Fachhochschule Köln
Cologne University of Applied Sciences

University of Applied Sciences Cologne
Fakultät für Informations- , Medien- und Elektrotechnik

Diplomarbeit

Webbasiertes Live-Streaming auf einer iTV Testplattform

vorgelegt von
Volker Lux

Betreuer:
Prof. Dr. rer. nat. Stefan Grünvogel, Fachbereich Computeranimation und Datenverarbeitung

Prof. Dr. Klaus Ruelberg, Fachbereich Fernseh- und audiovisuelle Medientechnik

Köln, im Januar 2009

ABSTRACT

Diese Arbeit befasst sich mit der Thematik des Live-Streamings und webbasierter Medienübertragung. Im Fokus steht in diesem Zusammenhang der Adobe Flash Media Server 3.0 als Streaming-Plattform sowie Adobe Flash CS3 als Entwicklungsumgebung. Neben den theoretischen Grundlagen wird die praktischen Umsetzung in Form einer iTV^{G}-Applikation für einen exemplarischen Anwendungszweck eingehend erläutert. Dabei werden sowohl der technische Hintergrund der Programmierung wie auch die Methoden und Prinzipien des interaktiven Designs, der Usability, diskutiert und in den Entwicklungsprozess integriert. Die Flash-Media Applikation wird abschließend einer kritischen Betrachtung unterzogen, um mögliche Fehlerquellen sowie weitere Entwicklungspotentiale aufzuzeigen.

This dissertation examines the topic of livestreaming and web based media delivery. It is focussed on the Adobe Flash Media Server 3.0 as streaming platform as well as Adobe Flash CS3 used as programming environment. Beside the theoretical basics in the beginning the final chapters specifiy a practical realization in the form of an iTV application for a custom purpose. In addition to the technical background of programming the methods and principles of usability were discussed and integrated in the further development process. Finally the Flash Media Application will be reviewed critically to reveal possible errors and prospective development potentials.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

API	Application Interface
AS	Actionscript
FMDS3	Flash Media Development Server, Version 3.0
FMIS3	Flash Media Interactive Server, Version 3.0
FMS3	Flash Media Server, Version 3.0
FMSS3	Flash Media Streaming Server, Version 3.0
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protokoll
MPEG	Motion Picture Expert Group

RSS	Really Simple Syndication (versch. Definitionen vorliegend)
RTMP	Real Time Media Protokoll
SSAS	Server-side Actionscript
UI	User Interface
URL	Universal Ressource Locator
VoD	Video-on-Demand
XML	Extensible Markup Language

Begriffe, die mit [G] gekennzeichnet sind werden im Glossar gesondert erklärt.

I EINLEITUNG

1 Ausgangssituation

Diese Arbeit befasst sich mit der Thematik der Videoübertragung, im speziellen der Live-Übertragung über das Internet. Das exponentiell wachsende Angebot an Video-On-Demand Inhalten und die stetige Weiterentwicklung von webbasierten Streaming-Servern, sowohl kommerzieller Art als auch von Open-Source[G] Systemen wie z.B. der Red5 Server, bieten Anlass, sich mit den Entwicklungen des Live-Streamings näher zu beschäftigen. Durch das wachsende Verteilungsgebiet an Breitbandinternetverbindungen und effektiveren Videokodierungssystemen werden die Einschränkungen für eine Übertragung von Live-Videos über das Internet geringer. Auf Grund der simplen Installationsroutinen findet sich die kostenfreie Flashplayer-Plattform auf einer Vielzahl heimischer Computer wieder (siehe Abbildung III-1). Zudem erfordern auch immer mehr kommerzielle und private Internetseiten die Installation eines solchen FlashPlayers. Der Entwicklung einer Anwendung basierend auf den gerade genannten Systemvoraussetzungen kann somit eine weitreichende Kompatibilität eingeräumt werden.

2 Zielsetzung

Das Ziel der Arbeit ist die praktische Analyse der Potentiale des Adobe Flash Media Servers im Hinblick auf die Videoübertragung. Dabei sollen nutzerorientierte Schnittstellen unter Berücksichtigung der Grundsätze des standardisierten Usability Designs (siehe Kapitel IV) geschaffen werden. Als wesentliches Arbeitsziel steht die Entwicklung einer Applikation im Vordergrund, die auf einem Flash-System mit Actionscript basiert und den Flash Media Development Server als Streaming-Plattform nutzt. In diesem Prototypen sollen insbesondere die Praxistauglichkeit von simultanen Streamings getestet und eventuelle Hindernisse sowie Optimierungsmöglichkeiten aufgezeigt werden. Bei der Methodik der Programmentwicklung wird unter Berücksichtigung des zeitlichen Rahmens eine Lösung angestrebt, die den wesentlichen Merkmalen der modernen Applikationsherstellung wie auch einer nutzerorientierten Gestaltung der Funktionalität und Oberfläche folgt.

3 Aufbau

Da mit dieser Arbeit auch eine praktische Umsetzung einer Applikation verbunden ist, sollen die angewandten Techniken und Hintergründe mit theoretischen Grundlagen ergänzt werden. Somit erklärt sich der stufenweise Aufbau dieser Arbeit. In Kapitel II Abschnitt 1 werden die derzeit verfügbaren Optionen einer Videoübertragung über das Internet aufgezeigt und differenziert. Daran anknüpfend befasst sich Kapitel II Abschnitt 2 mit gängigen Videocodecs, die insbesondere unter der Adobe Flash Plattform Verwendung finden. Die softwarebasierten Enkodierungsmöglichkeiten für Video-Streams sowie die Struktur beschreibender Zusatzinformationen werden in Kapitel II Abschnitt 3 näher betrachtet. Mit fortschreitendem Praxisbezug befasst sich Kapitel III im Detail mit den Eigenschaften des Flash Media Servers. Als Übergang vom technisch-funktionalen Hintergrund der Applikationsentwicklung schlägt Ka-

pitel IV die Brücke zu den Grundlagen einer sinnvollen und effektiven Gestaltung grafischer Benutzerschnittstellen. Daran anknüpfend wird in Kapitel V die Entwicklung des Prototypen als konkreter Realisierungsprozess der Flash Applikation erläutert. Neben der Herausstellung wesentlicher Inhalte und Prozesse der Entwicklungsphase widmet sich Kapitel V der Beschreibung des Funktionsumfangs der Beta-Version. Abschließend werden in einem Resumé die Erweiterungs- und Optimierungsmöglichkeiten der Applikation dargestellt und ein Ausblick auf die Entwicklungspotentiale des Systems gegeben.

II GRUNDLAGEN

1 Streaming Media Übertragungsverfahren

Zum Zeitpunkt der Anfertigung dieser Arbeit existieren drei grundlegende Übertragungsverfahren zur Vermittlung von Mediendateien über das Internet. Nachfolgend sollen diese Verfahren erläutert und deren Vor- und Nachteile gegenüber gestellt werden.

1.1 Progressive Download

Der sog. „Progressive Download“ wird von Adobe wie folgt beschrieben:

> *“When a video file is delivered from a standard web server, the video data is loaded using progressive download, meaning the video information loads in sequence. This has the benefit that the video can begin playing before the entire file is downloaded; however, it prevents you from jumping ahead to a part of the video that hasn't loaded.”*

[Quelle: http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000252.html, Zugriff: 05.12.2008]

Im Unterschied zum Streaming-Abspielverfahren ist beim progressiven Download zur Videodarstellung kein Media-Server notwendig. Die Display-Komponente der Videostreaming Anwendung lädt Videos vielmehr direkt über das HTTP-Protokoll[G] vom WebServer herunter. Nach einer kurzen Vorlaufzeit wird das Video im Display des Betrachter-PC’s angezeigt, während im Hintergrund der Ladeprozess fortschreitet. In dem bereits herunter geladenen Teil des Videos sind eine Navigation und ein Abspielen des Films möglich. In der Gänze kann ein Video aber erst nach Fertigstellung des kompletten Ladeprozesses betrachtet werden.

[Quellen: http://www.richinternet.de/.../Flash Media Server Consulting/Streaming vs. Progressive/, Zugriff: 05.12.2008; Quelle: http://www.vikar.de/uploads/media/StA_Fabrizio_Branca.pdf, Zugriff: 05.12.2008]

Tabelle II-1: Vor- und Nachteile des Progressive Download-Verfahrens.

Vorteile	Nachteile
Kein spezieller Videostreaming-Server notwendig	Player u. entsprechende Codecs müssen auf dem Client installiert werden
Navigation und Abspielen von bereits herunter geladenen Datenfragmenten möglich	Vollständige Videobetrachtung erst nach vollständigen Download möglich
Gesicherte Bildqualität, da keine dynamische Anpassung erfolgt	Video kann beim Betrachten ins Stocken geraten, wenn der Datendurchsatz die Bandbreite des Anschlusses übersteigt
	Video wird im Cache des Client-Rechners abgelegt > eventuelles Speicherproblem
	Eher geeignet für einen begrenzten Nutzerzugriff u. Anschlüsse mit hohen Bandbreiten

1.2 On-Demand Videostreaming

Videostreaming bezeichnet im Allgemeinen die Übermittlung von Audio-, Video und Daten-Inhalten, die auf einem Media- bzw. Streaming-Server gespeichert sind, zu einem Client[G]. Beim sog. On-Demand Videostreaming werden die auf einem Streaming-Server abgelegten Daten vom Client nach Bedarf (On-Demand) durch ein Anfrageprotokoll als Streams abgerufen. Dabei kommt ein sog. „publish-and-subscribe model" zum Einsatz. Während sowohl ein Server als auch ein Client einen Stream publizieren können, kann dieser ausschließlich vom Client betrachtet werden [vgl. Flash Media Server Documentation – Technical Overview].
Im Unterschied zum Progressive Download-Verfahren ist die eingesetzte Datenrate der Mediendateien und somit die Wiedergabequalität beim Videostreaming meist geringer. Allerdings kann durch die Verwendung bestimmter Protokolle eine dynamische Anpassung der Videoqualität an die Verbindungsgeschwindigkeit des Clients erreicht werden. Ebenso sollte das Kodierungsverfahren an die Übertragungsmethode angepasst werden (siehe Kapitel II Abschnitt 2.5). Das On-Demand Videostreaming ermöglicht die Ansteuerung einer bestimmten Szene, ohne dass das komplette Video geladen werden muss. Hier besteht ein entscheidender Vorteil dieser Methode im Hinblick auf die Anwenderfreundlichkeit.
Zu den gängigsten Wiedergabeprotokollen des On-Demand Videostreamings zählen das Real-Time-Streaming-Protokoll (RTSP) sowie das beim Adobe Flash-Media Server verwendete Real-Time-Media-Protokoll (RTMP).

Dem On-Demand Videostreaming liegen folgende Prozessabläufe zu Grunde:
Ein Webserver schickt die medialen Inhalte an den Browser des Client-Rechners. Zur Darstellung des Videos im Browser des Clients wird ein Browser Plug-In wie z.B. der Flash-Player von Adobe eingesetzt. Bei der Verbindung des Webservers mit dem Streaming Server wird auf das jeweilige Protokoll des Wiedergabeplayers zurückgegriffen. Im Fall des Flash Media Servers ist dies meist das RTMP Protokoll, welches eine dynamische Anpassung der Videoqualität an die Bandbreite ermöglicht.
Im Folgenden werden die Vor- und Nachteile des On-Demand Videostreamings noch einmal gegenüber gestellt:

[Quelle: http://livedocs.adobe.com/flashmediaserver/3.0/docs/help.html?content=Book_Part_28_tech_overview_1.html, Zugriff 20.12.2008]

Tabelle II-2: Vor- und Nachteile des On-Demand Videostreamings.

Vorteile	Nachteile
Videodaten werden an Verbindungsqualität angepasst und übertragen	Qualität wird dynamisch angepasst, was zu Betrachtungsverlusten führt
Navigieren im Video möglich	Es wird ein Streaming Server benötigt, der unter Umständen sehr teuer ist
Unterstützt eine hohe Anzahl von Betrachtern	
Betrachten von sehr langen Videos möglich	
Schnelles Starten der Videos möglich, nur kurze Vorlaufzeit	

[VGL. BÖSKEN, M. (2007), S. 32]

1.3 Live Videostreaming

Der Begriff Streaming bezeichnet im Allgemeinen das gleichzeitig Empfangen und Abspielen eines Mediums über das Internet. Dabei werden nur die benötigten Daten für einen bestimmten Zeitraum, der Länge des Puffers, übertragen. Die bekanntesten Vertreter der Live-Streaming-Technik sind Realmedia[G], WMV[G] und Flash Video.

[Quelle: http://www.vikar.de/uploads/media/StA_Fabrizio_Branca.pdf, Zugriff: 05.12.2008]

Das Live Videostreaming von medialen Inhalten erfordert wie das On-Demand Videostreaming einen speziellen Server, der auf das jeweilige Videoformat ausgerichtet ist. Beim Videostreaming mit Live-Inhalten wird genauso verfahren wie mit gespeicherten Videos. Der einzige Unterschied ist die Tatsache, dass die Videos von einer Live-Quelle (wie z.B. einer Studio-Kamera oder Webcam) unmittelbar in ein Format konvertiert werden (Enkodierung), welches dann von einem Streamingserver in Echtzeit als Stream bereitgestellt wird. Eine Wiedergabe von Live-Videos über die Progressive Download-Methode ist nicht möglich. Live Videostreaming erlaubt z.B. durch den kombinierten Einsatz eines Adobe Flash Media Live Encoders und des Flash Media Server 3 die 24 Stunden-Bereitstellung von Live-Videos auf Nachrichtenportalen oder Event- und Sportplattformen.

