erdrückend, aufgrund seiner übertriebenen Spiegelungen.

Abbildung 7.3–a Erster Designvorschlag für V.watch



7.3.2 Vorschlag 2

Dieser Vorschlag war, aufgrund der bereits ausgereiften Kenntnisse mit Glanz und Spiegelungen in Photoshop[↗], schlicht und gut strukturiert gehalten. Wir wählten hier eine etwas andere Methode, um das Reitermenü zu gestalten. Es ging in diesem Entwurf, im Gegensatz zum Hauptbereich, über die komplette Breite des Bildschirmes und sogar darüber hinaus. Aufgrund des dunklen Hintergrunds wurde das Logo mit einem weißen „Schein nach außen"-Effekt (siehe *Abbildung 7.5–c*) versehen. Doch genau wegen dieses dunklen Hintergrunds blieb dieser Entwurf lediglich ein Vorschlag. Wir dachten er würde nicht freundlich auf den Benutzer wirken und ihn somit eventuell abschrecken.

Abbildung 7.3–b Zweiter Designvorschlag für V.watch

7.3.3 Vorschlag 3

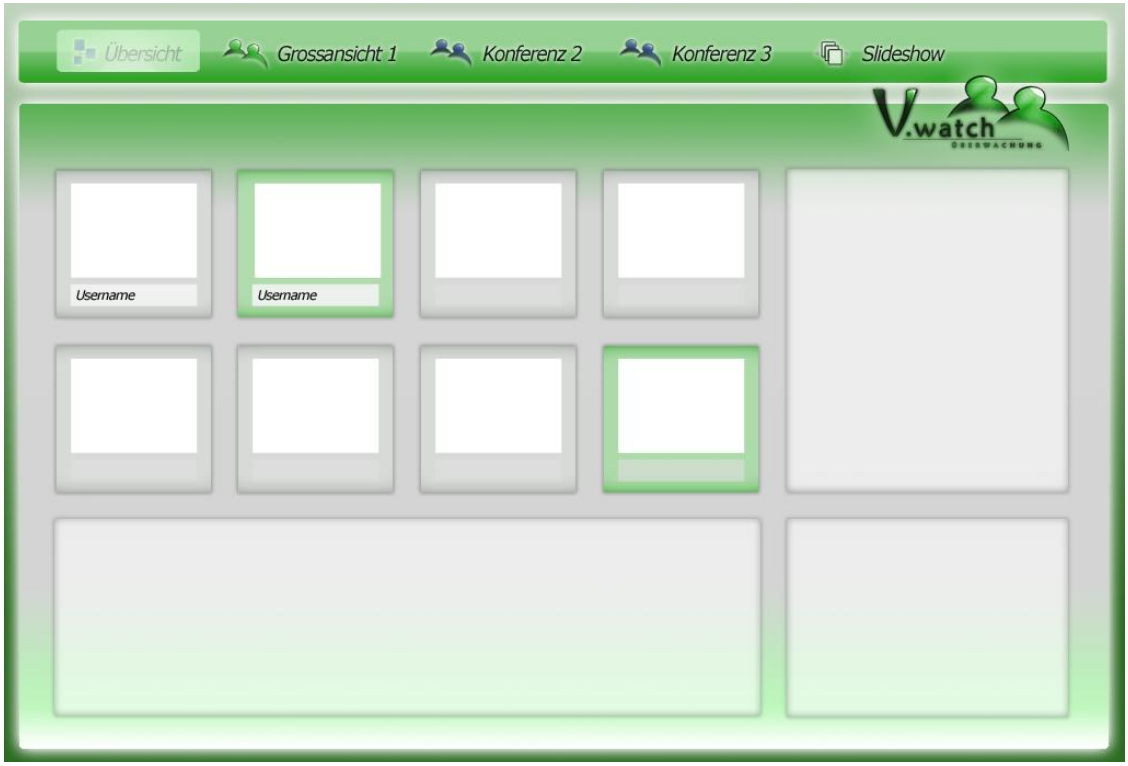
Bei diesem Entwurf konnten wir die bisherigen Erkenntnisse kombinieren. Wir kreierten eine zarte Glanzoberfläche und arbeiteten mit helleren Farben. Das Logo erhielt einen etwas zarteren Schein als bei Vorschlag 2 und stach damit nicht sofort ins Auge. Dies war von Vorteil, da man sich eher auf das Programm konzentrieren sollte als auf das Logo. Wir kehrten auch hierbei wieder zu unserem alten Entwurf zurück und richteten die Leiste für das Reitermenü so aus, dass es nicht bis zum Rand reichte. Außerdem versuchten wir schon im Design, im Hauptbereich eine Trennlinie für den Interaktionsbereich (siehe *Kapitel 7.2.3, Ad Bereich C*) zu schaffen. Das einzige Manko, das der „Vorschlag 3" noch mit sich brachte war, dass zu viel Grün darin enthalten war.

Abbildung 7.3–c Dritter Designvorschlag für V.watch

7.3.4 Grunddesign

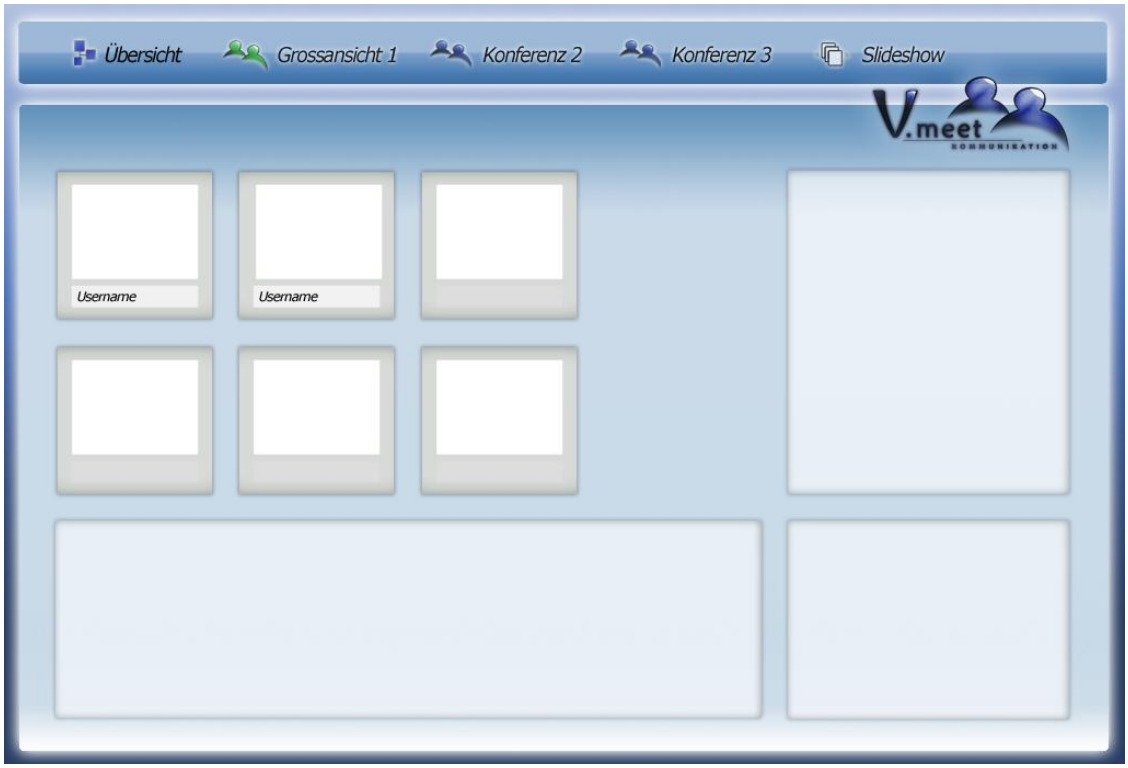
Der vierte Vorschlag, der eine Verbesserung des dritten Entwurfs war, überzeugte uns so sehr, dass er von uns als V.watch Design ausgewählt wurde. Es wurde vor allem der Hintergrund des Hauptbereiches geändert. Das viele Grün wurde durch zartes Grau, mit wenigen grünen Akzenten, ersetzt. Außerdem wurde eine klarere Strukturierung der einzelnen Bereiche, durch weiße, leicht transparente Rechtecke gewährleistet. Nachdem wir uns für dieses Design entschieden hatten, wurden die entsprechenden Felder für die Thumbnails[↗] entwickelt. Ansonsten wurden alle Elemente aus dem „Vorschlag 3" übernommen, wobei sich das Logo noch um einige Pixel[↗] verschoben hat.

Abbildung 7.3–d Grunddesign von V.watch



Nachdem wir unser Design gefunden hatten, mussten wir es nun auch für Konferenzen also V.meet generieren. Hierzu verwendeten wir die Farbton/Sättigung-Funktion von Photoshop[↗], womit wir die einzelnen Bereiche einfach umfärbten. Das fertige Ergebnis sieht folgendermaßen aus.

Abbildung 7.3–e Grunddesign von V.meet

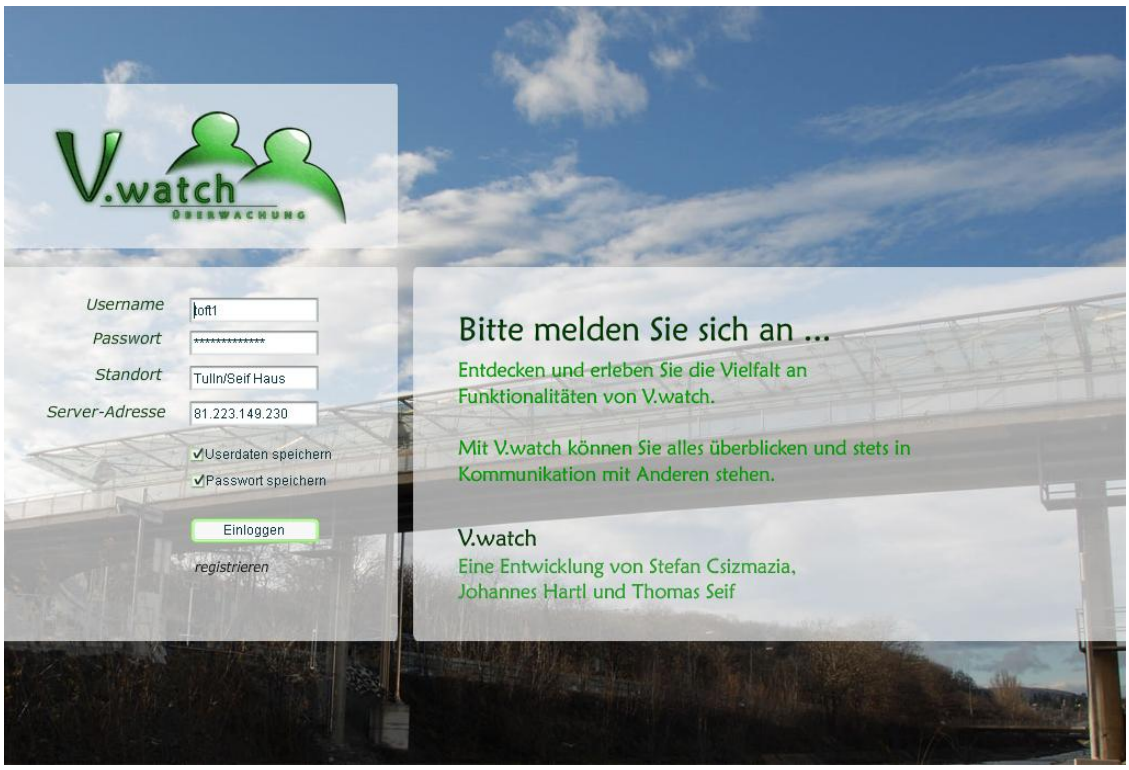


In dem Design von V.meet ist außerdem zu beachten, dass nur maximal sechs Benutzer angezeigt werden können. Prinzipiell sind sieben Teilnehmer in einer Konferenz erlaubt, jedoch wird das eigene Video nicht angezeigt.

7.3.5 Design des Login-Bildschirmes

Um dem Benutzer den ersten Blick auf V.watch, also den Login-Bereich, sehr ansehnlich zu gestalten, mussten wir uns hierfür etwas Ausgefallenes überlegen. Wir dachten von Anfang an, dass ein Foto im Hintergrund des Login-Bereichs diese Ausgefallenheit erzielen würde. Wir wollten V.watch mit diesem Foto außerdem ein Motto zuordnen. Wir überlegten, was am besten zu den Wörtern Sicherheit, Überblick und Verbindung passen und zusätzlich noch gut aussehen würde. Nach langem Überlegen wurde uns klar, dass das Richtige dafür eine Brücke sei. Wir wussten, dass es bei der U-Bahn Station „Hütteldorf" im 14. Wiener Gemeindebezirk eine moderne Fußgängerbrücke gibt und schossen einige Fotos von dieser. Danach entwerteten wir diese und filterten das Beste davon heraus, was nun nach einigen Bearbeitungen in Photoshop[↗] unseren Login-Bereich ziert. Durch weiße, halb transparente Flächen gewährleisteten wir uns einen Bereich auf dem wir etwas anzeigen konnten. Das fertige Design des Anmeldebildschirms sieht wie folgt aus.

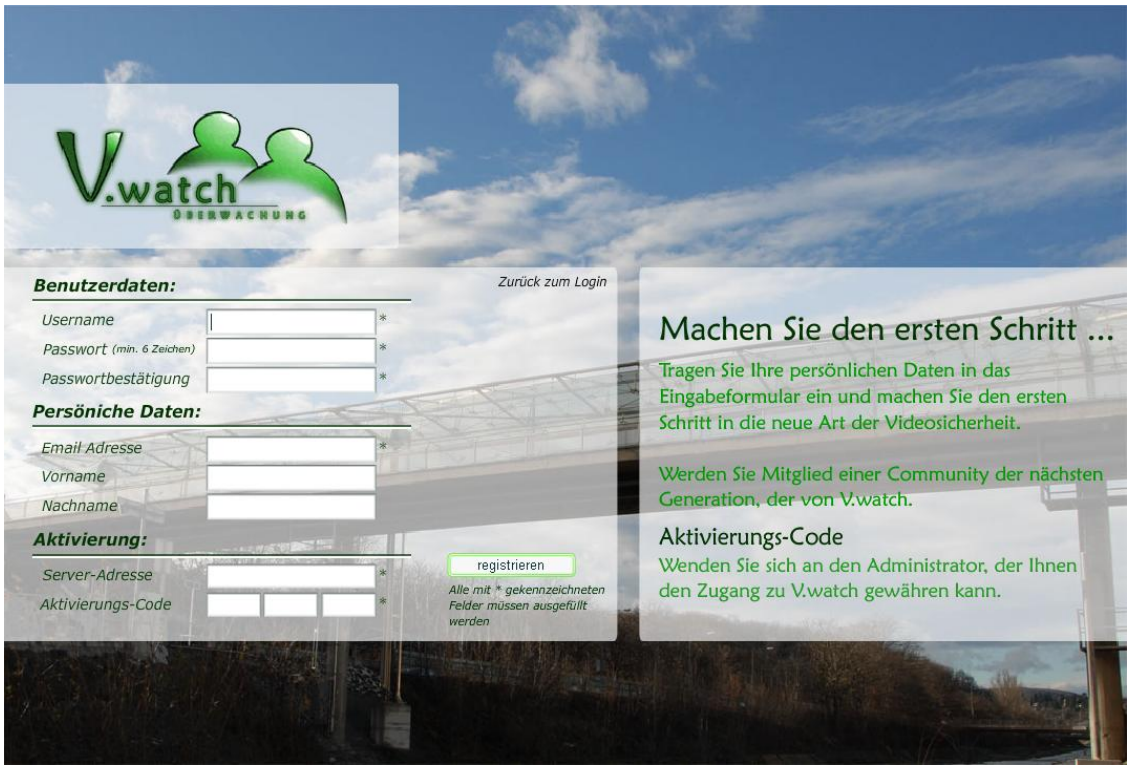
Abbildung 7.3–f Fertiges Design des Login-Bereichs



7.3.6 Design des Registrierungsbildschirmes

Weil die Registrierung eng mit dem Einloggen in das System zusammenhängt, haben wir ein ähnliches Design gewählt. Es sind lediglich die Größen der weißen, halb transparenten Flächen verändert worden. Die Anordnung der Formularelemente wurde ebenso ansehnlich und übersichtlich gestaltet.