[Quelle: http://www.adobe.com/de/products/hdvideo/supported_technologies/streaming.html, Zugriff: 05.12.08]

Während das Progressive Download Verfahren - wie dargelegt - auf ein einfaches HTTP-Protokoll zur Datenübermittlung zurückgreift, setzen sowohl On-Demand wie auch Live Videostreaming das RTMP Protokoll zur Content Übermittlung ein.

1.4 Das RTMP Protokoll

Das RTMP (Real-Time-Messaging Protocol) ist ein von Adobe Systems entwickeltes proprietäres Netzwerkprotokoll, um in Echtzeit Video- und Audiodaten über eine TCP[G]-Verbindung oder einen HTTP Tunnel von einem Media Server zu einem Flash-Player zu transferieren. Eine einzelne Verbindung über RTMP kann mehrere Streams gleichzeitig übertragen. Möglich wird dies durch die Verwendung verschiedener Kanäle für die Datenpakete.

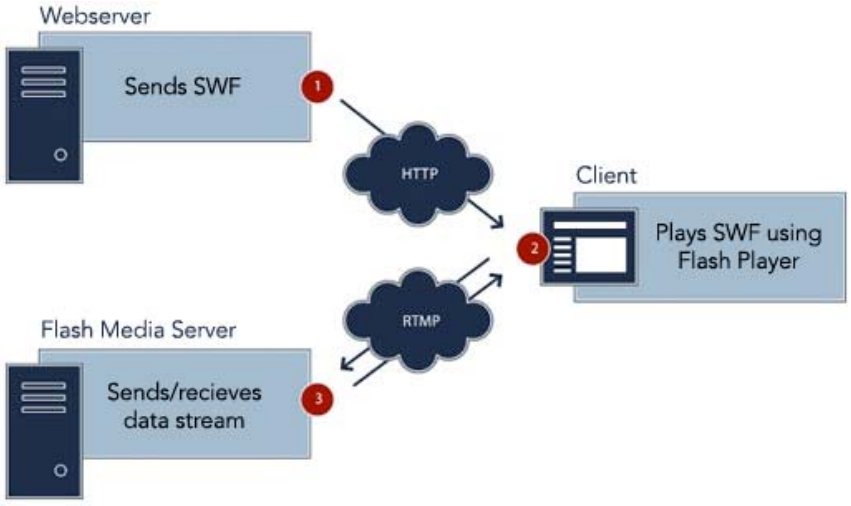


Abbildung II-1: Flash Media Server client/server Architektur.

[Quelle: http://www.adobe.com/devnet/flashmediaserver/articles/overview_streaming_fms3_02.html, Zugriff: 06.12.2008]

Der Flash Media Server 3 unterstützt fünf Arten von RTMP Verbindungen:

- **RTMP:** Standardisierter, unverschlüsselter Protokolltyp.
- **RTMPT:** Hierbei werden RTMP Daten in das HTTP Protokoll eingebettet. RTMPT ist im Grunde eine HTTP-„Hülle“ (Wrapper[G]) für das RTMP Protokoll, welches vom Client an den Server gesendet wird.
- **RTMPS:** Analog zum Standard RTMP Protokoll, allerdings durch eine SSL[G]-Verschlüsselung gesichert. SSL ist ein Sicherheitsprotokoll für eine Verbindung über TCP/IP. Der Flash Media Server unterstützt sowohl einkommende als auch ausgehende SSL-Verbindungen.
- **RTMPE:** Ebenfalls eine verschlüsselte Form von RTMP. RTMPE ist schneller als RTMPS besitzt jedoch schwächere Sicherheitsstandards. Es erfordert kein Zertifikatmanagement und ist bereits in der sog. `Adaptor.xml` Datei des Streaming Servers aktiviert. RTMPE ist besonders geeignet für Anwendungen, die keine Authentifizierung verlangen und mehr Performance und Geschwindigkeit benötigen als bei einer SSL-Verbindung vonnöten wäre. Allerdings erfordert RTMPE eine 15% höhere Prozessorauslastung als das reine RTMP Protokoll.

[Quelle: http://www.adobe.com/devnet/flashmediaserver/articles/overview_streaming_fms3_02.html, Zugriff: 06.12.2008]

Gegen Ende dieser Arbeit wurde ein weiteres Protokoll und dessen Funktionen vorgestellt, das RTMFP Protokoll. Es sei hiermit auf eine kurze Einführung in die Möglichkeiten von RTMFP verwiesen, die unter folgendem Link abrufbar ist:

http://www.flashcomguru.com/index.cfm/2008/12/18/Future-of-Communication-RTMFP

[Zugriff: 26.12.2008]

2 Streaming Video Codecs & Datei Formate

Zur Kodierung von medialen Inhalten existiert eine Vielzahl unterschiedlicher Codecs. Deshalb soll hier nur eine repräsentative Auswahl an Codecs vorgestellt werden, die häufig in Verbindung mit Streaming Übertragungen eingesetzt werden.

2.1 On2 VP6

Der Hersteller On2 Technologies hat mit TrueMotion VP6 einen Codec geschaffen, der laut eigener Aussage schneller und besser enkodieren soll als die Vorgängerversion und bessere Qualität bietet als konkurrierende Technologien wie bspw. Windows Media 9 and Real Networks 9. Im August 2005 wurde VP6 von Adobe als neues Standard-Videoformat für den Flash Player 8 und 9 sowie das Flash Video-Containerformat FLV ausgewählt. Der On2 VP6-Codec bietet im Vergleich zum Sorenson Spark-Codec eine höhere Videoqualität bei einer Kodierung mit identischer Datenrate. Jedoch erfolgt die Kodierung von On2 VP6-Codec beträchtlich langsamer als beim Sorenson-Spark-Codec. Außerdem erfordert die Dekodierung und Wiedergabe auf dem Client-Rechner mehr Prozessorleistung.

[Quelle: http://livedocs.adobe.com/flash/9.0_de/UsingFlash/help.html?content=WSd60f23110762d6b883b18f10cb1fe1af6-7ca8.html, Zugriff: 11.12.2008]

2.2 Sorenson Spark

Der Sorenson Spark Codec ist ein Video-Codec und zugleich ein Videoformat der Firma Sorenson Media, der erstmals im Jahr 1998 mit QuickTime 3 veröffentlicht wurde. Der Codec wird vorwiegend in älteren Apple Quick-Time- und Adobe Flash Video-Dateien verwendet. Nachdem sich Apple durch die Anwendung des MPEG[G]-4-Standards vom Sorenson-Format abwendete, vergab Sorenson Media die Lizenz für die neueste Version seines Codecs als Sorenson Spark (Sorenson H.263) an die Firma Macromedia. Sorenson H.263 wurde zusammen mit Macromedia Flash 6/MX im März 2002 veröffentlicht. H.263 wurde speziell als Encoder mit einer niedrigen Bitrate für Videokonferenzen entwickelt. Sorenson Spark war neben On2 VP6 der Standard Codec für die Video-Kodierung mit Flash CS3.

[Quelle: http://www.on2.com/index.php?416, Zugriff: 11.12.2008;
Quelle: http://www.nationmaster.com/encyclopedia/Sorenson-Spark, Zugriff: 11.12.2008]

2.3 MPEG-4

Der MPEG-4 Codec wurde von der Moving Picture Experts Group (MPEG) definiert und im Jahr 1998 fertig gestellt. Die Firma Apple hat mit MPEG-4 part 2 (MPEG-4 simple profile) und MPEG-4 part 10 (Codec H.264) zwei eigene ISO-zertifizierte Codecs entwickelt und 2002 publiziert. Apple hat den Codec als offenen Standard für Entwickler freigegeben. Neben den Kernelementen Audio und Video beinhaltet MPEG-4 auch einen Support für 3D Objekte sowie andere mediale Inhalte.
Die Vorteile des MPEG-4 Codec liegen in der Unterstützung einer variablen Bitrate (VBR) und der optionalen Maximierung der Enkodierungsgeschwindigkeit. Ein Anwendungsbereich ist das TV–Broadcasting[G] von Live-Events im MPEG-4 Format. Einige Fernsehanbieter nutzen den MPEG-4 Codec aufgrund einer guten Wiedergabequalität verstärkt für die Bereitstellung von digitalem Fernsehen. Auch im wachsenden Markt hochqualitativer Multimediaan-

wendungen auf drahtlosen Geräten hat der MPEG-4 part 10, auch bekannt als H.264 Codec Einzug gehalten.

[Quelle: http://www.apple.com/quicktime/technologies/mpeg4/, Zugriff. 11.12.2008]

2.4 H.264

Der im Jahre 2003 durch ein Entwicklerkonsortium verabschiedete H.264 Codec ist eine Videokompression, die sich durch eine qualitativ hochwertige Wiedergabequalität und einen guten Kompressionsfaktor auszeichnet. Ziel des Projektes war es, ein Kompressionsverfahren zu schaffen, das im Vergleich zu bisherigen Standards sowohl für mobile Anwendungen als auch für die Videotelefonie und im TV- und HD-Bereich die benötigte Datenrate bei gleicher Qualität mindestens um die Hälfte reduziert. Im Vergleich zu MPEG-2 kann mit H.264 deshalb eine Bitratenreduktion von bis ca. 50 % bei vergleichbarer Qualität erreicht werden. Allerdings ist der Rechenaufwand auch um den Faktor 2 bis 3 höher.
H.264/AVC (Advanced Video Coding) ist eine Weiterentwicklung des MPEG-4 Codec, basierend auf einer neuen netzwerkfähigen Kodiermethode, die mehrere anwendungsspezifische Profile besitzt. Die H.264-Kompression unterstützt konstante (CBR) und variable Bitraten (VBR). Zu den H.264-Profilen gehören das Baseline-Profil, das Main-Profil und das Extended-Profil. Das Baseline-Profil dient der Echtzeitkommunikation z.B. in Videokonferenzen oder Videotelefonaten. Aufgrund der Fehlertoleranz und niedriger Latenzzeiten hat das Baseline-Profil seine Stärken bei dynamischen Netzen geringer Bandbreite. Das auf den Broadcast-Markt zielende Main-Profile benutzt hingegen alle Bildframes[G] der Group of Picture (GoP) und unterstützt das Zeilensprungverfahren. Das Extended-Profil unterstützt das Zeilensprung[G]- und Progressive Scan[G] Verfahren. Es dient der Realisierung problembehafteter Kommunikationskanäle. H.264/AVC eignet sich für die Übertragung breitbandiger Videoströme über das Internet.

[Quelle: http://www.itwissen.info/definition/lexikon/H-264-advanced-video-coding-H-264-AVC-H-264.html, Zugriff am 11.12.2008]

2.5 CBR und VBR Kodierungsverfahren

Ja nach Anwendungsgebiet und Übertragungsmethode sollte entschieden werden, auf welche Weise die Medieninhalte kodiert werden. Grundsätzlich stehen hierbei konstante Bitrate (CBR) oder variable Bitrate (VBR) zur Wahl.
Eine Präsentation von JAN OZER (09/2008) zeigt die Vor- und Nachteile sowie Einsatzgebiete im Zusammenhang mit dem H.264 Codec von CBR und VBR auf.

[Quelle: Pre-Conference: Encoding H.264 Video for Streaming and Progressive Download , http://www.streamingmedia.com/west/presentations/SMWest2008-H264.pdf, Zugriff: 20.12.2008]

Die folgende Grafik vergleicht die beiden Kodierungsverfahren im Hinblick auf die Datenrate sowie die Bildinhalte.

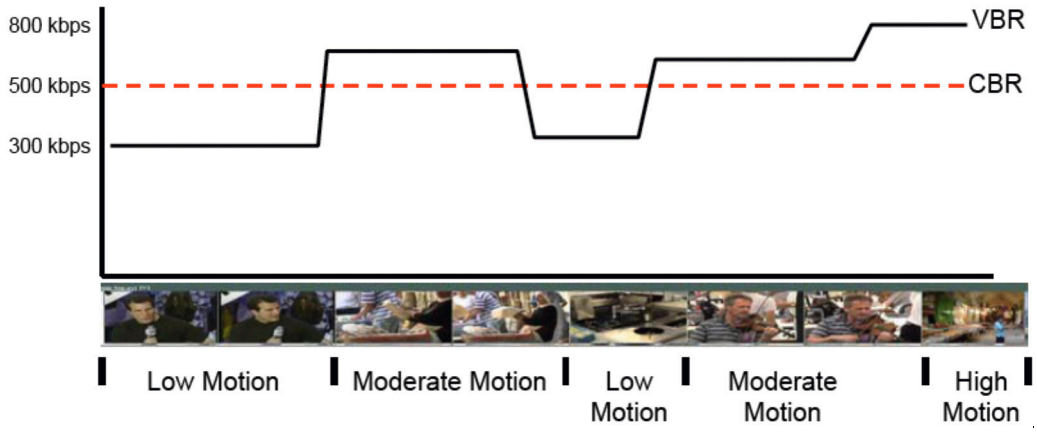


Abbildung II-2: Vergleich von VBR und CBR Kodierungsverfahren abhängig vom Bildinhalt.