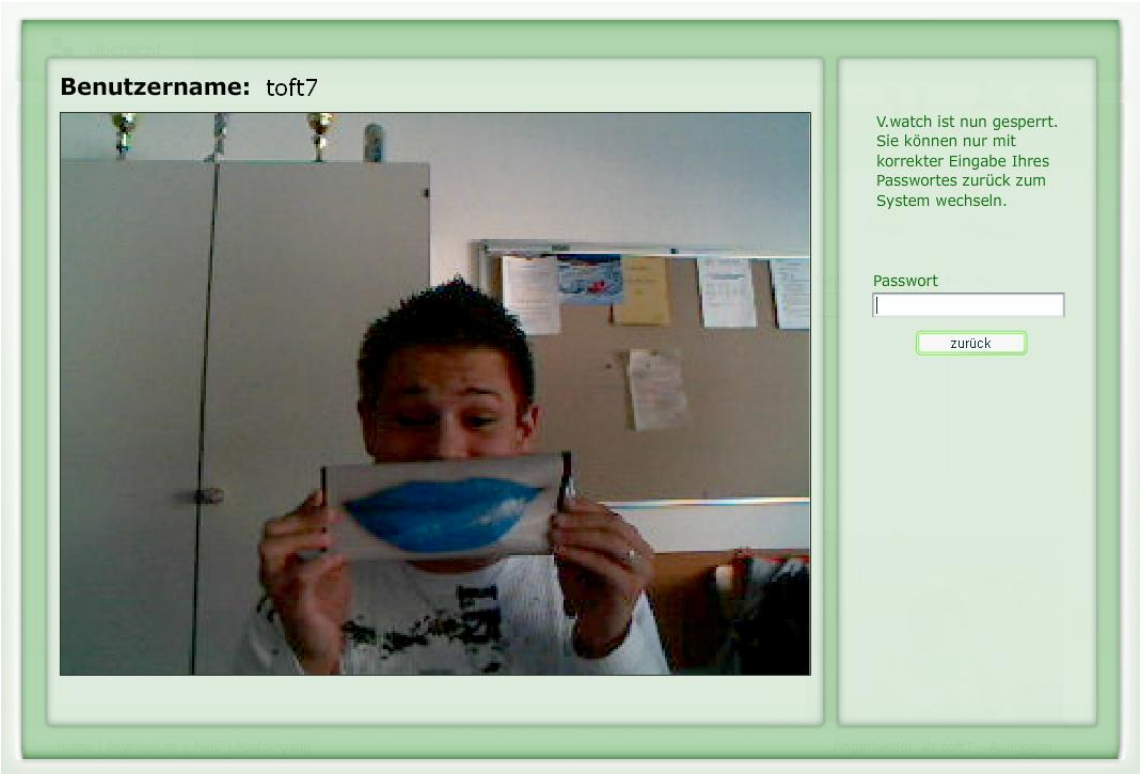
Abbildung 7.3–g Fertiges Design des Registrierungsbildschirmes



7.3.7 Design des Überwachungsmodus

Das Grundlayout mit den einzelnen Bereichen konnte für den Überwachungsmodus nicht verwendet werden, da es sich hier um einen eigenen Bildschirm, der sich über das V.watch-Fenster ausspannt, handelt. Dieser ist vergleichbar mit der Sperrung des PCs unter Windows. Wir haben nun anhand des Grunddesigns eine Lösung für diesen Modus gefunden. Auch hier finden die klaren Abgrenzungen aus dem herkömmlichen Erscheinungsbild des Programms Verwendung. Dieses Design sieht folglich so aus:

Abbildung 7.3–h Design des Überwachungsmodus



7.4 Farben

Wie man schon den Designvorschlägen entnehmen konnte, gibt es für die zwei Hauptteile unserer Diplomarbeit – V.watch und V.meet – unterschiedliche Farbkombinationen. Im folgenden Abschnitt werden diese Farben aufgezeigt und ihre Bedeutung sowie Einsatzbereiche näher beschrieben.

7.4.1 Grün

Wir haben uns bei unserer Software für die Farbe Grün entschieden. In allen Modulen, die nicht mit der Konferenz zu tun haben, die mit dem Überbegriff V.watch zusammengefasst werden können, herrschen vor allem Grün-Töne vor. Kombiniert mit grauen Aspekten prägt das Grün auch das Grunddesign.

Da das Thema der Überwachung, womit unsere Software hauptsächlich zu tun hat, von Anfang an ein ziemlich heikles war, suchten wir eine auf den Benutzer beruhigend wirkende Farbe. Die Farbe Grün hat genau diese Wirkung. Außerdem ist sie angenehm für die Augen und kann somit den Ausblick auf das Wesentliche freihalten. Außerdem kann Grün auch die Sicherheit signalisieren. Wenn man zum Beispiel eine Ampel betrachtet, wissen die Menschen, dass sie bei Grün sicher über die Straße gelangen. Diese Sicherheit soll V.watch ebenfalls repräsentieren. ([SEIL1998a])

7.4.2 Blau

Für die Erweiterung zur Überwachung, also der Konferenz, haben wir uns für die Farbe Blau entschieden. Grund für diese Wahl der Gestaltung von V.meet war vor allem die Abhängigkeit von Grün, da diese aus den Farben Blau und Gelb besteht. Außerdem beschreibt die Farbe Blau auch das Ferne, was damit assoziiert werden kann, dass man via V.meet mit Menschen in der Ferne kommunizieren kann. Zusätzlich wirkt auch die Farbe blau beruhigend und sie stimmt positiv. Auch hier gibt es ein Beispiel aus dem Alltagsleben. Blaue Briefe zum Beispiel, die meistens Versetzungen oder auch Einberufungen beinhalten, sollen durch ihre Farbe besser entgegengenommen werden. ([SEIL1998b])

7.5 Logo

Ein sehr wichtiger Punkt für eine Software ist natürlich das Logo. Ein Logo ist ein Symbol, das ein Unternehmen oder in diesem Fall ein Programm repräsentiert. Es sollte an die Zielgruppe angepasst sein und einen hohen Wiedererkennungswert haben. ([GRAF2005])

Im folgenden Abschnitt werden die einzelnen Entwicklungsschritte, bis hin zum fertigen Logo unserer Software, beschrieben. Dies wird anhand des V.meet-Logos geschehen, da davon noch Grundrisse auf dem Papier vorhanden sind.

7.5.1 Vorzeichnen auf Papier

Um das Logo, das zumeist zuerst im Kopf entsteht, in die Wirklichkeit umzusetzen, hilft zuerst ein Entwurf, den man mit Papier und Bleistift durchführt. So war es auch bei unserer Arbeit.

Abbildung 7.5–a Logoentwurf auf Papier

Angefangen hat alles mit der Vision eines Schriftzuges (V.meet) und einer Grafik rechts neben diesem. Wir versuchten eine Grafik zu finden, die zu unserem Produkt passen könnte und sie dabei auf Silhouetten *(=Umrisse)* von zwei Menschen gekommen. Diese Personen sind deswegen ein gutes Zeichen, da unsere Software vor allem Menschen, die gerne mit anderen kommunizierem, ansprechen soll. Außerdem war es uns wichtig, das „V" auffallend zu gestalten, da die Namen V.watch und V.meet für mehrere Bedeutungen, aufbauend auf das „V" stehen.

Zum einen steht dieser Buchstabe für „Video" und zum anderen für „We" (engl. für „wir"), um die Vereinigung der Menschen via V.watch zu symbolisieren. Um dem Benutzer klar zu machen, wobei es sich, in diesem Fall um V.meet, handelt, haben wir unter den eigentlichen Schriftzug, getrennt durch eine Linie, noch das Wörtchen Kommunikation gesetzt.

7.5.2 Übertragung auf den PC

Nun mussten wir diese Zeichnung möglichst originalgetreu auf den Computer übertragen. Dazu fotografierten wir die Zeichnung und pausten sie in Adobe Illustrator[↗] ab. Dabei wurden auch die Kurven genauer und die Linien gerader gezeichnet. Nach dem vollständigem Abzeichnen der Grafik und der Verschönerung durch einige Effekte in Adobe Photoshop[↗] sah unser erster digitaler Entwurf wie folgt aus.

Abbildung 7.5–b Erster Logoentwurf am PC

Hier haben wir der Grafik einen gewissen Schein nach außen hinzugefügt und versucht ihr damit mehr Ausdruck zu verleihen. Dieses Verfahren kann mit dem „Schein nach außen"-Effekt von Adobe Photoshop[↗], der oben schon erwähnt wurde, leicht realisiert werden. Um sich diese Art von „Glühen" besser vorstellen zu können, ist folgende Abbildung, auf der zwei Kreise mit und ohne diesen Schein nach außen zu sehen sind, bestimmt hilfreich:

Abbildung 7.5–c Darstellung des „Schein nach außen"-Effekts in Adobe Photoshop

Eine weitere Änderung zum ersten Entwurf des Logos war, dass die Personen im Logo einen anderen Effekt als auf der Zeichnung vorgesehen bekommen haben.

7.5.3 Verbesserungen und finale Version

Nach einem erneuten gemeinsamen Betrachten des Logos haben wir konstruktive Verbesserungsvorschläge entwickelt. Wir wollten das Aussehen des Buchstaben „V" und den Effekt der Personen weiter verbessern. Nach diesen Veränderungen hatten wir dann unsere finale Version des Logos. Außerdem haben wir danach die Farben und die Texte im Logo geändert und somit das V.watch-Logo kreiert. Die beiden fertigen Logos sehen folgendermaßen aus:

Abbildung 7.5–d Fertiges V.watch Logo

Abbildung 7.5–e Fertiges V.meet Logo

7.6 Usability

Dieses Kapitel beschäftigt sich mit der Benutzerfreundlichkeit *(=engl. Usability)* unserer Software. Grundsätzlich ist zu sagen, dass diese bei Programmen nicht so genau definiert sein sollte wie auf Web-Seiten. Auf Internetseiten fungiert der Benutzer so, dass er sich aussuchen kann, auf welcher Web-Seite er etwas tut. Sollte ihm die Benutzerfreundlichkeit auf der einen Seite nicht gefallen, wechselt er so schnell wie möglich zur Konkurrenz.

Bei Programmen hingegen kann sich der Anwender, nachdem er die Software gekauft hat, nicht einfach eine neue besorgen. Aufgrund dieser Tatsache hat er keine andere Wahl als sich mit der Software auseinanderzusetzen und sie zu erlernen. ([SCHW2003])

Trotz dieser Erkenntnisse wollten wir dem späteren Benutzer von V.watch eine gute Benutzerfreundlichkeit gewährleisten. Im folgenden Kapitel wird auf die Navigationsmöglichkeiten für den Anwender, die Verwendung von Komponenten und auf einen selbst erstellten Usability-Test eingegangen.

7.6.1 Navigation

Das Wichtigste für die Usability ist, dass sich der Benutzer gut und einfach orientieren kann. Im folgenden Abschnitt werden alle von uns im Programm verwendeten Navigationsmöglichkeiten beschrieben und analysiert.

7.6.1.1 Reitermenü

Um dem Benutzer aufzuzeigen, was die Grundfunktionen von V.watch eigentlich sind, dennoch aber die Navigation durch die Software übersichtlich zu gestalten, haben wir uns dazu entschlossen ein Reitermenü als Hauptnavigation einzusetzen. Mit der Maximalanzahl von sechs Reitern befinden wir uns im „Optimalfeld" der Übersichtlichkeit. Es gibt vier verschiedene Arten von Reitern. Einen Übersichtsreiter, der nicht geschlossen werden kann, maximal drei Konferenzreiter, einen Großansichtsreiter und einen Reiter für die Slideshow. Ursprünglich war auch noch eine fünfte Art, die des Alarmreiters, geplant, mit der das Nebenziel „Die Erstellung eines Alarmsystems" dargestellt werden sollte.

Dadurch, dass wir dieses Nebenziel hardware-basierend realisiert haben, fiel dieser Reiter jedoch weg. Außerdem wäre eine Anzahl von sieben Reitern eventuell nicht mehr übersichtlich gewesen.

7.6.1.2 Fußzeile:

Die etwas weniger wichtigen Menüpunkte sind in einer Fußzeile, die sich durch das komplette Programm zieht, zu finden. Diese Punkte sind keine Grundmodule von V.watch wie zum Beispiel die Übersicht oder die Großansicht und haben deshalb keinen Platz in dem Reitermenü gefunden. Man kann mit ihnen stattdessen zurück zum Willkommensbildschirm gelangen, das Impressum und die Hilfe anwählen, und zu den privaten Nachrichten wechseln. Außerdem kann man sich mithilfe des „Ausloggen"-Links, der sich ebenfalls in der Fußzeile befinden, schlicht und einfach aus dem Programm abmelden.

7.6.1.3 Navigation in der Übersicht

Wenn man nun die einzelnen Module von V.watch betrachtet, findet man in einem davon weitere wesentliche Navigationsmöglichkeiten. Aus der Übersicht heraus kann der Benutzer zu vielen verschiedenen Bereichen wechseln. Hierzu hat er die Möglichkeit im Interaktionsbereich (siehe *Kapitel 7.2.3, Ad Bereich C*) auf mit Icons[↗] besetzte Schaltflächen zu klicken, wie in der folgenden Abbildung zu sehen ist.

Abbildung 7.6–a Interaktionsbereich mit Schaltflächen in der Übersicht

Solche Interaktionsschaltflächen gibt es auch in anderen Modulen, jedoch sind diese weniger zur Navigation als zur Aktion vom Benutzer zu verwenden.

Außerdem gibt es in der Übersicht noch die Möglichkeit zwischen verschiedenen Seiten, auf denen die einzelnen Thumbnails[↗] platziert sind, zu wechseln, wie hier zu sehen ist.

Abbildung 7.6–b Seitennavigationsbereich in der Übersicht

Hier kann der Anwender mit den Pfeiltasten, nach links und nach rechts, die aktuelle Seitenzahl erhöhen oder verringern. Um dem Benutzer immer aufzuzeigen, auf welcher Seite er sich gerade befindet, wechselt das Feld in der mittigen `ComboBox`-Komponente automatisch mit. Mit diesem Element kann man auch, falls unübersichtlich viele Seiten vorhanden sind, leicht und schnell auf eine bestimmte Seite wechseln. Außerdem ist es hier möglich nach eigenen Wünschen die im System befindlichen Benutzer zu sortieren (zweite `ComboBox` von links). Um ein lästiges Abwählen der einzelnen Benutzer zu vermeiden, die man zuvor markiert hat, gibt es einen „Markierung aufheben"-`Button`, der dies schnell und einfach ermöglicht.

7.6.2 Komponenten

Wir haben uns auch aus Usability-Gründen dazu entschieden, in unserer Diplomarbeit Komponenten zu verwenden. Diese vorgefertigten Elemente kennt der Benutzer vor schon allem aus Web-Seiten und –Formularen und ist dadurch bereits mit ihnen vertraut. Um sich das vorstellen zu können, hilft vielleicht ein kleiner Vergleich zwischen einem Element einer Web-Seite und einem aus Flash. Beispiel: Die `ComboBox`-Komponente:

Abbildung 7.6–c Vergleich zwischen Flash Komponenten und Elementen aus dem Web

Hier ist zu sehen, dass sich die `ComboBox`-Komponente von Flash kaum von der Web-Version unterscheidet. Dies ist auch bei allen anderen so genannten „User-Interface"-Komponenten der Fall.

Zu diesem positiven Aspekt kommt noch hinzu, dass diese Komponenten leicht anzusprechen sind und sie eine Vielzahl an Möglichkeiten zur Veränderung bieten.

7.6.2.1 Nutzen für den Benutzer

Durch den Einsatz dieser User-Inteface-Komponenten fühlen sich alle Benutzer, die öfter etwas mit dem Computer machen, mit unserer Software vertraut. Sie kennen diese Elemente und müssen sich dadurch keine neuen aneignen, was meistens bei neuen Programmen der Fall ist. Außerdem kann durch die leichte grafische Anpassung der Komponenten gewährleistet werden, dass für den Benutzer bekannte Elemente in das Design des Programms eingebunden werden können.

7.6.3 Usability-Tests

Dadurch, dass wir als Entwickler dieser Software nicht die Meinung der späteren Anwender von V.watch, also unserer Zielgruppe, imitieren konnten, mussten wir dafür eine Lösung finden. Nur die Menschen, die wirklich später damit arbeiten sollten, konnten uns konstruktive Auskunft über Verbesserungen geben. Außerdem war für uns wichtig, die Meinung von Personen mit unterschiedlichen Computerkenntnissen einzuholen.

Aufgrund dieser Tatsachen entwickelten wir einen Usability-Test. Dieses Formular sollte uns helfen Verbesserungen an der Benutzerfreundlichkeit von unserer Software zu ermitteln. Der Gedanke hinter diesem Test lag dabei, dass einer unserer Mitarbeiter jeweils eine Person testet. Dazu sollten alle Rahmenbedingungen vorbereitet werden und der Test vom Mitarbeiter, nach Rücksprache mit der Testperson, ausgefüllt werden.

Der Test ist so aufgebaut, dass zuerst persönliche Daten des Testers aufgenommen und danach seine Computerkenntnisse durch einige Fragen ermittelt werden. Nach diesen einleitenden Maßnahmen bekommt die Testperson unterschiedliche Aufgabenstellungen.

Diese sind zum Beispiel „Registrieren Sie sich im System" oder „starten Sie eine Konferenz". Während die Testperson diese Aufgaben versucht zu bewältigen, betrachtet der Mitarbeiter folgende Punkte:

- Wie hat der Tester die Aufgabenstellung aufgenommen?
  Hier ist wichtig, die Fragestellung für den Test zu optimieren, damit durch Unklarheiten das Ergebnis, das eigentlich nur die Benutzer-freundlichkeit des Systems und nicht des Tests misst, nicht verfälscht wird.

- Wie war der Lösungsansatz des Testers für dieses Problem?
  Dies ist der wichtigste Abschnitt einer Aufgabenstellung. Hier muss der Mitarbeiter auf die Reaktionen der Testperson achten und sie notieren. Besonders zu beachten ist hier, wohin der Tester zuerst klickt bzw. welche Aktionen er zuerst durchführen möchte.

- Durchführungszeit
  Diese ist vor allem für spätere Statistiken wichtig. Diese konnten wir aufgrund der wenigen Tests zurzeit noch nicht aufstellen, jedoch könnte man herausfinden, wie stark sich das System verbessert hat. Solche Statistiken würden uns auch Auskünfte darüber geben, wie Personen mit unterschiedlichen Computerkenntnissen auf unser System reagieren und wie man dies effizienter und schneller gestalten könnte.

Außerdem können nach einer Aufgabenstellung noch eigene Kommentare vom Mitarbeiter und Anmerkungen vom Tester zur Übung eingetragen werden. Hier ist zu beachten, dass dies dezidiert nach der Übung stattfindet, um die Durchführungszeit nicht zu beeinflussen.

Das Formular für den Usability-Test von V.watch ist im Anhang zu dieser Arbeit zu finden.

7.7 Reitermenü

Wie schon im Punkt Usability erwähnt, war unser Ziel ein Reitermenü als Navigationsmöglichkeit in V.watch einzubauen. Es sollte sich dynamisch, den Einstellungen des Benutzers gegenüber, verhalten. Das heißt, es gibt keine fixe Anordnung der Reiterpunkte, stattdessen können immer neue Reiter hinzugefügt werden, wenn die jeweilige Funktion (Konferenz, Großansicht oder Slideshow) angewählt wurde. Grundsätzlich befindet sich der Punkt „Übersicht" immer an erster Stelle in der Navigation, da man von ihr aus alle anderen Bereiche erreichen kann.

Es bot sich uns auch die Möglichkeit das Reitermenü statisch aufzubauen. Diese Möglichkeit zogen wir jedoch nicht vor, da sich V.watch komplett an den User anpassen sollte. Außerdem fragten wir uns, ob der Benutzer das Reitermenü als solches erkennen würde, wenn wir es statisch anordneten. In der dynamischen Variante sieht man sofort, wie sich ein neuer Reiter öffnet, wenn man eine Funktion anwählt. Außerdem haben wir in der von uns gewählten Variante auch die Möglichkeit Reiter wieder zu schließen, was wiederum die Übersichtlichkeit von V.watch fördert. Ungebrauchte Punkte in der Navigation können so vom Benutzer leicht entfernt werden.

Nun stellte sich uns die Aufgabe diese dynamische Lösung zu realisieren. Sie musste sich flexibel gegenüber dem Öffnen, Wechseln und Schließen der einzelnen Reiter verhalten.

7.7.1 Realisierung

Es gibt drei Funktionen im Programmcode, die als Grundlage zur Realisierung des Reitermenüs dienen. Diese sind `createTabsMenu`, `showMcs` und `initTab`.

7.7.1.1 createTabsMenu

Diese Funktion ist eigentlich für den Aufbau des Reitermenüs zuständig. Sie nimmt ihre Informationen aus dem `TabsArray`-Array, in dem, mit Buchstaben gekennzeichnet, die einzelnen Reiter *(=Tabs)* stehen.

Hier ein Beispiel wie dieses Array, und danach auch das Reitermenü aussehen könnte:

```
TabsArray[0] = "O";
TabsArray[1] = "F";
TabsArray[2] = "M";
TabsArray[3] = "M";
TabsArray[4] = "S";
TabsArray[5] = "";
```

O … Overview

F … Fullscreen

M … Meeting

M … Meeting

S … Slideshow

Abbildung 7.7–a Reitermenü in V.watch

Sobald die Informationen aus dem Array vorliegen, kann die Funktion zwischen den Buchstaben differenzieren, sie nach und nach abarbeiten, die dazupassenden Reiter einfügen und die Positionen im Reitermenü setzen.

7.7.1.2 showMCs

Beim Anwählen eines Reiters im Menü wird die Funktion `showMCs` aufgerufen. Sie kontrolliert die Variable `actTab`, die davor gesetzt wird, und genau wie die einzelnen Elemente des `TabsArray`-Arrays die Buchstaben O, F, M und S beinhalten kann. Je nach dem, welcher Buchstabe sich in der Variable `actTab` befindet, wird entweder die Übersicht, die Großansicht, eine Konferenz oder die Slideshow dargestellt. Das Anzeigen der verschiedenen Menüpunkte funktioniert so, dass ein `MovieClip`, indem sich der Inhalt, der angezeigt werden soll, befindet, dynamisch in das Programm geladen wird. Die Inhalte aller anderen Menüpunkte werden entfernt, sodass nur der aktuell angewählte Inhalt zu sehen ist.

7.7.1.3 initTab

Nachdem ein Inhalt von einem der verschiedenen Menüpunkte angezeigt wurde, muss er nun „mit Leben gefüllt" werden. Das bedeutet, dass die `Listener` (siehe *Kapitel 2.4.5, Listener*) für alle Schaltflächen initialisiert, die Textfelder gefüllt und viele andere Dinge gemacht werden müssen. Dazu gibt es eigene Funktionen wie zum Beispiel `initMeeting`, die die jeweiligen Inhalte sozusagen starten. Da sich jedoch in allen Inhalten Flash Komponenten befinden, die erst einige Millisekunden nach dem Laden des `MovieClips` vom Programm ansprechbar sind, muss die Funktion `initTab` zuerst in einem gewissen Intervall aufgerufen werden. Die Funktion selbst ist dann eigentlich sehr simpel. Sie betrachtet auch das Attribut `actTab` und ruft danach die entsprechende `init`-Funktion auf (z.B. `initMeeting` für den Buchstaben „M").

Somit wurde das Reitermenü dynamisch generiert, festgelegt, was passiert, wenn ein Reiter ausgewählt wurde, und das Problem mit dem Ansprechen der Flash Komponenten gelöst. Um die Funktionalität des Reitermenüs zu vervollständigen, stellte sich uns die Aufgabe, Reiter ordnungsgemäß zu schließen.

Das Hauptaugenmerk beim Schließen lag für uns darauf, Reiter, die rechts vom zu schließenden Reiter liegen, mit all ihren Eigenschaften nachrücken zu lassen. Mit einem kleinen Algorithmus in der Funktion `createTabsMenu`, die etwas weiter oben schon erläutert wurde, schufen wir uns Abhilfe für dieses Problem. In diesem Programmteil werden alle leeren Elemente aus dem `TabsArray`-Array entfernt und dadurch rücken alle dahinter liegenden Felder nach. Dadurch kann einfach beim Schließen, also in der Funktion `closeTab`, das `TabsArray`-Array an der aktuellen Stelle geleert werden, also ein so genannter Leerstring[↗] eingesetzt werden. Wenn danach wieder die Funktion `createTabsMenu` aufgerufen, und somit das Reitermenü neu erstellt wird, fällt der aktuell gelöschte Reiter einfach weg.

Um diesen komplizierten Sachverhalt etwas verständlicher zu erläutern, wird nun eine kurze Abfolge von Ereignissen aufgezeigt, die dieses Problem exakt beschreiben:

1. Es besteht ein Reitermenü mit folgenden Reitern:
   Übersicht, Konferenz, Großansicht und Slideshow

Abbildung 7.7–b Reitermenü vor dem Schließen

2. Der Benutzer wechselt in die Großansicht und schließt diese danach. Das aktuelle Feld im `TabsArray`-Array wird nun vom System geleert, das heißt es wird ein Leerstring[↗] eingefügt. Ohne den Algorithmus in der Funktion `createTabsMenu` würde das Reitermenü wie folgt aussehen:

Abbildung 7.7–c Reitermenü nach dem Schließen ohne Bereinigungsalgorithmus

3. Doch, da wir diesen Algorithmus in das System eingebaut haben, wird nun aufgrund des Leerstrings im Inhalt dieses Element aus dem `TabsArray`-Array herausgefiltert und entfernt. Daraus resultierend, sieht der Benutzer das Reitermenü nicht in der oberen Form, sondern folgendermaßen:

Abbildung 7.7–d Reitermenü nach dem Schließen mit Bereinigungsalgorithmus

Die Funktionen im Hintergrund, also die Nachrichten an den Server, dass etwas geschlossen wurde, werden auch in der Funktion `closeTab` aufgerufen. Hier wird wieder unterschieden, welcher Reiter geschlossen werden soll, und je nach dem, werden Funktionen am Server aufgerufen.

Ein ziemlich umfangreiches Problem kam uns beim Schließen in die Quere. Nähere Informationen dazu, sind dem *Kapitel 8.6.2Unterscheidung der Konferenzen* zu entnehmen.

Zur Realisierung der Hauptnavigation ist noch zu sagen, dass sie zuvor in einem externen Flash-Dokument stattfand, und nach erfolgreichen Tests in V.watch implementiert wurde.

8 SPEZIELLE PROBLEMSTELLUNGEN {CSI}

Da jedes Projekt neu und einzigartig ist, treten im Verlauf dessen oft Probleme auf, mit denen man in genau dieser Form bisher noch nicht konfrontiert war – so auch während der Entwicklung von V.watch. Dieses Kapitel beschreibt spezielle Problemstellungen und passende Lösungen, mit denen wir im Lauf unserer Arbeit zu tun hatten und die uns erwähnenswert erscheinen.

8.1 Systemzeitdifferenz {CSI}

Die Zeit ist oft ein Problem – im täglichen Leben genauso wie in der Technik. Oft scheitert es an der Einteilung und Ähnlichem; in unserem Fall war aber die Synchronisation verschiedener Zeiten oft eine Fehlerquelle.

8.1.1 Problemstellung

Sobald mehrere Rechner miteinander verbunden werden und kommunizieren sollen, sind in manchen Punkten abweichende Konfigurationen keine Seltenheit. In unserem Fall haben wir das besonders bei der Systemzeit gemerkt, die, solange keine Zeitserversynchronisation verwendet wird, höchstwahrscheinlich immer eine unterschiedliche ist.

Da in V.watch an mehreren Punkten Berechnungen mit Zeitabhängigkeit erfolgen, z.B. bei der Anzeige der Zeit, die ein anderer Client bereits online ist, kam es wegen diesen Unterschiedlichkeiten oft zu Problemen.

8.1.2 Lösung

Glücklicherweise haben wir eine Komponente im System, die immer gleich funktioniert – der Server. Also kam uns die Idee, die Berechnungen mit Zeitabhängigkeit immer auf Basis der Serverzeit durchzuführen. Das würde aber heißen, dass bei jeder Berechnung eines Clients am Server angefragt werden müsste, welche Zeit dieser gerade hat – also eine nicht allzu schöne und effektive Lösung.

Unsere Lösung des Problems funktioniert mit nur einer Abfrage und zwar folgendermaßen:

Kommt ein Client online und wird seine Verbindungsanfrage vom Server akzeptiert, so ist das Erste, was er macht, eine Serverfunktion aufzurufen, die ihm dessen aktuelle Systemzeit als Timestamp, also in Millisekunden seit dem 1.1.1970, zurückgibt. Von diesem Wert wird danach die eigene Systemzeit subtrahiert, bevor die erhaltene (Systemzeit-)Differenz in einer globalen Variablen gespeichert wird.

Soll nun beispielsweise die Zeit angezeigt werden, die ein bestimmter Benutzer bereits angemeldet ist, so werden dessen Daten aus einem `SharedObject` ausgelesen. Darunter befindet sich unter anderem auch ein Timestamp der Zeit, zu der dieser online gekommen ist. Dieser Wert wurde zum Zeitpunkt der Anmeldung vom Server bestimmt und verzeichnet.

Mit der einfachen Formel

gesuchterWert = eigeneSystemzeit – loginZeit + clientServerDifferenz

kann nun der gesuchte Wert berechnet und angezeigt werden.

Mit diesem System ist es möglich, dass alle Berechnungen, auch wenn sie auf unterschiedlichen Clients ausgeführt werden, immer dasselbe richtige Ergebnis liefern – die Synchronisation ist damit geschafft.