[Quelle: http://streamingmedia.com/west/ > SMWest2008-H264-1.pdf, S. 2, Zugriff: 20.12.2008]

Grundsätzlich ist die variable Bitrate (VBR) als effektives Kodierungssystem zu betrachten, da hier die Datenrate abhängig vom Bildinhalt und der dafür erforderlichen Datenmenge dynamisch errechnet wird. Somit steht prinzipiell für jeden Bildinhalt eine angepasste Datenrate zur Verfügung. Auch in der Audio-Kodierung wird dieses Verfahren eingesetzt. Dennoch ist die Eignung der VBR Kodierung abhängig von der Art des Streamings.

Tabelle II-3: Eigenschaften sowie Vor- und Nachteile von CBR und VBR.

CBR	**VBR**
Einheitliche Datenrate für das ganze Video unabhängig vom Bildinhalt	Qualität wird dynamisch angepasst
Vorteile	
Einfach und schnell	Bessere Qualität
Nachteile	
Kein Qualitätsoptimierung	Langsam, kann ungleichmäßiges Streaming verursachen

[Quelle: http://streamingmedia.com/west/ > SMWest2008-H264-1.pdf, S. 3, Zugriff: 20.12.2008[

Dadurch ergeben sich unterschiedliche Einsatzgebiete für CBR und VBR (s. Tabelle II-4).

Tabelle II-4: Einsatzgebiete von CBR und VBR.

Verwendung von CBR	**Verwendung von VBR**
Schnelle Kodierung notwendig (z.B. Live Enkodierung)	Clips mit einer Länge von über 60 Sekunden
Produziert für Streaming	Produziert für Progressive Download
Konstante Bewegungsinhalte im Bild (z.B. Interviews)	Wechselndem Bewegungsinhalt im Clip (schnelle Handlungen, statisches Interview)
	Zeitrahmen für Kodierungsprozess peripher

[Quelle: http://streamingmedia.com/west/ > SMWest2008-H264-1.pdf, S. 4, Zugriff: 20.12.2008]

Der entscheidende Punkt im Hinblick auf die Anwendung eines Kodierungsverfahrens bei einer Applikation ist vor allem die Geschwindigkeit der Kodierung. Wie in der Präsentation von OZER (09/2008) angedeutet, ist ein schneller Kodierungsprozess bei der Live-Übertragung von hoher Bedeutung. Diese Voraussetzung bietet vor allem das CBR-Verfahren. Ist hingegen eine Bereitstellung der Medieninhalte über Progressive Download geplant, eignet sich im Hinblick auf die bessere Qualität eher das VBR-Verfahren.
Bei der Applikationsentwicklung sollte das eingesetzte Kodierungsverfahren also stets unter Berücksichtigung der zukünftigen Übertragungsmethode gewählt werden.

Eine optionale Möglichkeit besteht in der Applizierung eines sog. „mutli-bit rate“ Systems bei der CBR-Kodierung. Der Vorteil läge darin, dass mehrere Qualitätsstufen (verschiedene Datenraten) des Streams respektive einer Datei zur Verfügung stünden. Ferner könnten die Stufen dynamisch an die verfügbare Datenübertragungsrate angepasst werden.

3 Video Encoder & Meta-Daten

Bevor ein Video über den Flash Media Server wiedergeben werden kann, muss es zunächst vom Videorohdaten-Format in ein streamingfähiges Format kodiert werden. Diesen Prozess übernehmen die sog. Encoder. Exemplarisch sollen an dieser Stelle zwei gängige Methoden, die mit unterschiedlichen Systemen arbeiten vorgestellt werden. Es handelt sich um zwei Softwaresysteme mit unterschiedlicher Auslegung. Der FME2.5 eignet sich für ein variables und umfangreich definiertes Streaming aufwendigerer Videoinhalte. Der in Flash CS3 integrierte Encoder ist vor allem für rudimentäre Anwendungen wie Videokonferenzen oder Videochats einsetzbar.
Zur Berechnung der Verhältnisse von Videodatenrate und nötiger Datenübertragungsrate ist unter folgender Adresse ein sinnvolles Hilfsmittel von ROBERT REINHARDT verfügbar:

http://www.adobe.com/devnet/flash/apps/flv_bitrate_calculator/

3.1 Flash Media Encoder 2.5

Der Flash Media Encoder 2.5 (FME2.5) bietet eine grafische Benutzeroberfläche mit einer Vielzahl von Einstellungsmöglichkeiten. Daher sei vorab gesagt, dass er für Live-Streamings mit professionellerem Anspruch wesentlich geeigneter ist als der im nächsten Kapitel erläuterte integrierte Encoder. Ein grundlegender Vorteil liegt bereits auf der Hand: Der FME kann von der FMS und Client-side-Plattform abgekoppelt werden und auf einem solitären Rechnersystem laufen, was einen Gewinn an Leistungsfähigkeit der ganzen Systemkette zur Folge hat.

Nachfolgend soll ein grober Überblick des Funktionsumfangs des FME2.5 präsentiert werden:

In Kombination mit dem Flash Media Server 3.5 ist eine gleichzeitige Enkodierung von bis zu drei Streams mit unterschiedlichen Bitraten möglich (Dynamic Streaming). Weiterhin zeichnet sich der FME2.5 durch folgende Funktionalitäten aus:

- Streaming und gleichzeitiges Aufnehmen von Inhalten (DVR Funktionalität)
- Systemeigener Timecode verfügbar, der auch in Meta-Daten eingebettet werden kann
- Unabhängig voneinander definierbare Datei- und Streamnamen
- Segmentierung der Ausgabedateien möglich
- Begrenzung der Größe und Dauer der Ausgabedatei durch segmentierte Speicherung
- Live Videostreaming über eine Webcam, FireWire, oder ein USB Gerät, das Rohvideodaten produziert und den Microsoft DirectShow Filter unterstützt.
- Videocodec-Unterstützung für On2 VP6 und H.264
- Audiocodec-Unterstützung für Nellymoser und MP3. Mit kommerziellem Plug-In der Fa. MainConcept wird auch AAC und HE-AAC unterstützt.
- Kommandozeilen-Kontrolle
- Auto-Restart nach Fehlfunktion

[Quelle: http://www.adobe.com/products/flashmediaserver/flashmediaencoder/features/, Zugriff: 11.12.2008]

Neben den obligatorischen Einstellungsmöglichkeiten für Video und Audio wie Art des Codecs, Datenrate, Bildgröße, Samplerate etc. können auch die Meta-Daten individuell erweitert werden. Ebenso sind auch die Zielinstanz für den Live-Stream und der Zielordner für die gespeicherte Datei unabhängig voneinander wählbar. Es sei ausdrücklich erwähnt, dass die Entwicklung der Applikation im Rahmen dieses Projektes für die Verwendung des FME2.5 oder anderer Encoder ausgelegt ist und den in Flash CS3 integrierten Encoder nicht berücksichtigt.

3.2 Flash CS3 integrierter Encoder

Flash CS3 bietet über die `Camera`-Klasse und die zugehörige Methode `getCamera()` die Option, eine angeschlossene Kamera, z.B. eine Webcam, in eine Videoinstanz einzubetten oder auf dem FMS über den Befehl `publish` zu veröffentlichen. Diverse Methoden ermöglichen es, die Qualität des Video-Capturings zu verändern, so z.B. `setQuality()` und `setMode()`. Mit erstgenannter Funktion können die verwendete Bandbreite des Videos und dessen Qualität variiert werden. Die `setMode()`-Methode beeinflusst folgende Parameter: `width`, `height`, `fps` und `favorArea`. Es lassen sich dadurch die Breite, Höhe und Bildwiederholrate einstellen. Der `favorArea`-Parameter legt fest, ob sich die Skalierungen und Anpassungen an den von der Kamera vorgegebenen Einstellungen orientieren sollen oder völlig unabhängig davon agiert. Ein über `getCamera()` erzeugtes Video besitzt den `videocodecid`-Wert `2`, wird also standardmäßig mit dem Sorenson Spark Codec kodiert (siehe Tabelle II-6). Eine Option den Codec einzustellen war in der Flash CS3 Dokumentation nicht zu finden, daher ist davon auszugehen, dass zwar diverse Codecs reproduziert werden können, jedoch über den integrierten Encoder der `getCamera()`-Methode lediglich der Sorenson Spark Codec zur Enkodierung verfügbar ist.

3.3 Die Struktur der Meta-Daten

Um Produktionsstandards zu dokumentieren und bei der Vielfalt der Videodaten im Internet bessere Beurteilungsmöglichkeiten zu bieten, werden Videos beschreibende Zusatzinformationen hinzugefügt, die sog. Meta-Daten.
Die folgenden Tabellen geben Aufschluss über die Struktur der Meta-Daten und die Bedeutung der enthaltenen Parameter.

Tabelle II-5: Struktur und Kodierung der Meta-Daten einer FLV-Videodatei.

Parameter	Description
audiocodecid	A number that indicates the audio codec (code/decode technique) that was used.
audiodatarate	A number that indicates the rate at which audio was encoded, in kilobytes per second.
audiodelay	A number that indicates what time in the FLV file "time 0" of the original FLV file exists. The video content needs to be delayed by a small amount to properly synchronize the audio.
canSeekToEnd	A Boolean value that is `true` if the FLV file is encoded with a keyframe on the last frame that allows seeking to the end of a progressive download movie clip. It is `false` if the FLV file is not encoded with a keyframe on the last frame.

cuePoints	An array of objects, one for each cue point embedded in the FLV file. Value is undefined if the FLV file does not contain any cue points. Each object has the following properties:
	`type`
	`name`
	`time`
	`parameters`

duration	A number that specifies the duration of the FLV file, in seconds.
framerate	A number that is the frame rate of the FLV file.
height	A number that is the height of the FLV file, in pixels.
videocodecid	A number that is the codec version that was used to encode the video.
videodatarate	A number that is the video data rate of the FLV file.
width	A number that is the width of the FLV file, in pixels.

[Quelle: Flash CS3 Documentation, Zugriff: 16.12.2008]

Die nachfolgende Tabelle schlüsselt die Bedeutung der `videocodecid`-Werte auf:

Tabelle II-6: Bedeutung der videocodecid-Werte.

videocodecid	Codec name
2	Sorenson H.263
3	Screen video (SWF 7 and later only)
4	VP6 (SWF 8 and later only)
5	VP6 video with alpha channel (SWF 8 and later only)

[Quelle: Flash CS3 Documentation, Zugriff: 16.12.2008]

Die `audiocodecid`-Werte entsprechen folgenden Befehlen:

Tabelle II-7: Bedeutung der audiocodecid-Werte.

audiocodecid	Codec Name
0	uncompressed
1	ADPCM

2	mp3
5	Nellymoser 8kHz mono
6	Nellymoser

[Quelle: Flash CS3 Documentation, Zugriff: 16.12.2008]

Diese Meta-Daten können durch den FME2.5 um eigene Parameter erweitert werden und stellen daher nur einen Auszug der gängigen besonders für diese Arbeit relevanten Meta-Daten dar.

[Quelle: Flash CS3 Documentation, Programming ActionScript 3.0 > Working with video > Using video metadata, Zugriff: 16.12.2008]

III DER ADOBE FLASH MEDIA SERVER 3

1 Ein Überblick der Server Architektur

Der Adobe Flash Media Server setzt sich aus drei Server-Versionen zusammen. Dazu gehören der Flash Media Streaming Server, der Flash Media Interactive Server sowie der Flash Media Development Server. Zur besseren Nachvollziehbarkeit der spezifischen Funktionalitäten folgt eine kurze Erläuterung dieser drei Server-Versionen.

Flash Media Streaming Server
Der Flash Media Streaming Server dient der Übermittlung von Live-Videos und Video on-Demand-Inhalten zum Flash Player. Dadurch lassen sich sowohl gespeicherte als auch Live-Informationen an den jeweiligen Client transferieren. Auf Entwicklerseite existiert eine Client-API, welche die Entwicklung von Anwendungen für den Flash Player, AIR oder Flash Lite ermöglicht. Im Anwendungsbereich des Servers gibt es laut Adobe folgende Einschränkungen: *„This server edition is not intended for high-performance, highly scalable, or custom video solutions.“*
Mit dem Flash Streaming Server können Clients für die Übermittlung von Live-Videos an eine unbegrenzte Zahl von Betrachtern aufgebaut werden. In der Bereitstellung von Videoinhalten gibt es keine zeitliche Beschränkung.

Flash Media Interactive Server
Ebenso wie der Streaming Server erlaubt der Flash Media Interactive Server die Darstellung von Videostreams, besitzt jedoch aufgrund seiner interaktiven Ausrichtung einen stärkeren Anwendungsbezug. Mit dem Interactive Server lassen sich z.B. „social media applications“ wie bspw. Multiyplayer Games für den Flash Player, AIR oder Flash Lite wiedergeben und entwickeln. Der Server bietet ein Software Development Kit (SDK), dass die Entwicklung von Mediaapplikationen in der Flash eigenen Programmiersprache Actionscript auf Server- und Client-Seite ermöglicht. Die Funktionalitäten lassen sich über Plug-Ins erweitern. Der Interactive Server kann als sog. Edge Server[G] Verbindung und Bandbreite steuern.
Der Interactive Server eignet sich durch die Ausbaumöglichkeit mit Actionsscript besonders für den Aufbau von Social Media-Plattformen, auf denen vom Nutzer selbst angefertigte Videos eingestellt werden (z.B. Video Blogging and Messaging). Möglich ist aber auch professionelles Video-Broadcasting.

Flash Media Development Server
Der Flash Media Development Server ist eine limitierte, frei zugängliche Version des Flash Media Interactive Servers. Mit dem Development Server ist die Implementierung einer einfachen Basis-Streamlösung bis zum Aufbau eines komplexen Social Media Networks durchführbar. Unterstützt werden neue Features wie Live Publishing, die Texteingabe bei laufenden Streams und das Ablegen eines Streams in einem Content Delivery Network[G].
Die einzige Limitierung des Flash Media Development Servers im Vergleich zum Interactive Server besteht in nur 10 möglichen simultanen Verbindungen.

Tabelle III-1: Direktvergleich der drei Server Editionen.

Feature	Flash Media Interactive Server	Flash Media Streaming Server	Flash Media Development Server
Simultaneous connections between client and server.	Unlimited	Unlimited	10
Bandwidth	Unlimited	Unlimited	Unlimited
Processor limit	8-way SMP	4-way SMP	8-way SMP
Streaming services (live and vod)	Yes	Yes	Yes
Multiple applications and publishing points	Unlimited	Unlimited (This server edition only runs the vod and live applications)	Unlimited
Archive (record) video on server	Yes	No	Yes
Server-side programming	Yes	No	Yes
Multipoint publishing	Yes	No	Yes
Edge server configuration	Yes	No	Yes
Custom C++ plug-in support	Yes	No	Yes
Core server processes	Unlimited	1	Unlimited
Encrypted streaming (RTMP)	Yes	Yes	Yes
Simple access control (SWF verification)	Yes	Yes	Yes
Adobe Media Player tracking service	Yes	No	No

Client-Server-Architektur

Mit dem Flash Media Server kommunizierende Anwendungen müssen sich erst am Server registrieren. Dazu schafft der Entwickler ein Unterverzeichnis für die jeweilige Anwendung im Hauptverzeichnis der Root-Installation des Flash Media Server 3. Für die jeweiligen Instanzen der Anwendung werden ebenfalls weitere Unterverzeichnisse im Anwendungsordner angelegt (Bspw. in: Flash Media Server 3/applications/exampleApplication/instance1). Der Server Administrator kann das Anwendungsverzeichnis abändern und den Server in mehrere Kodierelemente sowie „Virtual Hosts“ aufteilen. Jeder virtuelle Host kann sein eigenes Anwendungsverzeichnis erhalten.

Data-Modell

Das Datenmodell des Flash Media Servers basiert auf dem shared objects-Prinzip. Sowohl Client- als auch Server-side Actionscript enthalten eine shared objects-Klasse, die dem Entwickler gestattet, Daten über Clients auszutauschen, die mit einem Server verbunden sind. Es existieren lokale shared objects (Client Rechner) und remote shared objects (Server), die von temporärer oder persistenter Natur sein können.

Local shared objects: In diesen sind die Computerdaten eines Nutzers für den Offline-Zugriff sowie die Darstellungseinstellungen gespeichert.

Remote shared objects: Die vom Server gespeicherten remote shared objects können vom Entwickler für Nachrichten, Datensynchronisation und Datenspeicherung genutzt werden. Clients beziehen über die remote shared objects Updates zu Änderungen. Nachrichten können an alle Clients versendet werden, die mit dem jeweiligen remote shared object verbunden sind.

Invoking Remote Methods
Der Flash Media Interactive wie auch der Flash Media Development Server unterstützen eine bidirektionale, asynchrone Aufrufmethode. Clients können Methoden auf dem Server aufrufen und der Server kann Methoden auf den entsprechenden Clients ansteuern.
Im Client-side script muss die `NetConnection.call()`-Methode genutzt werden, um eine Methode des Server-side Client-Objekts aufzurufen. In einem Server-side script muss die `Client.call()`-Methode aufgerufen werden, um eine im `NetConnection`-Objekt des Clients definierte Methode aufzurufen.

Verbindung zu externen Quellen
Die Interactive und Development Server Editionen können mit externen Datenquellen wie Web Services, relationalen Datenbanken oder auch anderen Flash Media Server Applikationen interagieren.

Client-side Actionscript API
Client-side scripts, die in Actionscript 2.0 oder 3.0 geschrieben sind werden über eine API mit den *Server Ressources* verbunden.
Folgende Actionscript-Klassen können vom Entwickler für einen Flash Media Server Zugriff genutzt werden:

- `NetConnection` ist eine bidirektionale Verbindung zwischen Client und Flash Media Server oder zwischen Flash Media Server und einem weiteren Flash Media Server
- `NetStream` ist ein unidirektionaler Stream zwischen Client und Flash Media Server durch ein `NetConnection`-Objekt.
- Beide Klassen `NetConnection` und `NetStream` erlauben die Übermittlung von Video- oder Audiodaten und Textdaten an einen Flash Media Server
- Über die `SharedObject`-Klasse können Daten zwischen verschiedenen SWF Dateien oder Flash Media Server Instanzen ausgetauscht werden.

Server-side Actionscript API
Im Server-side Actionscript können über den Interactive und Development Server folgende Funktionen implementiert werden:
- Kontrolle über Log-In Prozeduren
- Übermittlung von Inhalten an andere Server
- Regelung der Nutzungs- und Zugriffsrechte auf Serverressourcen
- Update und Transfer von Informationen zwischen den Endnutzern

Server-side Actionscript ist die Bezeichnung von Adobe für JavaScript 1.5, welches Actionscript 1.0 ähnelt.
Eine Liste aller verfügbaren Server-side Actionscript-Klassen findet sich in der *Client-Side ActionScript Language Reference for Flash Media Server 2* unter:
http://livedocs.adobe.com/fms/2/docs/00000630.html [Zugriff: 24.12.2008]
Die Client-side Actionscript-Klassen sind in der *Server-Side ActionScript Language Reference* unter folgendem Link aufgelistet: http://livedocs.adobe.com/fms/2/docs/00000523.html
[Zugriff: 24.12.2008]

Plug-in API
Flash Media Interactive und Development Server bieten die Möglichkeit einer Funktionserweiterung durch Plug-Ins. So lassen sich z.B. Sicherheits- oder Zugangsbeschränkungen bei einem Serverzugriff über Access oder Authorization Plug-Ins einschalten.

Administration des Servers
Jede Edition des Flash Servers wird mit einem zusätzlichen Server, dem Flash Media Administration Server installiert. Dieser Server besitzt eine Administration-API zur Konfiguration des Flash Media Servers und kann zur Entwicklung eigener Server Administration Tools genutzt werden.

Administration Console
Die Administration Console bietet ein Interface[G], um Administratoren zu managen und den Server sowie darauf laufende Anwendungen detailliert zu überwachen. Dabei existieren zwei Formen des Administration Console Users: Server Administratoren und Virtual Host Administratoren.

Administration-API
Die Administration-API kann eingesetzt werden, um den Server über einen Flash Player oder AIR Client mittels RTMP zu managen und zu konfigurieren. Möglich ist dies auch über einen Web Client via HTTP. Folgende Funktionen werden von der API unterstützt:

- Überwachung des Servers und seiner Anwendungen und Services
- Durchführung administrativer Aufgaben wie das Hinzufügen von Nutzern mit Administrationsrechten oder Starten sowie Anhalten des Servers, von Virtual Hosts oder Applikationen
- Betrachten und Festlegen von Werten zur Serverkonfiguration

Einige Methoden sind nur für Serveradministratoren verfügbar, Virtual Host Administratoren können diese Funktionen nicht nutzen.

Die Liste der verfügbaren Methoden der Administration-API findet sich unter folgendem Link: http://livedocs.adobe.com/fms/2/docs/00000208.html.

[Quelle: http://livedocs.adobe.com/flashmediaserver/3.0/docs/help.html?content=Book_Part_28_tech_overview_1.html, Zugriff 28.12.2008]

2 Anwendungsgebiete

Der Flash Media Server 3 bietet potentielle Anwendungsfelder verschiedenster Art. Aktuelle Statistiken belegen, dass inzwischen mehr als 50% der Internetbenutzer über ein installiertes Flash-Player-Plugin in ihrem Internetbrowser verfügen, so dass die Wiedergabe der Inhalte des Flash Media Server bzw. der zugehörigen Applikationen für viele Anwender gewährleistet ist.

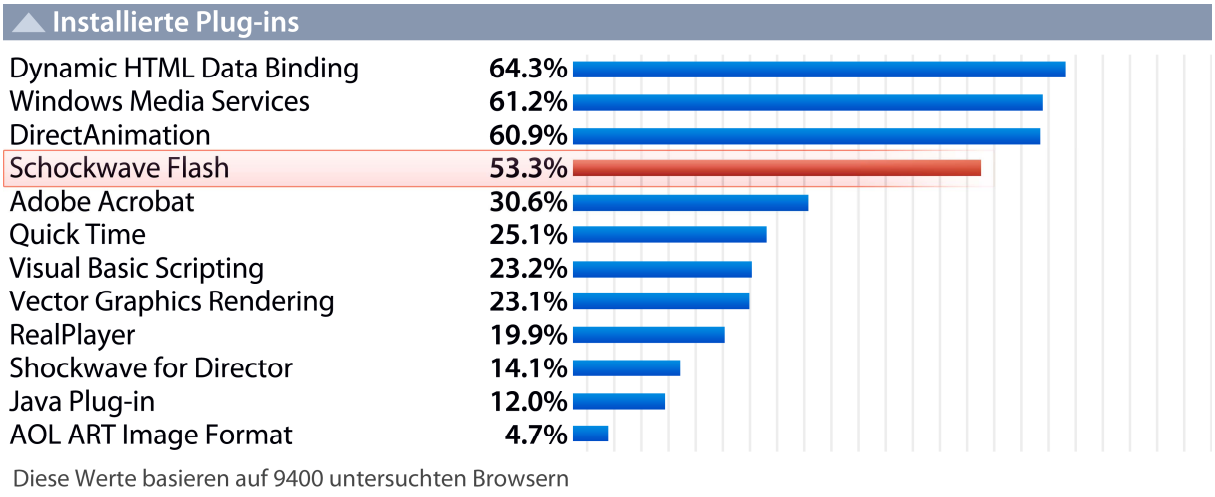


Abbildung III-1: Statistik der installierten Internet-Browser Plug-Ins.

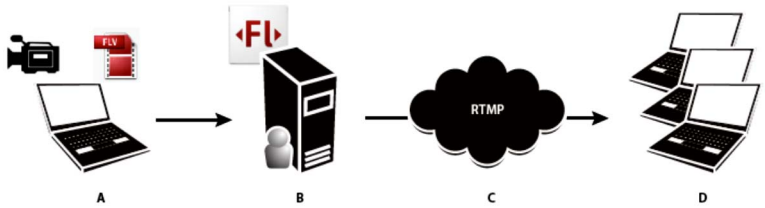
[Quelle: http://www.webhits.de/deutsch/index.shtml?webstats.html, Zugriff: 18.12.2008]

Im Folgenden werden exemplarisch einige verbreitete Anwendungsfelder der Flash Media Server Familie erläutert:

Video-Player

Es gibt die Möglichkeit einen Video-Player mittels der integrierten FLV Playback Komponente oder als individuelle Actionscriptvariante umzusetzen (s. Abbildung III-2). Die hier erwähnten Beispiele zeigen potentielle Streaming-Formate und -Anwendungen:

- Kurze Videoclips, wie z.B. Werbespots mit bis zu 30 Sekunden Länge
- Längere Clips (auch “long tail” genannt), wie z.B. Benutzervideos mit 30 Minuten Länge
- Sehr lange Clips, wie gespeicherte Fernsehsendungen mit mehreren Stunden Länge
- Über Wiedergabelisten können die Streams in Abfolge gezeigt werden, auch alternierende Typen wie live und gespeicherte Streams lassen sich über eine Liste wiedergegeben. Die Wiedergabeliste kann entweder statisch auf der Client-Seite definiert sein oder über den Flash Media Interactive Server in einem Server-side-Skript dynamisch erzeugt werden.



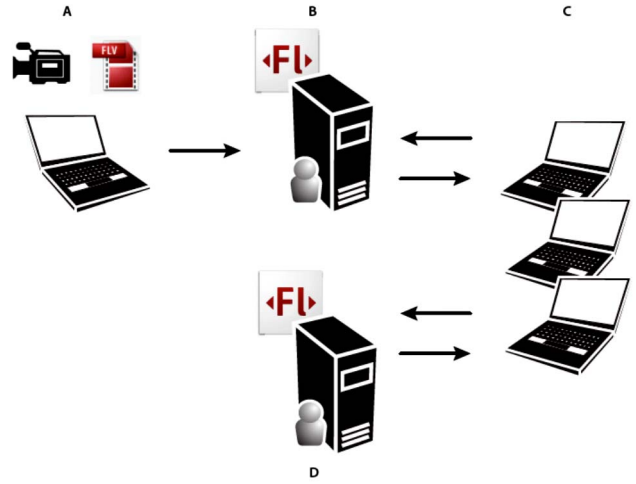
***A.** Flash Media Encoder (or custom-built Flash Player or AIR solutions) capture live audio and video and publish it to the server.*
***B.**FlashMedia Server streams live and recorded media to clients. **C.** Internet (RTMP) **D.** Flash Player, AIR, or Flash Lite run the video players*

Abbildung III-2: Anwendungsgebiete des FMS. Video-Player.