Diese Problemstellung wird, inklusive Lösung in allgemeiner Form, in [KOPE1997] S. 60ff behandelt und ausführlich erläutert.

8.2 Mehrere Kameras zeitgleich verwenden {HAR}

Im Laufe der Arbeiten an diesem Projekt hatten wir sehr früh das Problem, dass wir mehrere Video-Streams benötigten, um unter anderem die Übersicht oder die Slideshow richtig zu entwickeln und zu testen. Aus diesem Grunde, und da es ohnehin als optionales Ziel definiert war, realisierten wir die Funktionalität mehrere Kameras gleichzeitig auf einem Client-Rechner mit einem Benutzernamen betreiben zu können.

8.2.1 Problemstellungen

Doch diese Erweiterung stellte uns vor mehrere Probleme. Einerseits unterstützt jede Flash-Player-Instanz[↗] genau eine Kamera. Außerdem kann jede Kamera nur von einer Flash-Player-Instanz[↗] benutzt werden. Dann mussten wir unserer Zusatzapplikation die Anmeldedaten zukommen lassen, und am Server unterscheiden, wer ein normaler Benutzer und wer einfach nur ein zusätzlicher Kamera-Stream ist. Die nächste Herausforderung war, wie wir den Zusatzapplikationen mitteilten, dass ihre zugehörige Hauptapplikation geschlossen wurde, und daher auch keine weiteren Daten empfangen werden dürfen. Und dann hatten wir noch einige Probleme mit der Aktualisierung der Stream-Listen in den `SharedObjects` und dem Löschen von Benutzer-informationen beim Ausloggen.

8.2.2 Lösungen

Da diese komplexen Problemstellungen nicht mit einer einzigen Lösung realisierbar sind, werden die nächsten Kapitel einzelne Teillösungen erläutern, die dann zusammen die Lösung des Gesamtproblems bilden.

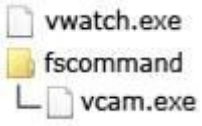
8.2.2.1 Neues Fenster öffnen

Da pro Instan[↗]z des Flash-Players nur auf eine Kamera zugegriffen werden kann, ist es notwendig, für jede zusätzliche Kamera ein neues .exe-File (Flash-Projektor) zu starten.

Lösung: die Flash-Methode `fscommand`.

Die Flash-Methode `fscommand` ermöglicht es uns ein neues File zu öffnen. Dieses File muss sich jedoch in einem Unterverzeichnis mit dem Namen `fscommand` befinden.

Abbildung 8.2–a Verzeichnisstruktur für Befehl „fscommand"



```
fscommand("exec","vcam.exe");
```

Der erste Parameter des Befehls `fscommand` entspricht dem Befehl, der ausgeführt werden soll. In unserem Fall lautet der Befehl `exec` – er bewirkt den Aufruf einer externen ausführbaren Datei.

Der zweite Parameter, der übergeben wird, entspricht dem Pfad der Datei, welche sich im Verzeichnis `fscommand` befindet.

8.2.2.2 Kamera-Auswahl

Wenn man mehrere Kameras auf einem Rechner benutzen möchte, ist es natürlich vor allem aus Sicht der Usability dringendst notwendig, dem Benutzer die Möglichkeit einer Kamera-Auswahl zu schaffen.

Lösung: `ComboBox` mit Werten aus `Camera.names` verwenden.

Der Benutzer hat somit die Möglichkeit, ganz einfach zwischen den angeschlossenen Webcams zu wechseln.

8.2.2.3 Übertragung

Die nächste Herausforderung bestand darin, das Video von der ausgewählten Kamera auf den Server zu übertragen (per Live-Stream).

Lösung: `NetConnection` und `NetStream` mit vorerst im Code festgelegten Login-Daten.

Die Funktionalität für den Verbindusaufbau wurde von unserer Hauptapplikation übernommen. Server-IP, Benutzername und Passwort wurden vorerst noch in den Code direkt programmiert, um zuerst einmal zu testen, ob alles andere, also der Aufruf von `vcam.exe` per `fscommand`, die Kameraauswahl in `vcam.exe`, die `NetConnection` und der `NetStream`, einwandfrei funktioniert.

8.2.2.4 Lokales SharedObject

Zur vollständigen Funktionalität und Modularität ist es jedoch notwendig, dass sämtliche für die `NetConnection` von `vcam.exe` benötigten Daten nicht in den Code direkt programmiert, sondern dynamisch von der dazugehörigen `vwatch.exe`-Hauptapplikation übernommen werden.

Lösung: lokales `SharedObject` für Login-Daten.

Die Hauptapplikation speichert die Verbindungsdaten in ein lokales `Shared-Object`. Benötigt werden die IP des Servers, sowie der Benutzername und das Passwort des angemeldeten Users.

```
Var localSO:SharedObject =
SharedObject.getLocal("userDetails","/");
```

Angegeben werden der Name und der Speicherpfad des Objekts. Der Schrägstrich bedeutet, dass dieses `SharedObject` für alle SWF-Files auf diesem Client zur Verfügung steht.

Dieses abgespeicherte `SharedObject` muss natürlich von `vcam.exe` ausgelesen werden, um mit den gewonnenen Daten die Verbindung herstellen zu können.

```
Var localSO:SharedObject =
SharedObject.getLocal("userDetails","/");
```

Mit den Eigenschaften `serverIP`, `clientID` und `clientPW` wird die `NetConnection`-Verbindung aufgebaut.

8.2.2.5 ClientID dynamisch verändern

Wenn die `vcam.exe`-Applikationen nun beginnen, ihre Videos auf den Server hochzuladen, dann passiert folgendes: Der Stream[↗] von der Hauptapplikation und der neue Stream überschreiben einander gegenseitig laufend und man sieht, sobald man diesen Stream wiedergibt, lediglich ein Standbild. Dieses Standbild ist das letzte Bild, das der Server von der Hauptapplikation empfangen hat, bevor der Stream überschrieben wurde.

Lösung: Änderung des ursprünglichen Clientnamens auf die Zeichenkette `[clientID]~[fortlaufendeNr]` und Übergabe des Verbindungstyps bei der Anmeldung.

Sobald sich eine Applikation zum Server verbindet, wird nun ein zusätzliches Attribut übergeben, das die Art des Clients angibt. Entweder man verbindet sich als `mainUser`, dann verläuft alles wie bisher, oder man hat als Wert dieses Attributs `additionalCamera` gesetzt, dann müssen kleine Änderungen und Abfragen durchgeführt werden. Die `clientID` soll jetzt abgeändert werden auf `[clientID]~[fortlaufendeNr.]`. Hierfür durchsucht das Server-Skript das Array `userInformation` und nimmt die niedrigste freie Nummer. Am Server wird jetzt der Eigenschaft `name` des `Client`-Objekts, das sich gerade verbinden will, der neue, erweiterte Name zugeteilt und außerdem wird in der Client-Applikation `vcam.exe` die Funktion `yourNameIs` aufgerufen und als Parameter die neue `clientID` übergeben.

8.2.2.6 Automatische Abmeldung

Falls ein Benutzer die Applikation beenden möchte, dann schließt er oder sie das Fenster, in dem die Hauptapplikation läuft, oder drückt auf „logout". Das heißt aber, dass nicht nur die Streams, die von der `vwatch.exe` kommen nicht mehr angezeigt werden sollen, sondern auch jene von den dazugehörigen `vcam.exe`-Instanzen.

Lösung: bei Logout/Disconnect des `mainUsers` wird ein Auto-Disconnect aller zugehörigen `additionalCameras` erzwungen.

Sobald ein `mainUser` die Verbindung beendet, wird das serverseitige Event `Application.onDisconnect` aufgerufen. In diesem Ereignis wird ein Funktion `dropChildren` aufgerufen und als Parameter die `clientID` übergeben.

Diese Funktion macht im Prinzip nichts anderes, als dass sie das Array `Application.clients`, welches sämtliche zum Server verbundene Clients enthält, durchgeht und vergleicht, ob es passende `additionalCameras` zu dem `mainUser` gibt. Falls welche gefunden werden, werden deren Verbindungen mit dem Server zwangsweise getrennt.

```
Application.disconnect(Application.clients[i],"mainUserDisconnected
");
```

Die Funktion verursacht das Trennen der Verbindung zum Server. Der erste Parameter ist das Client-Objekt, dessen Verbindung getrennt werden soll. Der letzte Parameter ist eine Statusmeldung, die an den Client geschickt wird.

8.2.2.7 Aktualisierung der SharedObjects

Wenn das Streaming eines Videos begonnen oder beendet wird, dann wird das Array `streamInformations` und daraufhin das `SharedObject streamSO` und dessen Eigenschaft `liveStreams` aktualisiert. Jedoch wird während dieses Vorganges eine Administratoren-Verbindung benötigt, um mit Hilfe eines Server-Management-Skripts die aktuelle Liste der Streams am Server abzurufen. Diese Aktion benötigt aber mehr Zeit, als sie haben dürfte – das Array und das `SharedObject` werden zu früh aktualisiert.

Lösung: mittels setInterval eine Wartezeit von 0.5 bis 1 Sekunde erzwingen.

Zwar ist dies keine professionelle Lösung, so gibt es jedoch vorerst keine andere Möglichkeit.

Zuerst wird dazu eine globale Variable angelegt, um die `intervalID` darin zu speichern. Wann immer die Stream-Liste aktualisiert werden soll, wird `setInterval` ausgeführt, die ID gespeichert und in der Rückgabe-Funktion `executeCallback` nach Ablauf der Wartezeit die Liste aktualisiert. Diese Lösung verschafft dem Management-Skript mehr Zeit für seine Aktionen.

```
intervalID = setInterval(executeCallback,500);
```

Der erste Parameter gibt die Rückgabe-Funktion an, die nach Ablauf der Zeit aufgerufen wird. Der zweite Parameter gibt die Zeit, die zwischen den Funktionsaufrufen gewartet werden soll, in Millisekunden an.

Die Funktion `executeCallback` ruft auf jedem Client die Funktion `updateStreamList_client` auf, um die Anzeigen zu aktualisieren. Außerdem wird das Intervall mit der global gespeicherten `intervalID` gelöscht.

8.2.2.8 Philosophenproblem & Bookingsystem

Die Probleme aus *Kapitel 8.2.2.6, Automatische Abmeldung* und *8.2.2.7, Aktualisierung der SharedObjects* zusammen ergeben nun ein neues Problem. Wenn ein `mainUser`, der eine oder mehrere `additionalCameras` verbunden hat, die Verbindung beendet, so wird für jedes einzelne `onDisconnect`-Event das `setInterval` aufgerufen und die globale `intervalID` gesetzt. Was jetzt passiert, ist die logische Folge. Die kritische Variable `intervalID` wird mehrmals überschrieben, und zwar bevor das dazugehörige Intervall gelöscht wurde. Dies führt dazu, dass ausschließlich das allerletzte Intervall gelöscht wird, die anderen laufen unendlich lange weiter. Es ist nicht mehr möglich sie zu löschen.

Lösung 1: Semaphor einsetzen.

Hierzu benötigen wir eine neue globale Variable vom Typ `Boolean`.

```
Var intervalIDInUse;
```

Diese Variable wird jedes Mal, wenn das Skript versucht `setInterval` aufzurufen, abgefragt. Und nur, wenn sie nicht gesetzt ist, darf `intervalID` ein neuer Wert zugewiesen werden. Natürlich muss der Semaphor auch irgendwann wieder entsperrt werden. Dies wird in der Funktion `executeCallback` gemacht.

Doch diese Lösung führte in bestimmten Situationen zum selben Problem. Wenn nämlich ein Client kurz vor dem Freigeben der kritischen Variable online kommt und versucht die Liste zu aktualisieren, dann wird er blockiert.

Wenn dann unmittelbar danach die Variable wieder freigegeben wird, versucht es der neue Client nicht mehr. Man erkennt hier vielleicht das Problem nicht auf den ersten Blick. Aber der Aktualisierungsvorgang, der bereits im Laufen war, während der neue Client online gekommen ist, hat die Streams dieses Clients noch nicht bedacht. Das heißt, sein Stream wird auch nach erfolgreicher Aktualisierung noch nicht angezeigt werden.

Lösung 2: Semaphor-Lösung auf Bookingsystem erweitern.

Nun werden die einzelnen Aktualisierungswünsche nicht mehr mit Unterstützung der globalen `intervalID` getätigt, sondern jeder Wunsch einer Aktualisierung wird mit einem Zeitstempel, der in ein Array eingetragen wird, geäußert. Es werden also die Aktualisierungen „gebucht". Dieser Zeitstempel entspricht der Zeit zum Zeitpunkt des Aktualisierungswunsches plus 500 Millisekunden. Alle 700 Millisekunden wird dieses Array abgearbeitet. Jeder Zeitpunkt, der bereits vorbei ist, wird aus dem Array entfernt. Dies wird solange gemacht, wie Einträge im Array sind. So wird sichergestellt, dass für jede angeforderte Aktualisierung auch richtig und zeitgemäß aktualisiert wird.

8.2.2.9 Status-Arrays aktualisieren

Wenn ein Client die Verbindung beendet, dann wird seine Zeile im Array `userInformation` auf den Wert `null` gesetzt. Wenn das jedoch sehr oft passiert, also sehr viele An- und Abmeldungen am Server geschehen, dann wächst das Array ins Unermessliche, obwohl im Prinzip nur einige wenige Clients eingetragen sind.

Lösung: Elemente mit `splice` aus Benutzerdaten-Array entfernen.

Anstatt das Array-Element wie bisher auf `null` zu setzen, wird ab jetzt die Array-Methode `splice` verwendet, um Einträge aus dem Array `userInformation` zu entfernen.

8.3 Netzauslastung vs. Streamqualität {CSI}

Des Öfteren muss man sich für eine Sache entscheiden – beides geht selten. In unserem Fall wäre der Idealfall eine geringe Netzauslastung bei sehr guter Streamqualität von Audio- und Videodaten. Klar ist aber auch, dass qualitativ hochwertige Streams mit hoher Auflösung eine große Belastung des Netzwerks verursachen.

Wir mussten uns also etwas einfallen lassen, um möglichst gute Streamqualität bei möglichst kleiner Netzbelastung zu garantieren, sodass der Benutzer sein Gegenüber detailgenau und ruckelfrei sehen und hören kann.

8.3.1 Problemstellung

In V.watch gibt es mehrere Möglichkeiten, wie ein Stream angezeigt wird. Zum einen gibt es die Übersicht, wo Streams nur in einer kleinen Auflösung (115x86 Pixel) dargestellt werden – dasselbe gilt übrigens für die Meeting-Ansicht. Andererseits gibt es Großansicht und Slideshow, wo Streams in einer höheren Auflösung dargestellt werden sollen.

Dass wir nicht immer in voller Auflösung streamen können, war uns von Anfang an klar – das würde das Netz einfach zu sehr belasten. Kleine Streams aber größer zu skalieren und interpolieren, um eine Großansicht zu erhalten, war aber auch nicht zufrieden stellend, da das Bild verschwommen dargestellt wurde.

Also musste ein System her, in dem beides – je nach Bedarf – funktioniert. Die Lösung im Detail sieht folgendermaßen aus:

8.3.2 Lösung

Sobald ein Benutzer den Button zum Anzeigen der Großansicht eines anderen Benutzers drückt, schickt er eine Anfrage zum Server. Als Parameter werden der Username des gewünschten Benutzers und der eigene Username mitgeschickt. Beim Öffnen einer Slideshow wird natürlich für jeden markierten User ein Request gesendet.