[Quelle: Flash Media Server Documentation – Technical Overview, Zugriff: 18.12.2008]

Video mit Werbung

Eine Streaming-Applikation kann an beliebigen Stellen auf kommerzielle Spots zurückgreifen und somit vor oder nach einem Event Werbung oder Logos einspielen (s. Abbildung III-3). Hierbei werden meist zwei diskrete Server verwendet, einer für die Wiedergabe der Inhalte und ein zweiter für die Einstreuung der Werbung bzw. anderer Clips. Dabei verbindet sich die Applikation in zyklischem Rhythmus zur Wiedergabe der Werbeinhalte mit dem entsprechenden Server und beendet abschließend diese Konnektivität, um die Verbindung mit dem inhaltsbasierten Video-Server aufzubauen.



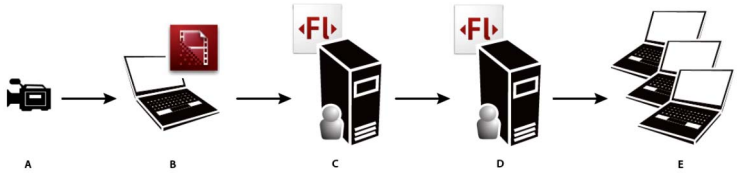
***A.** Live video **B.** Flash Media Server (serving recorded and live content) **C.** Flash Player, AIR, or Flash Lite clients **D.** Flash Media Server (serving ads)*

Abbildung III-3: Anwendungsgebiete des FMS. Video mit Werbung.

[Quelle: Flash Media Server Documentation – Technical Overview, Zugriff: 18.12.2008]

Live-Video mit „multipoint publishing“

Wenn zu Broadcasting-Zwecken Mediendateien an eine hohe Anzahl von Konsumenten übertragen werden, kann hierzu unterstützend das „multipoint publishing“, also die Arbeitsteilung durch mehrere Server, genutzt werden (s. Abbildung III-4). In diesem Fall wird das Live-Video zunächst an den „publishing server“ übertragen, dies könnte z.B. ein Flash Media Development Server sein, und dann an den „broadcast server“ - bspw. Flash Media Interactive Server - weitergeleitet, der den Stream an die Benutzer verteilt.



***A.** Live video **B.** Flash Media Encorder (or custom-build Flash Player or AIR solutions) **C.** Flash Media Server (publishing) **D.**FlashMediaServer (broadcasting) **E.** Flash Player, AIR, or Flash Lite clients*

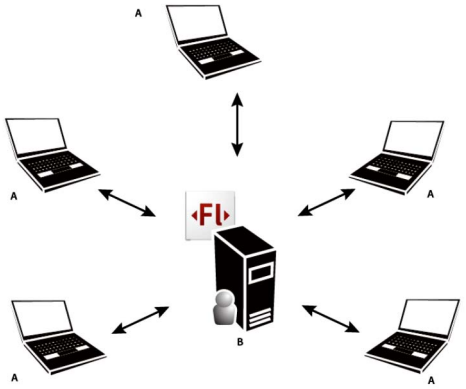
Abbildung III-4: Anwendungsgebiete des FMS. Live-Video mit „multipoint publishing“.

[Quelle: Flash Media Server Documentation – Technical Overview, Zugriff: 18.12.2008]

„Multipoint publishing“ kann dazu eingesetzt werden, um Applikationen mit aufwendigen Videoinhalten zu realisieren oder den Streams zusätzliche Meta-Daten zu addieren.

„Social media“ Anwendungen

Eine Applikation mit dem Flash Media Interactive Server kann dem Benutzer Video-Sharing, Online-Chat, Internet-Konferenzen und andere gruppenbasierte Funktionen offerieren (s. Abbildung III-5). Dabei ist eine Kommunikation mittels Audio, Video oder Textnachrichten möglich, die vom FMS an die anderen Benutzer vermittelt wird. Optional können auch Aufzeichnungen gestartet werden, die einen späteren Zugriff auf die Inhalte gewährleisten, so z.B. für Video-Nachrichten.



***A.**Clients can send and receive audio and video and text messages. **B.**Flash Media Interactive Server broadcasts the media and data to all connected users.*

Abbildung III-5: Anwendungsgebiete des FMS. Social Media Applications.

[Quelle: Flash Media Server Documentation – Technical Overview, Zugriff: 18.12.2008]

3 Die Systemvoraussetzungen

Die Systemvoraussetzungen für die Installation und Verwendung des Flash Media Interactive Server 3.0 wurden von Adobe wie folgt festgelegt:

- Microsoft® Windows Server® 2003 SP1 (All 32-bit editions)
- Linux® Red Hat® 4 (32-bit only)
- 3.2GHz Intel® Pentium® 4 processor (dual Intel Xeon® or faster recommended)
- RAM: 2gB minimum, 4gB recommended
- 1Gb Ethernet card

In der Praxis zeigt sich jedoch, dass der Flash Media Server Development Edition auch unter Microsoft Windows XP Professional mit installiertem Apache Webserver lauffähig ist.

[Quelle: http://www.adobe.com/de/products/flashmediainteractive/systemreqs/, Zugriff: 28.12.2008]

4 Modifikationsoptionen und Umgebungsvariablen

Der Flash Media Server 3 bietet die Option eine individuelle Verzeichnisstruktur aufzubauen und durch sog. virtuelle Verzeichnis relativ auf eigene Systemordner außerhalb der FMS3 Installation zuzugreifen. Zudem ermöglichen einzeln zuweisbare Freigabeberechte z.B. den Zugriff auf Funktionen einer Kontrollinstanz des FMS3 aus der Flash Applikation heraus. Die Server-Struktur sieht zudem nicht nur eine Kommunikation zwischen Client und Server sondern auch mehrerer Server untereinander vor. Der FMS3 kann also auf die jeweiligen Bedürfnisse recht gut adaptiert werden und bietet durch größtenteils selbsterklärende XML-Dateien vielfältige Einstellungsoptionen. Durch externe Plug-Ins können die Funktionen zusätzlich erweitert werden.

IV SYSTEMATIK DES INTERAKTIVEN DESIGNS

Zur Gestaltung interaktiver Objekte oder Programme, nachfolgend *Produkt* genannt, stehen einige Methoden zur Wahl, die eine Basis zur benutzeradaptierten Entwicklung bieten. Die Methoden und deren Anwendung werden unter dem Begriff Usability zusammengefasst. Die Definition von Usability ist nach DIN in der ISO-Norm 9241-11 unter der Übersetzung *Gebrauchstauglichkeit* wie folgt festgelegt:

> *„das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufrieden stellend zu erreichen.“* [DIN EN ISO 9241-11, 1998]

Daraus geht hervor, dass die Usability eines Produktes stets in einem zugehörigen Nutzer- und nutzungsbezogenen Kontext zu betrachten ist. Dies impliziert eine notwendige Anpassung des Produktes an die Ansprüche und Fähigkeiten des Benutzers. Dabei sollten folgende Punkte nach RICHTER UND FLÜCKIGER (2007, S. 5) beachtet werden:

> *„- Analyse der Benutzer, ihrer Aufgaben und des Anwendungskontexts*
> *- Definition des Funktionsumfangs und der benötigten Informationen*
> *- Erarbeitung der optimalen Abläufe und Prozesse“*

Folgende Usability Methoden stehen hierfür zur Verfügung:

> *„- Contextual Inquiry*
> *- Personas und Szenarien*
> *- Storyboard*
> *- UI Prototyping*
> *- Use Cases*
> *- Usability Guidelines und Styleguides*
> *- Usability Testing*
> *- Fragebögen“*

[VGL. RICHTER UND FLÜCKIGER, 2007, S. VII]

Diese Methoden greifen größtenteils ineinander über und können bzw. sollten in bestimmter Abfolge im Software-Entwicklungsprozess berücksichtigt werden. Da die Methode der Personas und Szenarien sowie des UI Prototyping in der praktischen Umsetzung dieser Arbeit angewandt wurde, wird in Kapitel IV Abschnitt 1 und Abschnitt 2 eine detaillierte Erläuterung dieser Systematiken vorgenommen. Nachfolgend werden die weiteren Methoden zur Vollständigkeit kurz zusammengefasst.

Contextual Inquiry beschreibt eine Befragung von potentiellen Benutzergruppen im Vorfeld. Dabei werden vor der Produktentwicklung vom Analysten Fragebögen erarbeitet, die Aufschluss über folgende Attribute des Befragten geben sollen:
Das soziale Umfeld des Benutzers, den kulturelle Einfluss, typische Rollenverteilung (Mann – Frau), Kommunikation (bspw. Kommunkationsmittel, Verantwortlichkeiten), Handlungsstra-

tegien und unterschiedliche Vorgehensweisen sowie Artefakte (z.B. benutzte Hilfsmittel zur Arbeitsverrichtung).
Weitere Themen ergeben sich aus dem Verlauf der Befragung. Die analysierten Ergebnisse fließen anschließend in die Erstellung von Personas und deren Ziele sowie geeigneter Szenarien ein.

Es sei angemerkt, dass der Schritt der Contextual Inquiry aus zeitlichen Gründen für diese Arbeit nicht umgesetzt werden konnte. Daher entstanden die in Kapitel V Abschnitt 1.1 verwendeten Persona aus persönlicher Erfahrung sowie in Anlehnung an das private und berufliche Umfeld des Autors.

Ein *Storyboard* visualisiert skizzenhaft u.a. bestimmte Abläufe und Zusammenhänge des künftigen Programms aus Benutzersicht. Diese Skizzen können mit den Auftraggebern verglichen werden und präzisieren sich mit Voranschreiten des Projektes. Sie dienen vor allem der unterstützenden Darstellung komplexer Funktionsabläufe.

Sog. *Use Cases-Modelle* werden in der Softwareentwicklung eingesetzt und dienen dazu, konkrete Anwendungsfälle durchzuspielen, die bei Interaktion des Anwenders mit der Software auftreten. Der Use Case ist hierbei der auszuführen Befehl bzw. Vorgang. Analysiert wird darauffolgend die Reaktion des Programms auf diese Anfrage. Die Resultate bieten eine erste *„Übersicht über die Funktionalität eines Systems“*. [RICHTER UND FLÜCKIGER, 2007, S. 48]

Die *Usability Guidelines und Styleguides* sind eine Zusammenfassung von Normen und Verordnungen bspw. der DIN, die Gestaltungsvorgaben und ergonomische Anforderungen von interaktiven Softwareelementen wie z.B. Dialogfeldern beinhalten.

Das *Usability Testing* wird in einem fortgeschrittenen Entwicklungsstadium des Produktes, z.B. nach dem ersten Prototypen, angewandt. Es handelt sich um eine von Testpersonen durchgeführte Testreihe des Produktus unter Beobachtung der Testleiter. Dabei wird zwischen *„formativer Evaluation, welche eine Verbesserung des geprüften Systems zum Ziel hat, und summativer Evaluation, welche ein Produkt im Sinne einer Qualitätskontrolle zusammenfassend prüft“* [VGL. RICHTER UND FLÜCKIGER, 2007, S. 55] unterschieden. Die Testpersonen prüfen die Benutzerschnittstelle an Hand standardisierter Testaufgaben und stehen unter ständiger Beobachtung der Testleiter, die nur in Ausnahmefällen eingreifen sollen. Das Ergebnis identifiziert die Schwachstellen und zeigt Verbesserungsmaßnahmen auf.

Die Fragebögen können an mehreren Zeitpunkten im Entwicklungsprozess eingesetzt werden. So zum Beispiel im *„Anschluss an Contextual Inquiry Evaluationsinstrument: Im Rahmen eines Pilottests oder zur ständigen Qualitätssicherung“* [VGL. RICHTER UND FLÜCKIGER, 2007, S. 68]. Die Ergebnisse der Befragungen dienen der statistischen Auswertung der Usability oder zur Qualitätssicherung des Systems.

[VGL. RICHTER UND FLÜCKIGER, 2007, S. 19FF]

1 Personas und Ziele

Personas

Personas sind Modelle, die auf Beobachtungen reeller Personen basieren. Sie sind also als Personifikationen zu verstehen. Personas verstärken den empathischen Faktor im Design und Entwicklungsprozess. Sie sollen den Entwicklern und Geschäftsinteressenten die Vorstellung einer reellen Person ermöglichen. [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 81]

Nichts desto trotz sollte verinnerlicht werden, dass eine spezifische Persona immer eine Gruppe von Personen repräsentiert und ein Modell darstellt. Es darf jedoch nicht mit einer stereotypischen Auffassung verwechselt werden, denn diese entbehrt meist eines faktischen und empirischen Fundaments und wirkt somit der sinnvollen Entwicklung einer Persona entgegen. Es soll kein Durchschnittsbenutzer entstehen, sondern konkrete Verhaltensweisen aus einer Benutzergruppe spezifiziert werden. Dies führt in vielen Fällen zu Erzeugung mehrerer Personas, einem Persona Set. Den einzelnen Persona werden dabei unterschiedliche Verhaltensmuster zu Grunde gelegt, die u. U. zu anderen Programmabläufen führen können.

Den Personas ähnliche Modellierungstypen sind*: user roles, user profiles und market segments*.
„A user role or role model, as defined by Larry Constantine, is an abstraction, a defined relationship between a class of users and their problems (...), they are not imagined as people, and do not typacally attempt to convey broader human motiviations and contexts.“ [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 84].
Der Abstraktionsgrad ist hier also höher, es handelt sich nicht um konkret definierte Benutzermodelle.
Personas und *user profiles* werden oft synonym genannt. *„There is no problem with this if the profile is truly generated from ethnographic data and encapsulates the depth of information the authors have described.“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 85]. COOPER führt die Problematik des o.g. Vergleichs vor dem Hintergrund an, dass es sich bei user profiles häufig um demographisch basierte Beschreibungen handelt, die Gefahr laufen in Klischeesysteme abzudriften und dabei spezielle Anforderungen zu oft außer Acht geraten.

Market segments: *„the main difference between market segments and design personas is hat forme rare based on demographics, distribution channels, and purchasing behavior, whereas the latter are based on usage behavior and motivations.“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 86]. Man kann also von einer stärker kommerziellen Ausrichtung der *market segments* Methode sprechen, wohingegen die Personas eher auf die Funktionalität und Benutzbarkeit des Produktes zielen und die Ansprüche der künftigen Benutzer berücksichtigen, weniger die der Käufergruppe und der Vermarktung.

Nahezu jeder Benutzer hat unterschiedliche Vorstellungen, dieser Sachverhalt liegt auf der Hand, die Personas bilden daher stets einen Kompromiss, jedoch lässt sich dieser Kompromiss eingrenzen, indem man mehrere Personas mit unterschiedlicher Gewichtung definiert.
„To create a product that must satisfy a diverse audience of users, logic might tell yout to make it as broad in ist functionality as possbile to accommodate the most people. This logic, however, is flawed. The best way to successfully acommodate a variety of users is to design

for specific types of individuals with specific needs. (...) then prioritize these individuals so that the needs of the most important users are met without compromising our ability to meet the needs of secondary users.“ [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 77]

In diesem Zusammenhang wird zwischen *primary*, *secondary*, *supplemental*, *customer*, *served* und *negative* Personas differenziert. Die *Primary* Personas repräsentieren die primären Ziele für die Entwicklung der Applikation, sie stehen im Vordergrund und sind für jedes Produkt singulär. Sollten spezifische Anforderungen von dieser Persona nicht abgedeckt werden, können *secondary* Personas hinzugezogen werden (nicht mehr als 2-3), die mit den *primary* Personas weitestgehend kongruent sind, jedoch zusätzliche Apsekte berücksichtigen.
Die Eigenschaften der *supplemental* Personas resultieren aus der Schnittmenge von primary und secondary Personas. *„Often political personas – the ones added to the cast to address stakeholder assumptions – become supplemental personas“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 105]. *Customer* Personas definieren die Ansprüche des Kunden resp. Auftraggebers, nicht die der End-Benutzer. Sie können bspw. für die Entwicklung eines Administrativen Interfaces angewandt werden. *Served* Personas stehen nur in inderektem Bezug zur Benutzerschnittstelle des Produktes. Die Zielgruppe arbeitet nicht direkt mit dem Produkt, profitieret jedoch inderekt von einer guten Usability der Schnittstelle (bspw. der Patient in einer technikgestützten medizinischen Untersuchung). *Served* Personas besitzen eine vergleichbare Gewichtung wie secondary personas. Die letzte Klasse, die *negative* Personas, unterscheidet sich deutlich von den zuvor genannten, denn sie spezifizieren die Benutzergruppe, die von dem Produkt keinesfalls berücksichtigt werden soll. *„Good candidates for negative personas are often technology-savvy early adopter personas for consumer products and IT specialists for business-user enterprise products.“*

[VGL. COOPER, REIMANN UND CRONIN, 2007, S. 105F]

Folgende Probleme können durch Personas gelöst werden:

The elastic user: Jeder Entwickler, der am Produktionsprozess beteiligt ist, hat andere Vorstellungen des End-Benutzers. Das daraus resultierende Problem: Sollen Entscheidungen im Entwicklungsprozess getroffen werden, gilt es unterschiedliche Meinungen der Entwickler zu kombinieren, was ggf. zu einer Kompromisslösung führt und den gesamten Prozess verlangsamt.

Self-referential design, *„self-referential design occurs when designers or developers project their own motivitaions, skills, and mental models onto a product’s design.“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 80]. Dieses Phänomen beschreibt die selbstbezogene Gestaltung eines Produktes aus Sicht des Entwicklers. Dieser reflektiert zu stark die eigenen Vorstellungen und berücksichtigt nachrangig die Eigenschaften des End-Benutzers.

Edge cases, „those situations that might possibly happen, but usually won’t fort he target personas.“ Sollten zwar berücksichtigt werden aber nicht im Fokus stehen.

[VGL. COOPER, REIMANN UND CRONIN, 2007, S. 79F]

Personas basieren auf Beobachtungen der Umwelt. Wie im Vorfeld erwähnt ist die Contextual Inquiry-Methode für die Entwicklung von Personas eine nützliche Basis.

Ziele
Die Benutzerziele definieren sich meist aus observierten Verhaltensmustern und aus Rückschlüssen des Personenumfeldes. Eine direkte Bestimmung der Ziele durch konkrete Befragungen führt meist zu keinem brauchbaren Ergebnis, da die Befragten diesen Fragentyp gar nicht oder nur unzureichend beantworten können.

Nach COOPER, REIMANN UND CRONIN lassen sich drei verschieden Arten von Benutzerzielen spezifizieren:

- Experience goals
- End goals
- Life goals

„Expereience goals express how someone wants to feel while using a product or the quality of their interaction with the product. These goals provide focus for a product's visual and aural chracteristics, its interactive feel – such as animated transitions, latency, and the snap ratio (clickiness) of a physical button" [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 92]
Die *experience goals* beschreiben also die aufkommenden Gefühle eines Benutzers bei der Interaktion mit dem Produkt.

„End goals represent the user's motivitaions for perfomring the tasks associated with using a specific product." [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 93]. Sie spiegeln die eigentlichen Ziele des Benutzers wieder, die er mit dem Produkt erreichen will. Ihnen muss somit eine hohe Bedeutung beigemessen werden.

„Life goals represent personal aspirations of the user that typically go beyond the context of the product being designed. Thesse goals represent deep drives and motivations that help explain why the user is trying to accomplish the end goals he seeks to accomplish." [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 93]. *Life goals* stellen also die emotionale entscheidungsbasis für die End goals dar.

Weiteres Unterscheidungsmerkmal der Ziele ist deren Typisierung. Im Vorfeld wurden die Benutzerziele, *user goals*, behandelt. Des Weiteren lassen sich *Customer goals*, *business and organizational goals* und *technical goals* spezifizieren, die an dieser Stelle mit Ausnahme der *technical goals* jedoch nicht näher erläutert werden sollen. Die *technical goals* repräsentieren die Ziele die für den Prozess und Hintergrund der Entwicklung von Bedeutung sind. Sie dienen also eher den Entwicklern und der Effektivität des Produktes und nur indirekt den Benutzern selbst.

Die Benutzerziele sind vorrangig unter dem Verständnis der Motivation und Zielsetzung und weniger als Spezifizierung spezieller Aufgaben und demographischer Merkmale zu verstehen.
[VGL. COOPER, REIMANN UND CRONIN, 2007, S. 94]

„The most important purposes or goals to consider when designing a product are those of the individuals who actually use it, not neccessarily those of its purchaser." [VGL. COOPER, REIMANN

UND CRONIN, 2007, S. 96]. Die Benutzerziele sollten demnach stets prioritär behandelt werden und sekundär die Ziele der Vermarkter und Kunden.

Die Konklusion von COOPER, REIMANN UND CRONIN bezüglich der erfolgreichen Entwicklung von Produkten und deren Benutzerschnittstellen spiegelt sich in folgender Aussage wieder: *„The essence of good interaction design is devising interactions that achieve the goals of the manufacterer or service provider and their partners without violating the goals of users.“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 97]

2 Gestaltung von grafischen Benutzeroberflächen

Bei der Gestaltung grafischer Benutzeroberflächen spielt der folgende Begriff eine wesentliche Rolle: *UI Prototyping*. Es *„wird im Usability Engineering eingesetzt, um Aspekte der Benutzerschnittstelle zu entwerfen, zu evaluieren und zu verbessern, noch bevor ein lauffähiges System vorhanden ist.“* [VGL. RICHTER UND FLÜCKIGER, 2007, S. 36].

Sukzessive mit dem Entwicklungsprozess des Produktes steigt auch die Darstellungstreue der Skizzen. Begonnen wird meist rudimentär mit Bleistift und Papier. So lassen sich die Vorstellungen schnell verbildlichen und kommunizieren, der Zeitaufwand bleibt gering.
Im Vorfeld des UI Prototyping gilt es, zunächst den Umfang des Prototypen festzulegen. RICHTER UND FLÜCKIGER legen hierzu folgende Aspekte vor:

> *„Funktionsumfang: Wie viel der vorgesehenen Funktionalität des Benutzerschnittstelle soll im Prototyp gezeigt werden? Sind dies ausgewählte Ausschnitte, oder geht es darum, den gesamten Umfang darzustellen?*
> *Funktionstiefe: Wie detailliert sollen die einzelnen funktionalen Elemente wiedergegeben werden? (...)*
> *Darstellungstreue: Wie ähnlich soll der Prototyp dem Endprodukt in Bezug auf Aussehen der Benutzeroberfläche (Look&Feel) sein?*
> *Interaktivität: (...) Braucht es lauffähige Beispiele, um komplexe Abläufe wiederzugeben, oder genügen statische Darstellungen der Benutzerschnittstelle?*
> *Datengehalt: Sollen reale Daten zum Einsatz kommen, genügen realistische Beispiele oder gar Platzhalter für Bezeichnungen und dargestellte Inforamtionen? (...)*
> *Technische Reife: (...) Muss der Prototyp mit der Entwicklungsumgebung der Zielplattform entwickelt werden, oder sind einfache Zeichnugnswerkzeuge ausreichend?“*
>
> [VGL. RICHTER UND FLÜCKIGER, 2007, S. 36]

Repräsentativ soll die Entwicklung eines UI Prototypen für die Applikation dieser Arbeit am Beispiel des Front-Ends^{G} dargestellt werden (siehe Abbildung IV-1).

Der *Funktionsumfang* soll zunächst nicht in den einzelnen Abläufen berücksichtigt werden, sondern nur aus der Übersicht der gesamten Oberfläche hervorgehen. Dies ist mit der bewussten Reduzierung von Dialogboxen im Front-End zu begründen.
Da die Funktionselemente größtenteils als *immediate vectors* umgesetzt sind, also unmittelbar nach einer Benutzeraktion ein Ereignis auslösen, können die meisten Funktionen auf einer einzelnen Übersicht dargestellt und somit eine relativ weitreichende *Funktionstiefe* erreicht werden. Zudem sollte ein möglichst hoher Grad der *Darstellungstreue* erzielt werden. So soll sich das Aussehen des Prototypen der finalen Oberfläche stark annähern. Auch die Größenverhältnisse der Elemente sollten sofern möglich Berücksichtigung finden.

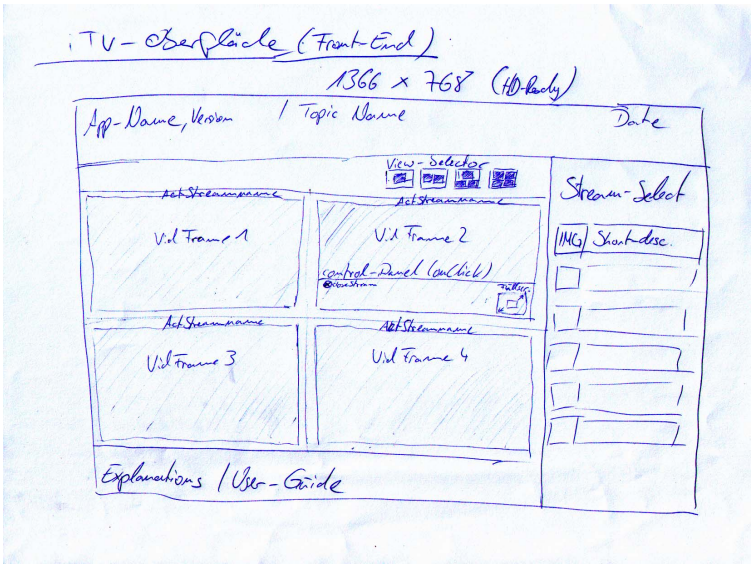


Abbildung IV-1: Skizze des Front-End User Interfaces.
[Quelle: eigener Entwurf, größere Abbildung siehe Abbildung A-9 im Anhang]

Da die Daten für den End-Benutzer grafisch aufbereitet und so übersichtlich wie möglich gehalten werden sollen, genügen Platzhalter zur Verbildlichung der Funktionselemente, der *Datengehalt* ist somit gering. Die Papierzeichnung impliziert eine geringe *technische Reife*. Obgleich zwischen diesem Prototypen und der finalen Oberfläche keine weitere Visualisierungsinstanz vorhanden ist, fielen die Transferschritte vom Prototyp zur Entwicklungsumgebung realtiv klein aus, so konnte das grafische Aussehen auf der Zielplattform (Flash) weitestgehend dem Papierprototypen angenähert werden (s. Abbildung A-9 u. Abbildung A-10).