Der Server speichert nun in einer Liste mit, wer diesen so genannten `fullscreenRequest` geschickt hat, wem er gilt und wie viele Requests dieser User bereits bekommen hat (der Zähler wird erhöht). War dies der erste Request für den gewünschten Benutzer, bekommt dieser den Befehl, seine Kamera in eine höhere Auflösung umzuschalten und einen qualitativ hochwertigeren Stream[↗] zu senden. Wenn davor schon mindestens ein Request verzeichnet war, wird nur der Zähler erhöht, der Benutzer sendet ja bereits einen hoch auflösenden Stream.

Um die Kamera umzuschalten, wird am Client folgender Code verwendet:

```
serverConnection.sendFullscreenStream = sendFullscreenStream;
function sendFullscreenStream():Void  {
      // Kamera-Settings verändern
      my_cam.setMode(640,480,30);
};
```

Die Funktion `sendFullscreenStream` wird der `NetConnection` zugeordnet, damit sie vom Server aufgerufen werden kann. In der Funktion passiert folgendes: Die Kamera schaltet mit dem Befehl `setMode` auf eine Auflösung von 640x480 Pixel und eine Bildrate von 30 Bildern pro Sekunde um.

Das Szenario beim Schließen einer Großansicht bzw. Slideshow sieht analog dazu aus. Es wird wieder eine Funktion am Server aufgerufen, die diesmal den Zähler vermindert und gegebenenfalls, also falls der Zähler gleich 0 ist, das Clientprogramm dazu auffordert, die Kameraauflösung wieder runter zu schalten. Die passende Funktion sieht so aus:

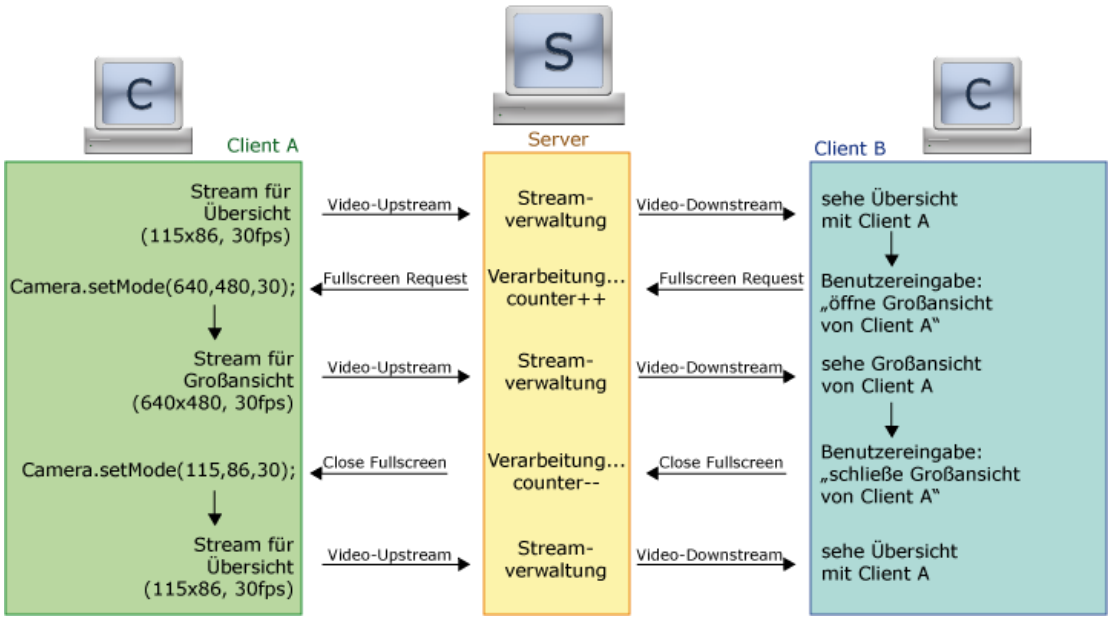
```
serverConnection.sendOverviewStream = sendOverviewStream;
function sendOverviewStream():Void  {
      // Kamera-Settings verändern
      my_cam.setMode(115, 86, 30);
};
```

Mit dieser Lösung wird also immer nur so viel Traffic verursacht, wie auch wirklich notwendig ist.

Gemeinsam mit der von uns entwickelten automatischen Bandbreiten-kontrolle, die am Server abläuft (siehe *Kapitel 8.4, Netzauslastung optimieren {HAR}*), ist damit eine hohe Benutzerfreundlichkeit gegeben.

Abbildung 8.3–a stellt unsere Lösung im zeitlichen Verlauf grafisch dar:

Abbildung 8.3–a Umschalten zwischen Übersicht und Großansicht



8.4 Netzauslastung optimieren {HAR}

Die Stream-Qualität den Anforderungen des Benutzers – automatische Bereitstellung von Übersichts- oder Großansichts-Stream – anzupassen ist zwar eine gute Präventivmaßnahme um überlastete Netzwerke einigermaßen gut zu verhindern (siehe *Kapitel 8.3, Netzauslastung vs. Streamqualität {CSI}*). Doch um auf Netz-Überlastungen zu reagieren, mussten wir zusätzlich dazu noch eine Art Bandbreitenkontrolle entwickeln.

8.4.1 Problemstellung

Je mehr Benutzer mit unserem System arbeiten, desto ausgelasteter sind auch die Netzwerke, mit denen unser Server verbunden ist. Je überlasteter ein Netzwerk ist, desto langsamer kann man Daten senden.

Bei hoher Stream-Qualität ist es jedoch notwendig sehr viele Daten zu übertragen, um nicht beim Abspielen ein stockendes Video zu sehen. Das bedeutet, dass die Stream-Qualität sich der zur Verfügung stehenden Bandbreite anpassen müsste, und das ist das Problem.

8.4.2 Lösung

Um herauszufinden, wie ausgelastet das Netz ist, mussten wir eine Bandbreitenkontrolle entwickeln. Dieses System ist auf zwei Teile aufgeteilt, denn da sich Clients sowohl aus einem internen Netzwerk, als auch aus dem Internet zum Server verbinden können, müssen wir zuerst herausfinden, ob es sich um einen Client aus einem internen oder einem externen Netz handelt. Und der zweite Teil überprüft die Auslastung der Verbindung zum jeweiligen Client, und setzt gegebenenfalls das Qualitäts-Level eine Stufe höher bzw. niedriger.

8.4.2.1 IP-Bereich feststellen

Sobald sich ein Client mit dem Server verbindet, wird neben der üblichen Authentifizierungsprüfung auch die IP-Adresse interpretiert. Denn es ist leicht zu verstehen, dass Verbindungen aus dem LAN viel höhere Bandbreiten erreichen, als jene aus dem Internet. Mit dem Befehl `Client.ip` wird die Adresse abgerufen. Diese wird anschließend zerlegt und dann mit den üblich als interne IP-Adressen verwendeten Adressen verglichen.

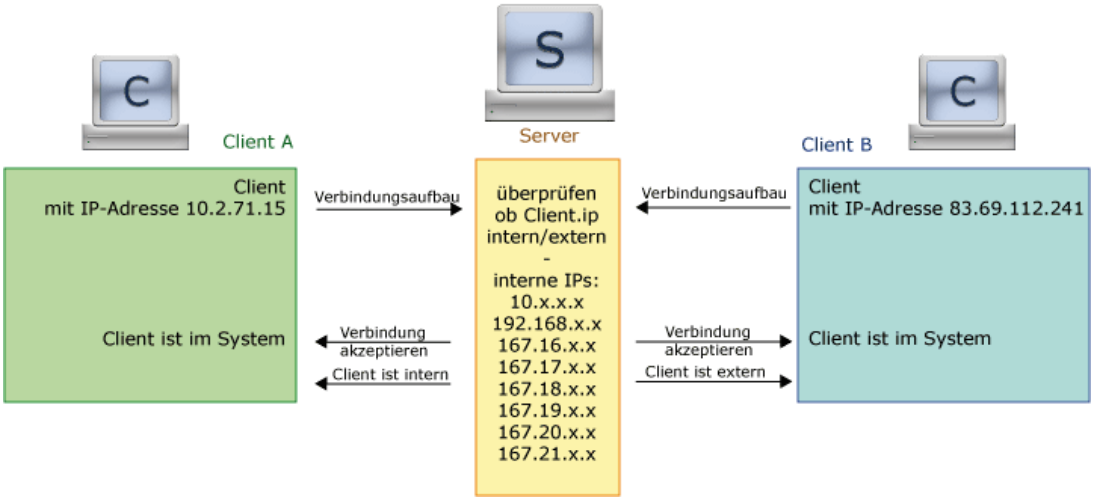
Tabelle 8.4-a interne IP-Adressen

IP-Adressen
10.x.x.x
192.168.x.x
167.16.x.x
167.17.x.x
167.18.x.x
167.19.x.x
167.20.x.x
167.21.x.x

Anmerkung zur Tabelle: Die „x“ bei den einzelnen Adressen stehen für beliebige Zahlen, die in den jeweiligen Adressbereich passen.

Wenn die Adresse mit einer der in der Tabelle eingetragenen IP-Adressen übereinstimmt, so wird der Client als intern eingestuft. Andernfalls wird angenommen, dass es sich um eine Adresse aus dem Internet handelt und der Client wird als extern eingestuft. Der zeitliche Ablauf dieses Vorganges wird in der nachfolgenden Abbildung 8.4–a dargestellt.

Abbildung 8.4–a IP-Bereich feststellen



8.4.2.2 *Auslastungsüberprüfung und Änderung des Qualitäts-Levels*

Zwar wissen wir jetzt, welcher Client aus dem LAN und welcher aus dem Internet mit dem Server verbunden ist, jedoch haben wir noch keinerlei Erkenntnisse über deren Bandbreitenauslastung erlangt. Hierzu verwenden wir die Methode `Client.getStats()`, mit der es möglich ist die sogenannte „Ping-Round-Trip-Time“ [↗], das ist die Zeit in Millisekunden, die ein Ping-Request-Response-Zyklus dauert, eines jeden Clients zu ermitteln. Und anhand dieser PRT-Time[↗] können wir feststellen, welches Qualitäts-Level diese Verbindung unterstützt.

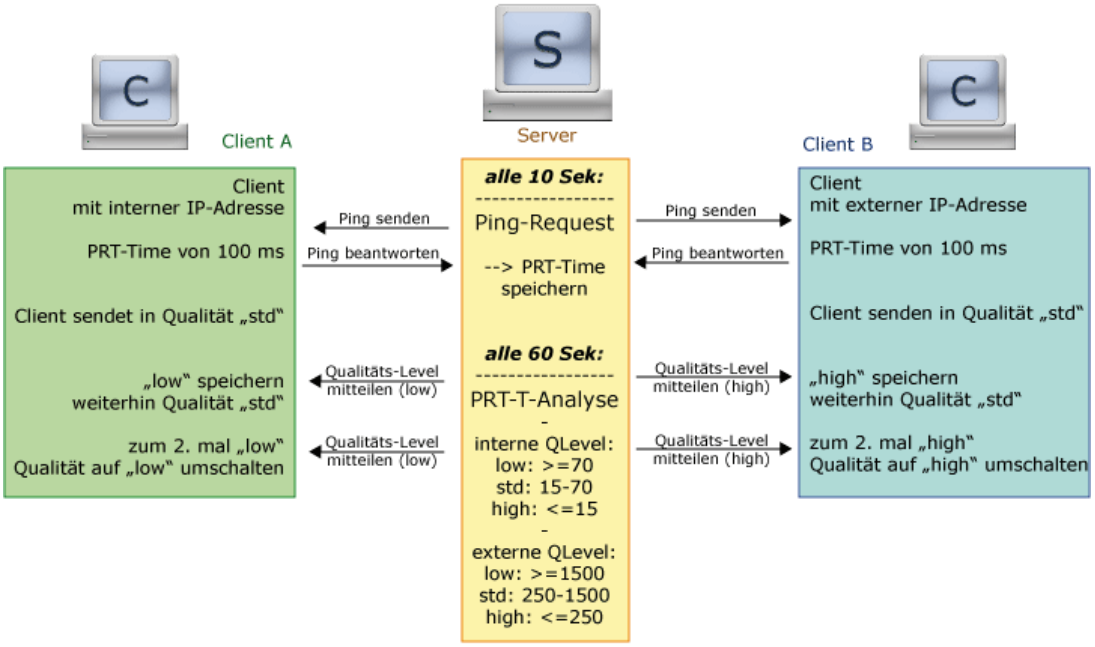
In der nachfolgenden Tabelle werden die Grenzwerte der PRT-Time[↗] der drei verschiedenen Qualitäts-Level „low", „standard" und „high" jeweils für interne und externe Verbindungen angegeben.

Tabelle 8.4-b Grenzwerte der Qualitäts-Level

	intern	extern
low	>=70	>=1500
standard	15-70	250-1500
high	<=15	<=250

Alle zehn Sekunden wird eine derartige PRT-Time-Erhebung[↗] durchgeführt. Die Ergebnisse werden abgespeichert und gesammelt, und alle 60 Sekunden analysiert und interpretiert. Da es durchaus sein kann, dass kurzfristig höhere Überlastungen des Netzwerkes auftreten können, fangen wir dieses Szenario ab und nehmen den zweithöchsten Wert für die Ermittlung des Qualitäts-Levels. Es wird also für jeden Client die zweithöchste PRT-Time[↗] mit der Tabelle verglichen, je nachdem, ob der Client im internen oder externen Netz ist, mit den Werten einer anderen Spalte. Dem Client wird dann anschließend mitgeteilt, für welche Qualitäts-Stufe er momentan qualifiziert wäre. Der Client vergleicht das aktuell verwendete Level mit dem neuen Wert. Ist der neue Wert gleich dem aktuellen, so ändert sich nichts. Sollte sich das Level jedoch geändert haben, so wird der neue Wert vorerst nur einmal abgespeichert. Nach der nächsten Bandbreiten-Analyse wird das neue Level erneut an den Client gesendet. Ist nun der Wert wiederum anders, als das aktuelle Level, so wird diesmal die Qualität des Streams geändert. Andernfalls bleibt die momentan gesendete Qualitätseinstellung. Man benötigt also zwei mal hintereinander schlechtere bzw. bessere Werte, um eine Qualitäts-Änderung zu bewirken. Hiermit wird verhindert, dass laufend die Qualität geändert wird, wenn das Netz ständig zwischen ausgelastet und unausgelastet wechselt. Der zeitliche Ablauf dieses Vorganges wird in der nachfolgenden Abbildung 8.4–b dargestellt.

Abbildung 8.4–b Auslastungsüberprüfung und Änderung des Qualitäts-Levels



8.4.2.3 Anwendung der Qualitätslevels am Client {CSI}

Um die Qualitätslevel im Client-Programm anzuwenden, wird die Methode `setQuality` der `Camera`-Klasse verwendet. Mit dieser Methode kann festgelegt werden, ob die Ausnutzung der Bandbreite oder die Bildqualität des Videofeeds für die Anwendung Vorrang haben soll.

```
Function setStreamQuality():Void {
      switch(actStreamQuality) {
            case 0 :
                  // maximale Bandbreite 1kB
                  my_cam.setQuality(1024,0);
                  break;
            case 1 :
                  // maximale Bandbreite 5kB
                  my_cam.setQuality(5120,0);
                  break;
            case 2 :
                  // maximale Bandbreite 100kB
                  my_cam.setQuality(102400,0);
                  break;
      }
};
```

Der erste Parameter gibt an, welche Bandbreite (in Byte pro Sekunde) für den Stream maximal verwendet werden soll. Wenn so viel Bandbreite verfügbar sein soll, wie nötig ist, um die angegebene Qualität zu übertragen, muss 0 übergeben werden. Als zweiter Parameter wird die gewünschte Bildqualität angegeben – sie ergibt sich aus dem Grad der Komprimierung jedes Videobildes. Es können Werte zwischen 1 (maximale Komprimierung) und 100 (keine Komprimierung) angegeben werden. 0 bedeutet, dass alleine auf die maximale Bandbreite geachtet werden soll.