Diese Art von Papierprototypen, also Attrappen der Benutzerschnittstellen, wird auch als *Mock-ups* bezeichnet. *„Mit Mock-ups können Benutzer bereits konkrete Fälle durchspielen und diskutieren.“* [VGL. RICHTER UND FLÜCKIGER, 2007, S. 37].
„Ein weitere wichtiger Punkt ist dass Mock-ups auf Papier und elektronische Prototypen nicht die gleiche Wirkung haben. Papierprototypen signalisieren durch ihre Skizzenhaftigkeit, dass noch viel offen ist und auch über Grundsätzliches diskutiert werden kann. Entsprechend lässt sich gezielter über Abläufe und den konzeptionellen Aufbau diskutieren, Bei einem endgültig aussehenden Prototypen gehen Personen eher davon aus, dass das Grobkonzept bereits feststeht und nur noch an den Details gefeilt werden soll.“ [VGL. RICHTER UND FLÜCKIGER, 2007, S. 41]

Um die Ästhetik des Designs zu prüfen ist ein Papieprototyp nur teilweise geeignet, da das grafische Aussehen auf der Zielplattform nur bedingt wiedergegeben werden kann. Besonders im Hinblick auf Farben und Kontraste, Verwendung von Schriften und Stil und Darstellung von Icons bieten Grafikprogramme hier bessere Funktionen Design-Varianten zu entwickeln und miteinander zu vergleichen.

Da die Entwiclungsumgebung dieser Applikation, Flash CS3, jedoch auch erweiterte grafische Funktionen bietet, war in diesem Fall keine gesonderte Nutzung von Programmen wie Adobe Photoshop nötig. Der Transfer vom Papier Mock-up zu Flash und damit der finalen Optik der Benutzerschnittstelle war barrierefrei und bot genügend Variationsmöglichkeiten in der Gestaltung.

3 Interaktionsfunktionalität

Es existieren eine Vielzahl unterschiedlicher Funktionselemente die dem Benutzer eine Interaktion mit der grafischen Oberfläche und somit dem Programm ermöglichen. *Menus*, *Toolbars*, *Dialogboxes*, *Buttons* sind nur einige dieser Elemente, die in unterschiedlichster Gestaltungsart vorkommen. Deren Anordnung zueinander, die Beschriftung/Symbolik und die Funktionalität sind für die Effektivität der Benutzeroberfläche von Bedeutung.
Die Kontrollelemente werden nach COOPER, REIMANN UND CRONIN als *command vectors* bezeichnet und in diesem Zusammenhang zwischen *immediate* und *pedagogic vectors* differenziert.

„Direct manipulation controls, like pushbuttons and toolbar controls, are immediate vectors. There is no delay between clicking a button and seeing the results of the function. (...) Neither menus nor dialog boxes have this immediate property. Each one requires an intermediate step, sometimes more than one.“ [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 552]. Immediate vectors repräsentieren somit Kontrollinstanzen, deren Effekt und Funktion direkt nach der Aktion des Benutzers eintreten und sichtbar werden.

„Some command vectors offer more support to new users. Typically, menus and dialog boxes offer the most, which is why we refer to them as pedagogic vectors“. [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 94]. Neuen, unerfahrenen Benutzer bieten diese Elemente den Lerneffekt, die Funktionen des Programms näher zu ergründen, ohne durch Fehlklicks unerwünschte Ereignisse auszulösen.
Der Aspekt der *windows pollution* sollte dabei stets berücksichtigt bleiben. So sollte die Anzahl der Dialogboxen auf ein sinnvolles Minimum reduziert werden, um die Bildschirmfläche nicht mit Fenstern zu überladen. *„Achieving many user goals involves executing a series of functions. If there is a single dialog box for each function, things can quickly get visually crowded and navigationally confusing.“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 434]

Wesentliches Gestaltungsmerkmal einer selbsterklärenden und effektiven Benutzeroberfläche ist die Gruppierung sinnverwandter Funktionselemente. *„It turns out that there is an idiom that takes the best elements of tiled windows and provides them within a sovereign, full-screen application – the idiom of multipaned windows. Multipaned windows consist of independent views or panes that share a single window. Adjecent panes are seperated by fixed or moveable dividers or splitters. (...) The advantage of multipaned windows is that independent but related information can be easily displayed in a single, sovereign screen in a manner that reduces navigation and window managment excise to almost nil.“* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 428F]. Gemäß dieser Aussage und auf Grund einer sinnvollen übersichtlichen Funktionsaufteilung wurde das Prinzip der *multipaned windows* für die praktische Applikation dieser Arbeit angewandt.

Um die Dichte der darstellbaren Informationen und Funktionen weiter zu erhöhen, kann eine Verschachtelung der Themenfelder innerhalb eines Bereiches (pane) von Nutzen sein. *„A tabbed dialog allows programmers to cram more controls into a single dialog box, but more controls won't necessarily mean that users will find the interface easier to use ore more powerful. (...) Tabs are useful because the idiom follows many users' mental model of how things are normally stored."* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 524F]. In diesem Fall wurde das Prinzip nicht direkt für eine Dialogbox sondern für die direkt sichtbaren Bereiche der Oberfläche angewandt. Dabei darf nicht ausser Acht gelassen werden, dass eine thematische Relation zwischen den einzelnen Tabs bestehen sollte. *„The contents of various tabs must have a meaningful rationale for being together (...)"* [VGL. COOPER, REIMANN UND CRONIN, 2007, S. 94]. Es sollte auch auf eine moderate Anzahl an Verschachtelungen geachtet werden.

[VGL. COOPER, REIMANN UND CRONIN, 2007, S. 523-525]

Der Benutzer hat somit die Möglichkeit die wichtigsten Funktionen des Programms ohne viele Maus-Klicks zu überblicken und sich thematisch schnell auf der Oberfläche zu Recht zu finden.

V ENTWICKLUNG EINER LIVESTREAMING APPLIKATION

1 Konzept und Anforderungen

Ziel des praktischen Teils dieser Arbeit war es, eine Applikation zu entwickeln, die prioritär den Live-Aspekt des webbasierten Streamings berücksichtigt und unter den Gesichtspunkten des Usability-Engineerings eine adäquate grafische Benutzeroberfläche liefert. Basierend auf der bereits sehr starken Verbreitung von Video-on-Demand-Inhalten im Internet und umfassenden Abhandlungen zu dem Thema steht diese Übertragungsmethode nicht im Fokus dieser Arbeit und wird daher nur peripher betrachtet. Obgleich eine Kombination von Live-Streaming und VoD in der technischen Umsetzung Berücksichtigung findet.
Ein weiterer Differenzierungsaspekt in der Applikationsentwicklung ist die Option der simultanen Übertragung mehrerer Streams mit interaktiver Zugriffsmöglichkeit. Die Inhalte sollen sowohl vom Benutzer frei wählbar als auch von einem Administrator steuerbar sein. Dem Benutzer soll es überlassen sein, ob er in der Applikation interagieren will oder sich auf die Regie des Administrators verlassen möchte.

Die Interaktionsmöglichkeiten sollen deutlich im Vordergrund stehen, jedoch von Automatisierungsprozessen unterstützt und simplifiziert werden. Voraussetzung hierfür muss ein möglichst intuitives Bedienkonzept sowohl für den fachlich qualifizierten Administrator als auch für den konsumorientierten Endbenutzer sein.

Die Differenzierung zwischen administrativem Benutzer und Endbenutzer resultiert in einem zweistufigen Aufbau der Applikation. Zum einem muss die Möglichkeit bestehen, die Streams und alle weiteren in Relation stehenden Inhalte zu selektieren, verwalten und zu publizieren. Zum anderen muss eine möglichst funktionell-effiziente jedoch grafisch ansprechende Oberfläche generiert werden, die eine möglichst native Bedienung garantiert.

In Analogie zum Content Management System[G] in der Webentwicklung sowie zur Produktionskette in TV Rundfunkanstalten sind zwei unterschiedliche Programmversionen mit unterschiedlichem Funktionsspektrum erforderlich. Ein Back-End[G], welches die Kontrolle über alle Inhalte und Streams sicherstellt und die Weitergabe resp. Veröffentlichung der Daten ermöglicht und ein Front-End, welches mit dem Back-End interagiert und dem Enduser definierte Inhalte publiziert, auf die er möglichst uneingeschränkt und zielgerichtet zugreifen kann.

Als Kausalfolge dessen ergibt sich ein stark differierender Grad der Komplexität beider Programmversionen, sowohl in der Funktionalität als auch im Bedienkonzept. Um den Richtlinien des zielgerichteten Designs (User-Centerd-Design) nachzukommen, ist es also vonnöten, zwei unterschiedliche Anwendergruppen zu spezifizieren, die als Entwicklungsgrundlage für die jeweilige Programmebene dienen (s. Kapitel V Abschnitt 1.1).
Desweiteren ist zu berücksichtigen, dass unterschiedliche Themengebiete für die Darstellung interaktiver Video-Inhalte auch verschiedenartige Funktionsstrukturen und grafische Interaktionselemente erfordern.

1.1 Zielgruppenspezifikation

Die Anwendungsfelder eine Live-Streaming Anwendung streuen sich in diverse Bereiche, deren Anforderungen differenziert und definiert werden müssen. Die Vorgehensweise nach dem Prinzip des User-Centered-Design[G] impliziert eine genaue Definition der Zielgruppe und eines exemplarischen Benutzers. Diese detaillierte Spezifikation erfordert zudem nicht nur eine konkrete Modellierung eines fiktiven Benutzers sondern auch eine genaue Bezeichnung des Themengebietes der Applikation.
Ausgangsbasis der Überlegungen waren zunächst 3 Themengebiete die möglichst unterschiedliche Ansprüche repräsentieren sollen:

- Sport
- Nachrichten
- Entertainment

Die Disparitäten dieser Einsatzgebiete liegen vorrangig in der Art der grafischen Darstellung und des interaktiven Designs. Additional treten jedoch auch differenzierte technische Anforderungen zu Tage. Live-Streaming im Sportbereich unterstützt die Nützlichkeit von simultanen Übertragungen bspw. für die Regie gesteuerte Live-Darstellung eines Streams und gleichzeitiger Option der wiederholten Wiedergabe eines Ereignisses. Einen Schwellenbereich zum Nachrichten-Kontext bietet auch die Split-Screen Darstellung verschiedener oder gleichartiger Ereignisse, die an unterschiedlichen Austragungsstandorten stattfinden. Diese Situation überschneidet sich mit der klassischen Konferenzsituation, wie sie auch in wirtschaftlichen Umfeldern zu finden ist. Der hohe Grad der Aktualität ist bei einer Nachrichtenübertragung unerlässlich, aber auch bei der Darstellung eines kollektiven (z.B. Olympia mit mehreren Parallelereignissen) oder singulären Sportereignisses (z.B. Fussballspiel) zu berücksichtigen. Diese Anforderung ist bei einem Entertainment Angebot vergleichbar weniger stark ausgeprägt. Hier dominiert die Zugriffs- und Auswahlmöglichkeit möglichst vieler unterschiedlicher Inhalte zu beliebiger Zeit. Das Erfordernis simultaner Live-Streams ist dabei nicht zwingend gegeben, zumal die Bildinhalte schnitttechnisch meist äußerst aufwendig gestaltet sind und einen hohen Grad der Fokussierung des Betrachters erfordern. Mit stetig wachsenden Verbreitungsgebieten von High-Speed DSL[G] steigt auch die Nachfrage nach hoher Wiedergabequalität der Videoinhalte, was eine proportional steigende Auslastung der verfügbaren Bandbreite zur Folge hat. Singuläre oder duale Visualisierungssysteme (z.B. Bild-im-Bild) stehen in der Entertainment Thematik bislang noch im Vordergrund.

Bezugnehmend auf die eingangs dieses Kapitels erläuterte notwendige Modellierung zweier unterschiedlicher Benutzergruppen, ist es vor allem für die Definition eines Endbenutzer-Prototypen von Bedeutung, in welches thematische Umfeld das Front-End eingebettet ist. Um also die Ansprüche dieses Benutzers in den Entwicklungsprozess der Applikation einfließen zu lassen, ist die konkrete Auswahl eines Themenfeldes erforderlich.