Der Wert, welcher mit der Methode `Camera.setQuality` gesetzt werden soll, hängt von den Vorgaben des Servers ab und bewegt sich zwischen 1kB (niedrigste Qualität) und 100kB (höchste Qualität).

8.5 Probleme mit Komponenten {SEI}

Im Zuge unserer Diplomarbeit sind wir auf einige wenige Probleme, zusammenhängend mit Flash Komponenten, gestoßen. Durch viel Recherchearbeit und vor allem langwierige Tests sind wir bei den meisten Problemen auf verschiedenste Lösungsansätze gekommen.

8.5.1 Komponenten in hinzu geladenen MovieClips

Eines unserer größten Probleme war, dass Komponenten in `MovieClips` (=`MC`), die von uns erst während der Laufzeit des Programms hinzu geladen wurden, nicht ansprechbar waren. Man konnte zwar alle Eigenschaften der `MovieClip`-Klasse im Bezug auf die Komponenten auslesen und verändern, jedoch keine Bearbeitung von anderen Eigenschaften vornehmen. Wie in Kapitel *2.4.3, Vererbung* schon angeführt wurde, ist die `MovieClip`-Klasse die oberste Vaterklasse aller Komponenten, da sie auch über der `UIObject`-Klasse liegt. Nun haben wir aufgrund von verschiedenen Tests und Proben herausgefunden, dass auch Eigenschaften der `UIObject`-Klasse nicht mehr anzusprechen sind.

Dadurch, dass wir in V.watch sehr viel mit hinzu geladenen `MovieClips` gearbeitet haben, mussten wir hierfür eine Lösung finden.

Als Beispiel zur Erläuterung unserer Lösungsansätze möchte ich das Reitermenü anführen. Sobald der Benutzer einen Menüpunkt ausgewählt hat, werden alle `MCs`, die den Inhalt der anderen Menüpunkte beinhalten (Großansicht, Slideshow,…) entfernt und der aktuell ausgewählte Inhalt als `MC` angezeigt. Die Prozedur des Hinzuladens wurde mit der Funktion `attachMovie` durchgeführt.

Der erste Lösungsansatz war, nach dem Mausklick auf einen Reiter im Menü eine `onMouseUp`-Prozedur aufzurufen, die ausgeführt werden sollte, wenn man die Schaltfläche auf der Maus nach dem Klick loslässt. Es funktionierte, und man konnte die Komponenten im vollen Spektrum ansprechen. Jedoch kam uns diese `onMouseUp`-Prozedur bei anderen Dingen, die in diesen Inhalts-`MovieClips` passierten, in die Quere. Somit brachte diese Lösung zu viele andere Probleme, um sich durchzusetzen.

Der zweite Lösungsansatz war, die Inhalts-MCs bereits fertig auf der Bühne[↗] zu platzieren und sie danach, statt zu entfernen, unsichtbar zu machen. Diese Lösung funktionierte am besten, jedoch war die Stapelung der MCs zu speicherintensiv und machte die FLA-Datei[↗] sehr groß.

Der dritte Lösungsansatz hatte mit einer Methode der `UIObject`-Klasse zu tun. Diese heißt `doLater` und muss direkt in den zu ladenden `MovieClip` geschrieben werden. Mit ihr kann man Funktionen angeben, die auf diese Komponenten später zugreifen möchten, und damit gewährleisten, dass sie ansprechbar sind. Dieser Ansatz zur Lösung wurde von Anfang an von uns abgelehnt, da wir den Programmcode nicht verschachtelt in `MCs` schreiben wollten. Dies wäre die alte Programmierweise von ActionScript, die aufgrund ihrer Verschachtelung bei Fehlerbehebungen nicht vorteilhaft war.

Der vierte Lösungsansatz war, nach dem Klicken auf einen Reiter im Menü, ein Intervall zu starten, das eine gewisse Zeit lang durchläuft und danach die Komponenten mit so genannten `init`-Funktionen (`init`=initialisieren) anspricht.

Wir haben uns im Endeffekt für diese Variante entschieden, da auch die Einteilung in `init`-Funktionen zusätzliche Strukturierung brachte. (nähere Informationen zu diesem Thema sind in *Kapitel 7.6.1.1, Reitermenü* zu finden)

Grundsätzlich ist zu sagen, dass man in späteren Projekten abwägen sollte, welchen dieser vier Lösungsansätze man verwendet, um dieses Problem zu lösen, da man vermutlich öfters darauf stoßen wird.

8.5.2 Probleme mit dem Level von Komponenten

Es tauchen manchmal Probleme in Flash auf, für die es offensichtlich keine Lösung gibt. Ein solches Problem hatten wir bei dem Versenden neuer privater Nachrichten. Die `ComboBox`-Komponente zum Auswählen des Empfängers der Nachricht ließ sich in manchen Fällen nicht öffnen. Nachdem wir das Fenster zum Verfassen der neuen Nachricht geschlossen haben, und die `ComboBox` kurz aufflackern sahen, sind wir darauf aufmerksam geworden, dass es sich zwar öffnen lässt, jedoch hinter diesem Fenster, sodass es der Benutzer nicht sehen kann. Wir vermuten, dass dieses Problem mit dem Flash-Projektor[↗] zusammenhängt, da es nach dem Neustarten des Projektors oftmals wieder einwandfrei funktionierte. Wir dachten, dass dieses Problem eventuell mit den verschiedenen Höhen *(=Levels)* zusammenhängen könnte, sodass die Komponente hinter dem `MovieClip` liegt. Wir versuchten mit der Methode `swapDepths` das Level der Komponente zu erhöhen, was uns jedoch nicht gelang. Wir fanden für dieses Problem keine Lösung.

8.6 Probleme mit dem Menü {SEI}

Natürlich funktionierte die Implementierung des Reitermenüs nicht auf Anhieb. Vor allem die Tatsache, dass zuvor nicht mit allen Funktionen getestet werden konnte, trug dazu bei.

8.6.1 Ansprechen der Komponenten

Wie schon bei der Realisierung des Menüs erwähnt, konnten wir die vorgefertigten Flash Komponenten nicht direkt nach dem Laden des `MovieClips` ansprechen. Verschiedene Problemlösungen zu diesem Thema kann man in *Kapitel 2.4, Flash-Komponenten {SEI}* genauer nachlesen. Die Möglichkeit, die wir in dieser Situation gewählt haben, war, eine Funktion in einem gewissen Intervall auszuführen, die wiederum die eigentlichen Initialisierungsfunktionen aufruft. Somit konnten wir einige Millisekunden gewinnen, die die Komponenten benötigen, um selbst initialisiert zu werden.

8.6.2 Unterscheidung der Konferenzen

Unser größtes Problem, das mit dem Reitermenü zusammenhing, war die Differenzierung der maximal drei Konferenzen, die ein User offen haben kann. Klar war uns, dass wir sie mit der Meeting-ID (=`Mid`) unterscheiden konnten. Im Bezug zum Reitermenü entschieden wir uns danach, ein eigenes `MidArray`-Array zu generieren, das anfangs nur ein Feld für die Meeting ID hatte. Es gab für jeden der fünf Reiter ein eigenes Element im `MidArray`-Array. Nun nehmen wir ein Beispiel zur Hand, um den Sachverhalt genauer zu verstehen. Wenn der Benutzer nun als zweiten Reiter eine Konferenz startet, ist im zweiten Element des `MidArray`-Arrays die ID des Meetings *(=Konferenz)* zu finden. Diese Differenzierung funktionierte bis zu diesem Zeitpunkt noch sehr gut. Dadurch, dass wir auch den Meeting Titel lokal speichern wollten, erweiterten wir das Array um ein Feld. Auch soweit war unser System stabil. Jedoch wollten wir auch Reiter schließen. Wenn wir nun einen Reiter schlossen, war das `MidArray`-Array mit dem `TabsArray`-Array nicht mehr synchron. Wir standen hier vor einem großen Problem.

Wir mussten eine gewisse Abhängigkeit zwischen den beiden Datencontainern schaffen. Also erweiterten wir das `MidArray`-Array nochmals um ein Element, indem wir den Index des Reiters eintrugen, hinter dem sich das Meeting versteckt. Somit war die Problematik in klare Teilgebiete strukturiert. Wir mussten diese Abhängigkeit verwenden, um eine Synchronisation zwischen den beiden Variablen herzustellen.

Dazu hilft uns auch im laufenden Programm noch die Funktion `findRightIndex`. Sie sucht das Element im `MidArray`-Array, das sich hinter dem aktuellen Reiter verbirgt. Der letzte Brocken, den wir bei diesem Problem noch abzuarbeiten hatten, war das Nachrücken der Elemente im `MidArray`-Array beim Schließen eines Reiters. Hier mussten wir wieder das Element finden, das sich hinter dem, zu löschenden Reiter, befand, es entfernen, danach alle dahinter liegenden Elemente verkleinern, herausfinden, welcher Reiter danach nicht mehr besetzt ist und ein Element mit dessen Werten am Ende des Arrays anhängen. Somit kann nun immer zwischen den einzelnen Konferenzreitern unterschieden werden.

8.7 Dateiversand {CSI}

Um den Benutzern von V.watch neben Audio-, Video- und Textnachrichten noch eine weitere Form der Kommunikation zu ermöglichen, haben wir uns dazu entschieden, eine Funktion zum Austausch von Dateien einzubauen. Diese Funktion ist an zwei Stellen in unserem Programm in leicht unterschiedlichen Ausprägungen verfügbar: Befindet sich der Benutzer in der Übersicht, kann er eine Datei zu einem oder mehreren markierten Benutzern übertragen. Diese müssen dem Transfer zuvor natürlich zustimmen. In Konferenzen haben Benutzer die Möglichkeit, Dateien quasi freizugeben. Freigegebene Dateien können von anderen Teilnehmern derselben Konferenz jederzeit und ohne Absprache mit dem Bereitsteller heruntgeladen werden.

Setzt die übrige Client-Server-Kommunikation in V.watch komplett auf dem Adobe-proprietären Protokoll RTMP[↗] auf, so muss beim Dateiversand auf ein anderes Protokoll zurückgegriffen werden: HTTP[↗].

8.7.1 HTTP

HTTP steht für „Hypertext Transfer Protocol" und ist die Grundlage der Dateiübertragung in V.watch. Genau genommen ist dieses Protokoll eine Beschreibung, wie die Kommunikation zwischen Client und Server, d.h. der Austausch von Dateien, ablaufen soll. Normalerweise wird HTTP verwendet, um Webseiten des WWW von Webservern zu Browsern auf Client-Computern zu transferieren.

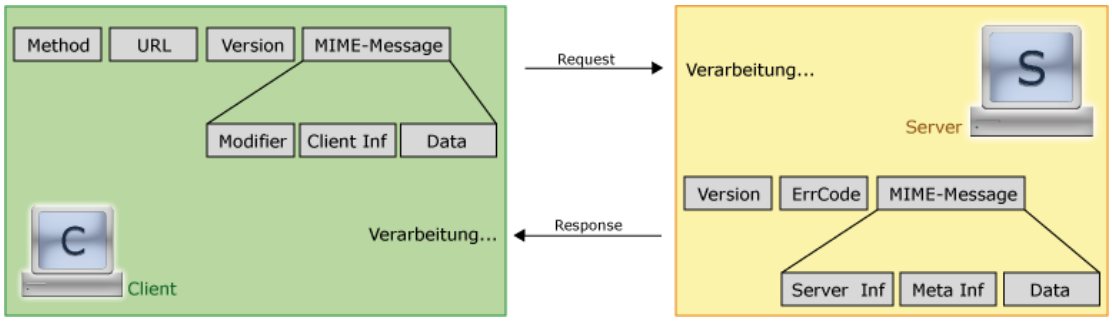
Als Transportprotokoll für HTTP wird in den allermeisten Fällen TCP verwendet, es ist jedoch nicht daran gebunden. ([HENN2003] S. 347ff)

Das Hypertext Transfer Protocol ist ein zustandsloses Protokoll, d.h. dass, sobald ein Dateitransfer zu Ende ist, die Verbindung zwischen den beiden Kommunikationspartnern beendet wird. Für jeden Transfer muss demnach eine eigene Verbindung aufgebaut werden. Die Verwaltung von Daten, die über mehrere Transfers verfügbar sein sollen, ist daher schwierig und wird entweder serverseitig (z.B. Session-Variablen) und/oder clientseitig (z.B. Cookies) realisiert. ([HENN2003] S. 347ff)

Meldungen, die mit dem HTTP-Standard versandt werden, sind in zwei Teile geteilt, die durch eine Leerzeile getrennt sind. Zuerst kommt ein Header, in dem wichtige Informationen, wie die verwendete Protokollversion, die Request-Methode, Statusmeldungen u.a. enthalten sind. Danach folgt optional noch ein Body-Teil. ([HENN2003] S. 347ff)

Abbildung 8.7–a zeigt die Datenstruktur eines HTTP-Request/Response-Paares.

Abbildung 8.7–a Datenstruktur eines HTTP-Request/Response-Paares ([HENN2003] S. 348)



Um mittels HTTP-Request Daten zu versenden, können zwei verschiedene Methoden verwendet werden: GET und POST. ([HENN2003] S. 347ff)

8.7.1.1 HTTP-GET

Im Falle der GET-Methode, werden so genannte Schlüssel-Werte-Paare („`key=value`“) mit einem Fragezeichen („`?`“) an die Server-URI im HTTP-Header angehängt und so übertragen. Sind mehrere Wertepaare vorhanden, werden diese durch „kaufmännische Und“ („`&`“) getrennt.

Das Problem hierbei ist, dass die URI-Länge mit 255 Zeichen begrenzt ist, und daher nicht beliebig viele Schlüssel-Werte-Paare versandt werden können. Außerdem können mit der GET-Methode nur reine Textdaten übertragen werden. ([HENN2003] S. 347ff)

8.7.1.2 HTTP-POST

Das Gegenstück zu GET ist die POST-Methode. Hierbei werden Daten nicht im Header mit gesendet, sondern im Body-Teil übertragen. Das hat den Vorteil, dass keine Längeneinschränkung vorliegt und auch Daten übertragen werden können, die nicht im Textformat vorliegen. ([HENN2003] S. 347ff)

Die `FileReference`-Klasse (siehe *Kapitel 8.7.2, Die FileReference-Klasse*) in Flash, mit der die Dateiübertragung in V.watch realisiert ist, benutzt die HTTP-POST-Methode.

Für die Realisierung des Dateiversands in V.watch ist daher serverseitig eine Programmiersprache notwendig, die HTTP-POST-Requests verarbeiten kann. Aufgrund unserer langjährigen Erfahrungen mit der freien Skriptsprache PHP haben wir uns für diese entschieden. Dieses PHP-Skript tut beim Aufruf nichts anderes, als dass es den POST-Request ausliest und, falls darin eine Datei vorhanden ist, diese in das richtige Verzeichnis verschiebt.

8.7.2 Die FileReference-Klasse

Die `FileReference`-Klasse stellt eine Möglichkeit dar, mittels HTTP oder HTTPS Dateien zwischen dem Computer eines Benutzers und einem Server hoch- bzw. herunter zu laden. Sie ist seit dem Flash Player 8 verfügbar.

Alle Informationen zur `FileReference`-Klasse sind online in den Adobe LiveDocs [ADOB2005d] verfügbar und ebendort entnommen.

Leider ist es nicht möglich, direkte Verbindungen zwischen zwei Client-Rechnern herzustellen. Aus diesem Grund müssen wir beim Dateiaustausch in V.watch den Server als Zwischenspeicher benutzen.

Nachfolgende Tabelle listet einige Methoden und Eigenschaften der FileReference-Klasse auf, die wir bei unserer Arbeit an V.watch benötigt haben.

Tabelle 8.7-a Methoden und Eigenschaften der FileReference-Klasse in ActionScript ([ADOB2005d])

Methode/Eigenschaft	Beschreibung
FileReference.browse([typelist:Array])	Zeigt ein Dialogfeld für die Dateisuche an, mit dem der Benutzer eine lokal gespeicherte Datei für einen Upload-Vorgang auswählen kann. Das Dialogfeld stammt aus dem Betriebssystem des Benutzers.
FileReference.cancel()	Bricht alle laufenden Updoad- bzw. Download-Vorgänge dieses FileReference-Objekts ab.
FileReference.download(url:String, [defaultFileName:String])	Ruft ein Dialogfeld auf, mit dem der Benutzer eine Datei von einem externen Server herunterladen kann. Mit Flash Player können Dateien bis zu einer Größe von 100 MB herunter geladen werden.
FileReference.name:String [schreibgeschützt]	Der Name der Datei auf der lokalen Festplatte.
FileReference.size:Number [schreibgeschützt]	Die Größe der Datei auf der lokalen Festplatte (in Byte).

FileReference.type:String [schreibgeschützt]	Der Dateityp. Auf Windows-Systemen ist diese Eigenschaft die Dateinamenerweiterung. Auf Macintosh-Systemen ist diese Eigenschaft ein aus vier Zeichen bestehender Dateityp.
FileReference.upload(url:String)	Beginnt mit dem Upload einer vom Benutzer auf einem externen Server ausgewählten Datei. Mit Flash Player können Dateien bis zu einer Größe von 100 MB hochgeladen werden.

Da der Flash Player ja systemunabhängig funktioniert und die verschiedenen Betriebssysteme aber oft unterschiedliche Dateisysteme verwenden, muss bei der Verwendung der `FileReference`-Klasse beachtet werden, dass einige Datei-Attribute, wie z.B. der Datei- oder der Erstellertyp, in unterschiedlichen Laufzeitumgebungen auch unterschiedlich sein können.

In V.watch sind nur die Attribute Dateiname und –größe von Bedeutung, weshalb auf Erstellertyp, etc. keine Rücksicht genommen werden muss. Als Dateiname wird automatisch der tatsächliche Name der Datei plus dem Dateityp angezeigt.

Sehr wichtig bei der Arbeit mit der `FileReference`-Klasse sind die Ereignis-`Listener`, mit denen auf Benutzereingaben (`FileReference.onCancel`, `FileReference.onOpen`, `FileReference.onSelect`), Fehler beim Transfer (`FileReference.onHTTPError`, `FileReference.onIOError`, `FileReference.onSecurityError`) und den Fortschritt des Vorgangs (`FileReference.onComplete`, `FileReference.onProgress`) reagiert werden können. Nachfolgend sind alle Ereignis-`Listener` der Klasse tabellarisch aufgelistet.

Tabelle 8.7-b Ereignis-Listener der FileReference-Klasse in ActionScript ([ADOB2005d])

Ereignis-Listener	Beschreibung
FileReference.onCancel	Wird aufgerufen, wenn der Benutzer das Dialogfeld für die Dateisuche per Mausklick oder ESC-Taste schließt.
FileReference.onOpen	Wird aufgerufen, wenn der Upload bzw. Download einer Datei gestartet wird.
FileReference.onSelect	Wird aufgerufen, wenn der Benutzer in einem Dialogfeld für die Dateisuche eine Datei für einen Upload- oder Download-Vorgang auswählt. Sobald ein Benutzer eine Datei auswählt und den Vorgang bestätigt (z. B. durch Klicken auf OK), werden die Eigenschaftenfelder des FileReference-Objekts mit Daten gefüllt.
FileReference.onHTTPError	Wird aufgerufen, wenn ein Upload wegen eines HTTP-Fehlers abgebrochen wird.
FileReference.onIOError	Dieser Listener wird aufgerufen, wenn der Upload bzw. Download aus einem der folgenden Gründe fehlschlägt: • Lese/Schreibe/Übertragungsvorgang durch Ein/Ausgabefehler abgebrochen • Server benötigt Authentifizierung • Ungültiges Protokoll für Übertragung (nur HTTP & HTTPS zulässig)
FileReference.onSecurityError	Wird aufgerufen, wenn der Upload bzw. Download einer Datei wegen eines Sicherheitsfehlers abgebrochen wird. Mit der aufrufenden SWF-Datei wurde unter Umständen versucht, ohne entsprechende Berechtigung auf eine SWF-Datei außerhalb ihrer Domäne zuzugreifen.
FileReference.onComplete	Wird aufgerufen, wenn der Upload bzw. Download einer Datei wurde erfolgreich beendet. Unter "erfolgreich beendet" wird hierbei verstanden, dass die gesamte Datei hoch- bzw. herunter geladen wurde.
FileReference.onProgress	Wird regelmäßig während des Upload- bzw. Download-Vorgangs aufgerufen.

8.7.3

8.7.4 Mit V.watch Dateien versenden

Befindet sich ein Benutzer in der Übersicht, so kann er einen oder mehrere andere Mitbenutzer markieren und selbigem eine Datei von seiner Festplatte schicken, indem er auf einen Button in der Steuerungsleiste klickt. Direkt darauf wird ein Dialogfeld angezeigt, in dem der Benutzer die gewünschte Datei auswählen kann.

Erlaubt sind Dateien, die kleiner als 20 MB groß sind – eine Beschränkung bezüglich der Dateitypen gibt es nicht. Eine Obergrenze der Dateigröße gibt es deshalb, um die Speicherressourcen des Servers nicht zu überlasten.

Sobald der Benutzer eine Datei ausgewählt hat, wird an alle potentiellen Empfänger eine Anfrage gesandt. Zusätzlich legt der Server eine Transferliste mit allen (potentiellen) Empfängern, dem Dateinamen und dem Startzeitpunkt der Aktion als Timestamp an.

Lehnt ein Benutzer das Annehmen der Datei ab, so wird sein Name aus der Transferliste am Server entfernt und der Sender bekommt eine Rückmeldung. Wenn alle Benutzer entweder angenommen oder abgelehnt haben, beginnt das Skript des Senders, die ausgewählte Datei auf den Server zu laden. Ist dieser Vorgang abgeschlossen, beginnen alle Empfänger, die vorher zugestimmt haben, automatisch mit dem Download. Ist ein Download abgeschlossen oder trat dabei ein Fehler auf, wird der entsprechende Benutzername aus der Transferliste am Server gestrichen. Ist eine dieser Listen leer, so wird sie gemeinsam mit der zugehörigen Datei vom Server gelöscht.

Für den Benutzer sieht es so aus, als würde dieser komplette Vorgang direkt zwischen den Client-Computern ablaufen. In Wirklichkeit agiert der Server aber als Organisationseinheit und Zwischenspeicher.

8.7.5 Mit V.watch Dateien in Konferenzen freigeben

Zusätzlich zum Versenden von Dateien an einzelne Benutzer, haben Teilnehmer von Konferenzen in V.watch noch die Möglichkeit, ihre Dateien den anderen Teilnehmern zur Verfügung zu stellen.

Das Prinzip ist hier ähnlich wie beim Dateiversand – nur etwas unkomplizierter. Mit einem Klick auf den Button „Datei hochladen", der bei allen Konferenzteilnehmern angezeigt wird, öffnet sich ein Dialog, in dem eine Datei ausgewählt werden kann. Auch hier gelten dieselben Beschränkungen wie beim Versenden von Dateien.

Wird eine Datei ausgewählt und bestätigt, startet der Upload-Vorgang und sobald dieser abgeschlossen ist, kann die entsprechende Datei in der Dateiliste der Konferenz eingesehen und herunter geladen werden. Dazu ist keinerlei Rücksprache mit dem Benutzer notwendig, der die Datei zur Verfügung gestellt hat.

Mit dieser Funktion soll unsere Software die Kommunikation in Konferenzen noch weiter erleichtern und den Austausch der Teilnehmer untereinander unterstützen.

9 ZUSAMMENFASSUNG UND AUSBLICK

Zusammenfassend kann man sagen, dass die Arbeit an V.watch uns als Einzelpersonen und als Team einiges gebracht hat. Wir haben im Laufe des Projekts alle geplanten Funktionalitäten und einige Extras realisert und sind dabei, besonders zum Ende hin, öfters auf unsere eigenen und die Grenzen von Flash gestoßen.

Trotz alledem sind immer noch optionale Ziele und Ideen offen, die bereits von uns angedacht, aufgrund von Zeitmangel aber nicht umgesetzt wurden. Dazu zählt zum Beispiel eine V.watch-Version für mobile Endgeräte, wie Handys, Smartphones und PDAs. Zwar gibt es bereits eine Flash Software für solche Geräte, Flash Lite 2.0, diese bietet aber leider noch keine Unterstützung von live Flash Video und remote SharedObjects, was für unser Programm natürlich essenziell ist. Eine neue Version von Flash Lite, die entsprechende Funktionalitäten unterstützen wird, ist von Adobe für Mitte des Jahres 2007 angekündigt. Auch mit der neuen Version von Flash, die ebenfalls noch in diesem Jahr auf den Markt kommen soll, sollten sich einige Verbesserungen im Bereich Flash Video und Kommunikation mit dem Flash Media Server ergeben.

Unser Plan ist es, auch nach Abschluss der Diplomarbeit gemeinsam weiter an V.watch zu arbeiten und das Programm unter Umständen sogar zur Marktreife zu bringen. Die Ergebnisse dieses Projekts sind dafür sicherlich eine sehr gute Grundlage, auf der sich aufbauen lässt.

GLOSSAR

Begriff	Erklärung
Add-On	„Erweiterung"; Optionales Modul, welches eine Hard- oder Software ergänzt und/oder erweitert.
Adobe Illustrator	Vektorbasiertes Grafikprogramm der Firma Adobe.
Adobe Photoshop	Pixelbasiertes Grafikprogramm der Firma Adobe.
API	„Application Programming Interface"; Ist ein Sammelbegriff und beschreibt die Eigenschaften, Ereignisse und auch die Methoden einer Klasse.
AS	„ActionScript"; proprietäre Programmiersprache von Adobe Systems für die Verwendung in Flash
Bühne	Bezeichnung für die sichtbare Fläche eines Flash-Films.
Chat-History	„Unterhaltungs-Verlauf"; Eine Dokumentation der Kommunikation, die über ein textbasiertes Unterhaltungswerkzeug vollzogen wurde. Es werden meist die Uhrzeit, der Verfasser und der Text der einzelnen Nachrichten gespeichert.
Cookies	Textdateien, die von Internetbrowsern zur lokalen Speicherung von Daten (z.B. Login-Daten einer Website) verwendet werden.
Digicam	Bezeichnung für handelsübliche Digitalkameras.
DMZ	„Demilitarized Zone", auch „Entmilitarisierte Zone"; Bezeichnet ein Computernetzwerk mit Zugriffsmöglichkeiten auf die darin angeschlossenen Server. Die Server in einer DMZ werden durch Firewalls gegen andere Netze abgeschirmt. Diese Trennung ermöglicht es, Dienste für das Internet freizugeben und gleichzeitig das interne Netzwerk vor Zugriffen zu schützen.
DV	„Digital Video"; Ist der Oberbegriff für den DV-Standard, der 1994 eingeführt wurde. Es umfasst u.a. die Kassettenformate DV, MiniDV, DVCAM und Digital8.
FLA-Datei	„Flash-Datei"; Ein in Adobe's Macromedia Flash erstelltes Dokument.
Flash-Projektor	Ein mit der Dateierweiterung .exe (Windows) bzw. .hqx (Macintosh) bezeichnetes Dateiformat, das eine Kombination aus einem SWF (Shock Wave File) und einem integrierten Adobe's Flash Player ist. Somit können Shock-Wave-Dateien auch auf Rechnern abgespielt werden, die keinen Flash Player installiert haben.
FMS	„Flash Media Server"; Produkt der Firma Adobe Systems. Derzeit in der Version 2 (FMS2) verfügbar.

full duplex	Bezeichnet die Übertragung von Daten in zwei Richtungen, wenn diese zeitgleich möglich ist.
GUI	„Graphical User Interface", auch „grafische Benutzerschnittstelle"; Bezeichnet eine Softwarekomponente über deren grafische Elemente die Benutzer mit der Maschine interagieren können.
HDTV	„High Definition Television"; Hoch auflösendes Fernsehen (1920x1080 Pixel)
HTTP	„Hypertext Transfer Protocol"; Ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Siehe *Kapitel 8.7.1, HTTP*.
Icon	„Zeichen"; Eine Miniaturabbildung, die oft auf Schaltflächen in Programmen eingesetzt wird. Der Benutzer sieht anhand der Grafik, welchen Zweck das Interaktionselement erfüllt. Außerdem oft verwendet, um Eigenschaften deutlicher und ansprechender darzustellen. Zum Beispiel bei Email-Programmen erscheint ein kleines Briefsymbol, um zu zeigen, dass der Benutzer eine neue Nachricht bekommen hat.
ID3-Tag	Metainformationen, die in MP3-Dateien eingebunden werden können. Können z.B. Titel, Interpret, Albumtitel, Länge etc. enthalten.
half duplex	Bezeichnet die Übertragung von Daten in zwei Richtungen, wenn diese nicht zeitgleich möglich ist.
Hash	Auch „Streuwert"; Eingabe aus einer üblicherweise großen Quellmenge erzeugt eine Ausgabe aus einer im Allgemeinen kleineren Zielmenge. Hash-Funktionen sind gut zum Verschlüsseln von Daten geeignet, da von einem Hash nicht ohne enormen Aufwand auf die Quellmenge geschlossen werden kann. Beispiele: MD5, SHA1.
Instanz	Auch „Objekt"; Ist ein Exemplar einer Klasse, mit deren Eigenschaften und Methoden gearbeitet werden kann. Erst mit der Initiierung einer Instanz einer Klasse wird diese auch erst verwendet.
Leerstring	Eine Variable, die als Wert zwei aufeinanderfolgende Auslassungszeichen im Programmcode festgelegt bekommen hat. Sobald ein Zeichen (auch ein Leerzeichen) zwischen diesen Auslassungszeichen steht, kann die Variable nicht mehr als Leerstring bezeichnet werden.
Pixel	Mischung aus den englischen Worten „Picture" (vgl. „Pix") und „Element". Die kleinste Einheit einer digitalen Grafik, auch Bildpunkte genannt. Eine solche Grafik ist durch eine Abfolge von verschieden eingefärbten Pixeln aufgebaut. Durch Gesamtbetrachtung des Bildes sind die Pixel nicht mehr erkennbar und es wirkt wie eine Einheit.

Pop-Up-Fenster	„to pop up“: engl. für „(plötzlich) auftauchen“; Ein sich plötzlich öffnendes Programmfenster, das wichtigen Inhalt für den Benutzer interessant machen soll. Wird oft für Werbung auf Webseiten missbraucht. Das plötzliche Auftauchen der Fenster kann mit so genannten Pop-Up-Blockern verhindert werden.
Port	Sind Adresskomponenten und werden in Netzwerkprotokollen eingesetzt, um Daten den richtigen Diensten zuzuordnen.
PRT-Time	„Ping-Round-Trip-Time“; Ist die Zeit in Millisekunden, die ein Ping-Request-Response-Zyklus dauert, also bis das Signal vom Server zum Client gesendet, vom Client wieder zum Server zurück geschickt und dort vollständig empfangen wurde.
Remote	Engl. für „fern“. Gegenteil zu „local“ (engl. für „lokal“); Bezeichnet einen Zugriff aus der Ferne. Beispiel: remote SharedObject oder Windows Remote-Desktop.
Root-Element	„Wurzel-Element“; Das in der Hierarchie am höchsten stehende Element. Von ihm gehen alle anderen Elemente aus.
RTMP	„Real Time Messaging Protocol“; Proprietäres Protokoll von Adobe zur Kommunikation zwischen Flash Player und Flash Media Server.
Sampling	Abtastung beim Digitalisieren von Audio- und/oder Videodaten.
Screenshot	„Bildschirmabzug/-abdruck“; Ein exaktes Abbild eines Computerbildschirmes. Kann mit der DRUCK-Taste auf der Tastatur durchgeführt werden. Hierbei wird die Grafik in die Zwischenablage kopiert und kann später in jedem beliebigen Grafikprogramm eingefügt werden. Wird oft für Hilfestellungen in Computerfragen, zur Veranschaulichung, verwendet.
SDTV	„Standard Television“; „Gewöhnliches“ Fernsehen.
simplex	Bezeichnet die Übertragung von Daten, wenn diese nur in eine Richtung möglich ist.
SO	„SharedObject“; Möglichkeit zur persistenten Speicherung von Daten mit Flash. Kann mit Browser-Cookies verglichen werden.
Socket	Ist eine Schnittstelle zur Netzwerk-Kommunikation in beide Richtungen zwischen zwei Programmen. Ein Socket bildet eine Schnittstelle zwischen dem Netzwerkprotokoll des Betriebssystems und der eigentlichen Applikation.
Stream, streamen, Streaming	„Data Stream“; Engl. für „Dateistrom“; Video/Audio-Daten werden zur Laufzeit empfangen und nahezu gleichzeitig wiedergegeben.

Tag	„Auszeichner", „Marke", „Etikett"; Beschreibt Teile eines Dokumentes und legt deren Formatierung fest. Wird in Auszeichnungssprachen wie HTML und XML verwendet. Mit Spitzklammern (<,>) werden Tags definiert. Sie müssen vor dem zu beschreibenden Teil geöffnet (<tagname>) und danach wieder geschlossen (</tagname>) werden. Die dazugehörige Auszeichnungssprache interpretiert den Tag-Namen und führt die dazugehörige Formatierung aus.
Thumbnail	„Miniaturansicht"; Kleine Version eines Bildes, die zur Vorschau auf das Original dient.
URI	„Uniform Resource Identifier"; Bezeichner zur Identifizierung einer abstrakten oder physischen Ressource.
XML	„Extensible Markup Language"; Auszeichnungssprache. W3C-Standard.
XML-Baum	Hierarchische Struktur eines XML-Dokuments.
XPath-API	Eine Klasse in Flash, mit der das Suchen in `XML`-Objekten schneller von statten geht. Mit ihren Funktionen können XML-Knoten und Attribute direkt, ohne langwierige Pfadangaben (z.B. `rootElement.firstChild.firstChild.lastChild`), angesprochen werden. ([ADOB2005d])
YC_RC_B	„Farb-Komponentenmodell"; Besteht aus den drei Komponenten Y (Luminanz = Graustufen bzw. Helligkeit), C_R (Blau minus Y) und C_B (Rot minus Y).

QUELLENVERZEICHNIS

Literaturverzeichnis

[BRUN2005] Bruns, Kai & Meyer-Wegener, Klaus, *Taschenbuch der Medieninformatik*, 2005, 1. Auflage, Fachbuchverlag Leipzig im Carl Hanser Verlag, Leipzig

[HENN2003] Henning, A. Peter, *Taschebuch Multimedia*, 2003, 3. Auflage, Fachbuchverlag Leipzig im Carl Hanser Verlag, Leipzig

[KOPE1997] Kopetz, Hermann, *Real Time Systems – Design Principles for Distributed Embedded Applications*, 1997, 1. Auflage, Kluwer Academic Publishers Group, Dordrecht

[SCHW2003] Schweibenz, Werner. & Thissen, Frank, *Qualität im Web – Benutzerfreundliche Webseiten durch Usability Evaluation*, 2003, 1. Auflage, Springer-Verlag, Berlin

Verzeichnis von Quellen aus dem Internet

[ADOB2002] *Adobe - TechNote : HTTP Tunneling protocols* [online], aktualisiert am 25.10.2003 [zitiert am 10.04.2007], verfügbar auf beigelegter CD (Verzeichnis „HTTP_tunneling_protocols“) oder im Internet: http://www.adobe.com/go/tn_16631

[ADOB2003a] *Flash MX 2004 – Komponenten verwenden* [PDF], erste Auflage im Oktober 2003 [zitiert am 03.04.2007], verfügbar auf beigelegter CD (Flash_Komponenten_verwenden.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/de/flash/mx2004/fl_using_components.pdf

[ADOB2004a] *Flash Video Lernhandbuch* [online], aktualisiert am 12.09.2005 [zitiert am 14.03.2007], verfügbar auf beigelegter CD (Verzeichnis „flash8_devnet") oder im Internet: http://www.adobe.com/de/devnet/flash/articles/video_guide.html

[ADOB2004b] *Encoding Best Practices for Prerecorded Flash Video* [online], aktualisiert am 02.09.2004 [zitiert am 25.03.2007], verfügbar auf beigelegter CD (Verzeichnis „flash8_devnet") oder im Internet: http://www.adobe.com/devnet/flash/articles/flv_encoding_02.html

[ADOB2005a] *Flash 8 – Komponenten verwenden* [PDF], erste Auflage im September 2005, [zitisert am 10.04.2007], verfügbar auf beigelegter CD (fl8_using_components.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/de/flash/mx2004/fl_using_components.pdf

[ADOB2005b] *Flash 8 – Komponenten Referenzhandbuch* [PDF], erste Auflage im September 2005, [zitiert am 10.04.2007], verfügbar auf beigelegter CD (fl8_clr.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/de/flash/fl8/fl8_clr.pdf

[ADOB2005c] *Developing Media Applications* [PDF], erste Auflage im Oktober 2005 [zitiert am 10.03.2007], verfügbar auf beigelegter CD (flashmediaserver_developing.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_developing.pdf

[ADOB2005d] *Flash 8 LiveDocs* [online], [zitiert am 25.03.2007], verfügbar auf beigelegter CD (Verzeichnis „flash8_docs“) oder im Internet: http://livedocs.adobe.com/flash/8_de/

[ADOB2005e] *Flash Media Server 2 LiveDocs* [online], [zitiert am 25.03.2007], verfügbar auf beigelegter CD (Verzeichnis „fms2_docs“) oder im Internet: http://livedocs.adobe.com/fms/2/docs

[ADOB2005f] *Client-Side ActionScript Language Reference For Flash Media Server* [PDF], erste Auflage im Oktober 2005 [zitiert am 10.03.2007], verfügbar auf beigelegter CD (flashmediaserver_cs_asd.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_cs_asd.pdf

[ADOB2005g] *Server-Side ActionScript Language Reference For Flash Media Server* [PDF], erste Auflage im Oktober 2005 [zitiert am 10.03.2007], verfügbar auf beigelegter CD (flashmediaserver_ss_asd.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_ss_asd.pdf

[ADOB2005h] *Using Flash Media Server Edge Servers* [PDF], erste Auflage im Oktober 2005 [zitiert am 10.03.2007], verfügbar auf beigelegter CD (flashmediaserver_edge.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_edge.pdf

[ADOB2005i] *Server Management ActionScript Language Reference* [PDF], erste Auflage im Oktober 2005 [zitiert am 11.04.2007], verfügbar auf beigelegter CD (flashmediaserver_mgmt_asd.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_mgmt_asd.pdf

[ADOB2005j] *Managing Flash Media Server* [PDF], erste Auflage im Oktober 2005 [zitiert am 10.03.2007], verfügbar auf beigelegter CD (flashmediaserver_managing.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_managing.pdf

[ADOB2005k] *Using Flash Media Server Components* [PDF], erste Auflage im Oktober 2005 [zitiert am 10.03.2007], verfügbar auf beigelegter CD (flashmediaserver_components.pdf) oder im Internet: http://download.macromedia.com/pub/documentation/en/flashmediaserver/2/flashmediaserver_components.pdf

[ADOB2006a] *Flash Player Penetration* [online], [zitiert am 25.03.2007], verfügbar auf beigelegter CD (Verzeichnis „flashplayer_study") oder im Internet: http://www.adobe.com/products/player_census/flashplayer/

[ADOB2006b] *Methodology for Adobe plug-in technology study* [online], [zitiert am 25.03.2007], verfügbar auf beigelegter CD (Verzeichnis „flashplayer_study") oder im Internet: http://www.adobe.com/products/player_census/methodology/

[ADOB2006c] *Adobe - Flash Media Server 2 : Features* [online], [zitiert am 08.04.2007], verfügbar auf beigelegter CD (Verzeichnis „fms2_features") oder im Internet: http://www.adobe.com/uk/products/flashmediaserver/productinfo/features/

[ATMI2007] *at-mix – Rich Media* [online], aktualisiert am 19.04.2004, [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „rich_media") oder im Internet: http://www.at-mix.de/rich_media.htm

[COMP2007] *Computer-Wörterbuch - Conferenzing* [online], aktualisiert am 26.02.2007, [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „conferencing") oder im Internet: http://www.computer-woerterbuch.de/?id=suchen&comDB=woerterbuch&suchen=Conferencing

[DÜNH1998] *Kuno Dünhölter - Das Web automatisieren mit XML* [online], aktualisiert am 01.09.1998, [zitiert am 16.04.2007, 02:54 MESZ], verfügbar im Internet:

http://members.aol.com/xmldoku/

[ELEC2006] *ElectroServer 3 - Features* [online], [zitiert am 08.04.2007], verfügbar auf beigelegter CD (Verzeichnis „electro_server") oder im Internet: http://www.gruenderblatt.de/logo-artikel114.html

[FOCU2006] *Focus online – Video on Demand* [online], aktualisiert am 10.05.2006, [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „video_on_demand") oder im Internet: http://www.focus.de/digital/internet/video_on_demand

[GRAF2005] *Fit für den Markt* [online], aktualisiert am 27.07.2006, [zitiert am 12.04.2007], verfügbar auf beigelegter CD (Verzeichnis „gruenderblatt") oder im Internet: http://www.gruenderblatt.de/logo-artikel114.html

[INTE2006] *Internet-Manual – Streaming Media* [online], [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „streaming_media") oder im Internet: http://www.internet-manual.de/streaming-media.htm

[KUBL2002] *Kublikon – Streaming Media Technology* [online], [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „streaming_media_technology") oder im Internet: http://www.kublikon.de/lexikon/html/s.html

[LRZ2007] *LRZ – Live-Streaming mit dem QuickTime Broadcaster* [online], aktualisiert am 06.02.2007, [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „live_streaming") oder im Internet: http://www.lrz-muenchen.de/services/peripherie/livestream/

[ON22006] *White Paper – Advantages of On2 VP6 Technology* [PDF], aktualisiert am 20.10.2006 [zitiert am 14.03.2007], verfügbar auf beigelegter CD (on2vp6_whitepaper.pdf) oder im Internet: http://www.on2.com/cms-data/pdf/1125607149174329.pdf

[OREG2006] *Oregano Multiuser Server - Features* [online], [zitiert am 08.04.2007], verfügbar auf beigelegter CD (Verzeichnis „oregano_server") oder im Internet: http://www.oregano-server.org/features.htm

[OSFL2005] *OSFlash - Red5 : FAQ* [SWF], aktualisiert am 24.10.2005, [zitiert am 11.04.2007], verfügbar auf beigelegter CD (Verzeichnis „osflash_red5_faq") oder im Internet: http://svn1.cvsdude.com/osflash/red5/doc/trunk/Frequently%20Asked%20Questions.swf

[OSFL2007] *OSFlash - Red5:Open Source Flash Server* [online], aktualisiert am 11.04.2007, [zitiert am 11.04.2007], verfügbar auf beigelegter CD (Verzeichnis „osflash_red5") oder im Internet: http://osflash.org/red5/

[SEIL1998a] *Die Farbe Grün* [online], aktualisiert am 23.03.2007, [zitiert am 12.04.2007], verfügbar auf beigelegter CD (Verzeichnis „seilnacht") im Internet: http://www.seilnacht.com/Lexikon/Gruen.htm

[SEIL1998b] *Die Farbe Blau* [online], aktualisiert am 23.03.2007, [zitiert am 12.04.2007], verfügbar auf beigelegter CD (Verzeichnis „seilnacht") im Internet: http://www.seilnacht.com/Lexikon/FBlau.htm

[SORE2005] *What's the diffence between the Sorenson Spark (Standard) and Sorenson Spark Pro codecs?* [online], aktualisiert am 15.07.2005 [zitiert am 14.03.2007], verfügbar auf beigelegter CD (sorenson.htm) oder im Internet: http://portal.knowledgebase.net/display/2n/index.asp?c=2970&cpc=6JUM8X432mIp4c25a374g8SsnN3QtCGpBQdU&cid=7546&cat=&catURL=&r=0.9318201

[TECH2005a] *TechWeb – Streaming Video* [online], [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „streaming_video") oder im Internet: http://www.techweb.com/encyclopedia/defineterm.jhtml?term=streamingvideo

[TECH2005b] *TechWeb - Video Instant Messaging* [online], [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „video_instant_messaging") oder im Internet: http://www.techweb.com/encyclopedia/defineterm.jhtml?term=video+instant+messaging

[UNIT2006] *Unity 2 - Features* [online], [zitiert am 08.04.2007], verfügbar auf beigelegter CD (Verzeichnis „unity_server") oder im Internet: http://www.moock.org/unity/u2mdk/

[VIDE2006] *Video on Demand – Grundlagen* [online], [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „video_on_demand_vod") oder im Internet: http://www.video-on-demand.info/grundlagen.php

[WHAT2006] *What is Vlog* [online], aktualisiert am 07.04.2006, [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „video_blog") oder im Internet: http://whatis.techtarget.com/definition/0,,sid9_gci1163516,00.html

[WEBR2006] *MP2 – Webradio Technik* [online], [zitiert am 05.04.2007], verfügbar auf beigelegter CD (Verzeichnis „webradio") oder im Internet: http://www.mp2.de/technik.html

ANHANG

Im Anhang ist der von uns speziell für unser Programm entwickelte Usability-Test zu finden. Der Test dient zur Verbesserung des User Interface von V.watch.