Auf Grund der vielseitigen Möglichkeiten und des stark ubiquitären Interesses in der Bevölkerung wurde das Themenfeld Sport gewählt.
Das breite Interesse an sportlichen Ereignissen, ein wachsendes Bewusstsein für den Einfluss von Sport auf die psychisch und physische Gesundheit und Leistungsfähigkeit sowie die an-

steigende Zahl der Werbekampagnen für Breitensport spannen einen äußerst weitläufigen Personenkreis der Sportinteressierten auf.
Einschränkend für die Zielgruppenspezifizierung sind lediglich die Anwendungsumgebung (PC) und die Qualifikation der Zielgruppe im Umgang mit Medien wie dem Internet und Flash. Somit entstand folgendes Modell einer geeigneten repräsentativen Persona (Primary Persona) sowie deren Informationsziele (End goals):

Tom
26 Jahre alt
kaufmännische Ausbildung
Seit 5 Jahren Bürokaufmann in einem Elektronikunternehmen
Vorrangige Arbeit am PC
Hobbys: Sport (Fussball, Fitness), Musik, Freunde

Ziele: *„Ich will auf einen Blick sehen, was im Sport los ist“*
„Ich will die Bundesligakonferenz auch bildlich erleben“
„Ich will selbst Einfluss auf die angezeigten Inhalte nehmen können“

Dieser prototypische Benutzer stellt die Ausgangsbasis der zielgerichteten Entwicklung der vorgestellten Front-End Applikation dar. Die Ziele und Interessen dieser Persona, Tom, bilden das Bindeglied zwischen Programmierung und Design, also der Funktionalität und dem grafischen Interface.
Da die Applikation aus zwei differierenden Interfaces besteht, deren Funktionalität unterschiedliche Erfahrungswerte voraussetzt, ist die Definition einer zweiten primären Persona erforderlich. Die Ansprüche und Zielrichtungen dieser Persona unterscheiden sich stark von der ersten Persona. Jedoch unterscheiden sich auch die angeforderte Qualifikation und Erfahrung dieser Persona, da es sich um einen administrativ tätigen Back-End Benutzer handelt. Somit entsteht folgender zweiter Prototyp:

Markus
30 Jahre alt
Ausbildung Mediengestaltung Bild&Ton
Seit 10 Jahren Mitarbeiter in einer Rundfunkanstalt
Zuständig für Pflege der Online-Mediathek
Hobbys: Videoschnitt, Sport,

Ziele: *„Ich möchte schnellen Zugriff auf die verfügbaren Streams“*
„Ich will Kontrolle über alle publizierten Inhalte“
„Die Oberfläche muss klar und sinnvoll Strukturiert sein“
„Schnelle Betätigung der täglichen und wichtigen Funktionen auf einen Blick“

Die Interessen und Anforderungen dieser zwei Persona bilden das stets vorhandene Gedankengerüst während der Entwicklung der beiden Applikationselemente resp. -Interfaces.

1.2 Rahmenbedingungen, Funktionsumfang & Problemstellungen

Zunächst gilt es die Funktionalität des Front-Ends festzulegen, worauf im Weiteren die Funktionen des Back-Ends basieren und definiert werden können. Sofern deutlich ist, welche Interaktionsmöglichkeiten im Front-End benötigt werden und auf welche Weise die audiovisuellen Inhalte dargestellt werden sollen kann eine Aussage über die notwendigen administrativen Kontrollinstanzen des Back-Ends getroffen werden.
Die anfänglich postulierte Möglichkeit der simultanen Streamübertragung folgert eine Analyse der Darstellungsoptionen mehrerer Videos auf einer begrenzten Bildschirmfläche. Als Norm für die Größenverhältnisse findet in der Applikation das inzwischen weltweit akzeptierte und für besonders nativ befundene Bildseitenverhältnis von 16:9 Verwendung. Die Nähe zum menschlichen Wahrnehmungsapparat und Gesichtfeld und die Umstellung der größeren Rundfunkanstalten auf dieses Format unterstützen den intuitiven Charakter der Applikation. Somit wurde die Fenstergröße beider Systeme, Back-end und Front-End, auf 1600x900 Pixel festgelegt.
Maximal vier Videofenster sollen im Front-End gleichzeitige Stream-Darstellungen offerieren. Vor dem technischen Hintergrund und der Kapazität des FMS ist eine hohe Anzahl an simultanen Streams möglich, die Einschränkung stehen hier auf Seiten der Nutzbarkeit und Qualität der darzustellenden Inhalte, sowie den derzeit verfügbaren Datenübertragungsraten der DSL-Netze. Je mehr Streams generiert werden, desto mehr wird die Leitungsbandbreite ausgeschöpft und desto geringer wird die Leistungsfähigkeit. Wie ein Studie von „Deutschland-Online“ zeigt, bewegen sich über 60% der momentan 52,5Mio [Quelle: http://www.internetworldstats.com/stats4.htm#europe, Zugriff:03.12.08] DSL-Nutzer in Deutschland mit einer durchschnittlichen Geschwindigkeit von weniger als 6Mbit/s im Internet. Da davon auszugehen ist, dass die effektiv nutzbare und reelle Datenübertragungsrate darunter liegt, ist ein Wert von ca. 4-5Mbit/s anzunehmen. Geht man von einem Video mit einer Datenrate von mind. 320 kbps zuzüglich 96 kbps Audio aus, also insgesamt 416kbps Datenrate pro Stream, so erreicht die Applikation bei vier simultanen Streams durchschnittlicher Qualität eine Gesamtdatenrate von etwa 1664 kbps. Damit ist die Leitung mit knapp 2Mbit/s für die Streamübertragung ausgelastet und es existieren noch Reservern für höhere Video- und Audio-Datenraten. Die vier gleichzeitigen Streams stellen also einen Kompromiss zwischen Videoqualität und akzeptabler Auslastung der verfügbaren Datenübertragungsrate dar.

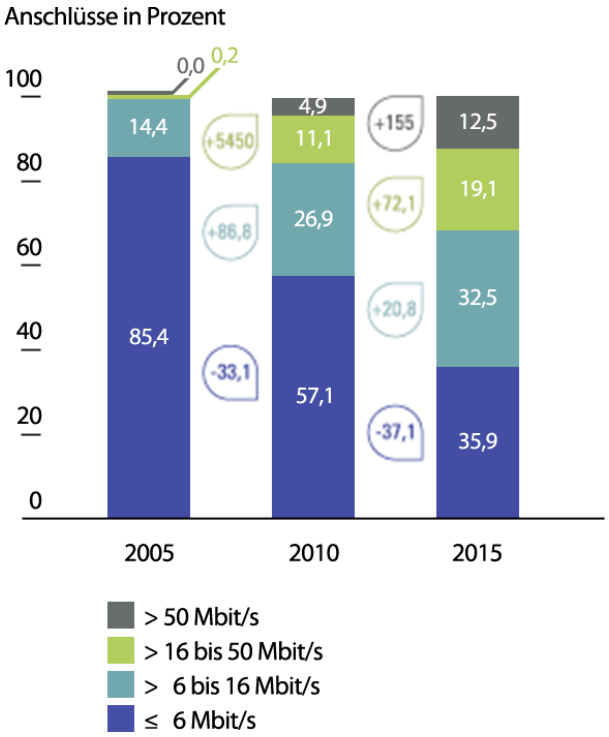


Abbildung V-1: Entwicklung der DSL-Bandbreiten bis zum Jahr 2015 (Breitband-Ökonomie-Experten)

[Quelle: http://www.studie-deutschland-online.de/do5/2200.html, Stand: 28.01.08]

Zusätzlich zu den audiovisuellen Daten sind auch Funktionen zum Abgleich zwischen FMS Back-End und Front-End nötig. So muss zum Beispiel eine dynamische Liste der verfügbaren Live-Streams sowie der verfügbaren gespeicherten Videodaten generiert werden und sowohl vom Back-End also auch vom Front-End aufrufbar sein. Hier sind unterschiedliche Zugriffsmethoden notwendig, da die Live-Streams temporär in den Arbeitsspeicher des Servers gestreamt werden und nur über einen sog. Mountpoint erreichbar sind.
Der Mountpoint entspricht einem Ordner der sich innerhalb des Applikationspfades befindet und beim Erzeugen des Live-Streams als Zielverzeichnis angegeben wird. Sofern kein spezieller Pfad angegeben wird, verwendet der FMS hier standardmäßig das Verzeichnis `streams` mit dem Unterordner `_definst_`. Über diese Pfadangabe ist eine Verbindung aus Flash mit dem FMS und den Live-Streaminhalten möglich. Listenelemente in der Applikation verknüpfen diese Live-Streams mit den gespeicherten on-Demand-Dateien, die sich im gleichen Ordner befinden, und visualisieren sie als formatierte Liste.

Ein Fokus sollte auch auf das Audiohandling gelegt werden. Simultane Videoübertragung bedeutet zwangsläufig auch simultane Audiowiedergabe. Gleichzeitige Tonereignisse lassen sich jedoch vom menschlichen Gehör nur schwer differenzieren geschweige denn unterschiedlichen Bildinhalten zuordnen. Daher ist es essentiell, dass stets nur ein Videokanal einen aktiven Audiokanal besitzen kann. Eine Lösung hierfür: Ein Klick auf das Videofenster nivelliert und aktiviert den entsprechenden Audioinhalt des Streams, was visuell durch ein Lautsprecher Symbol kenntlich gemacht wird. Alle anderen Kanäle werden stummgeschaltet. Diese Vorgehensweise findet sich im Back-End wie im Front-End.

1.3 Applikationselemente

1.3.1 Kontroll- und Administrationsmodul

Das Back-End lässt sich in vier Bereiche unterteilen:

1) Die vier öffentlichen Videokanäle
2) Die rechte Informationsleiste
3) Die Streamlisten
4) Das Vorschaufenster

(1)
Die öffentlichen Videokanäle (s. Abbildung V-2) stellen vier Streams dar, die vom Skript der Server-Seite generiert werden und von jedem Client verfügbar sind. Das besondere hierbei ist, dass der Inhalt der Videokanäle vom Administrator gesteuert werden kann und jeder Client den gleichen Inhalt sieht. Die Funktionsweise kommt also einem Broadcaster gleich. Über die zwei verfügbaren Schaltflächen wird direkt mit dem FMS kommuniziert und eine entsprechende Funktion zur Streamverarbeitung der öffentlichen Kanäle aufgerufen. Sie sind daher farblich besonders hervorgehoben, um die Immanenz des Klick-Ereignisses zu verdeutlichen.

Abbildung V-2: Bereich der vier öffentlichen Stream-Kanäle in der ControlCenter.fla Benutzeroberfläche

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

Der Stream, den es zu veröffentlichen gilt, wird aus dem Streamlisten-Bereich, welcher in Kapitel V Abschnitt 2.5.3.b detailliert beschrieben wird, selektiert. Das linke Textfenster bietet jeweils Aufschluss über die aktuelle Pufferlänge, die eingestellte Pufferlänge und die Spielzeit des Streams.

(2)
Die rechte Informationsleiste (s. Abbildung V-3) bietet über eine dreifache Reiternavigation folgende Informationen: Der erste Reiter stellt Daten zum aktuellen Verbindungsstatus zwischen Applikation und FMS sowie der Verbindung zur Administration-API des FMS zur Verfügung. Des Weiteren werden die Statusereignisse jeder Verbindung zum FMS protokolliert, um den Verlauf des Streamings (start, stop, buffer, etc.) verfolgen zu können.

Im zweiten Reiter sind vier Listen vertikal angeordnet, die jeweils die Meta-Daten der entsprechenden Videokanäle beinhalten. Hierbei werden alle enthaltenen Meta-Daten des Streams extrahiert, um eine komplette Übersicht über den Inhalt der Zusatzdaten des Streams zu offerieren.

Der dritte und letzte Reiter stellt die CuePoint[G]-Verarbeitung dar. Es handelt sich hierbei nicht nur um reines Informationsobjekt, sondern es erlaubt auch die Editierung der Cue-Points des ausgewählten Streams. Die CuePoints werden in einer externen XML-Datei gespeichert und jedem einzelnen Stream über den Namen zugeordnet. Einstellbar sind ein Cue-Point-Titel, die Zeit und bei Bedarf ein Offset, sofern der Startpunkt zeitlich versetzt werden soll.

Abbildung V-3: Seitliches Register der ControlCenter.fla. Aktivierter Reiter „Stream-Informationen“

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

(3)
Im unteren Bereich der Applikation befindet sich die Verwaltungseinheit der Streams (s. Abbildung V-4). Separiert durch vier Reiter stehen folgenden Funktionen zur Verfügung:

Das Element „Available Streams“ stellt in zwei Fenstern zum einen die Liste aller verfügbaren Streams, Live und VoD, sowie im unteren Bereich die Liste der zu veröffentlichenden Streams dar.

Die Registerkarte „Upcoming Streams“ präsentiert eine informative Liste der nächsten Live-Ereignisse. Es steht zudem eine Eingabemaske zur Verfügung, die den Eintrag eines neuen Events ermöglicht und erleichtert.

Der Bereich „Thumbnails“ zeigt die verfügbaren Miniaturbilder, die den Streams zugeordnet werden können. Die vertikale Unterteilung zeigt im oberen Abschnitt die Bilddarstellungen nebst Dateinamen, im unteren Abschnitt die Publizierungsliste aus Registerkarte 1. Diesen Streams können die Thumbnails[G] zugeordnet werden.

Vierter und letzter Reiter ist der nicht minder wichtige Bereich „Temporary Files“. In ihm werden in zwei vertikal getrennten Listen alle Dateien erfasst, die derzeit in den Applikationsordner geschrieben werden, sowie die fertig gestellten Mediendateien. Der Administrator hat hier die Möglichkeit die finalisierten Dateien in den regulären Streamordner zu übertragen.

Alle Inhalte der vier Registerkarten werden dynamisch generiert und stehen nach Freigabe bzw. Schaltflächenbestätigung durch den Administrator im Front-End für den End-User zur Verfügung.

(4)
Das Vorschaufenster (s. Abbildung V-5) ist mit dem zuvor beschrieben Menüpunkt „Available Streams“ verknüpft. Die Vorschau bezieht sich stets auf das Videoelement, welches in der Streamliste ausgewählt ist. Diese Funktion basiert auf der Überlegung, dass es wenig Sinn macht, einen Stream zu veröffentlichen, den man nicht vorher begutachtet hat oder dessen Inhalt man nicht im Vorfeld bereits kennt. Daher erscheint eine Vorschaufunktion unumgänglich, auch hinsichtlich der Vergabe von Cue-Points. Die Elemente des Vorschaufensters sind die Videoinstanz inkl. eines Regelbereichs mit den wesentlichsten Funktionen, sowie eine Textbox zur Darstellung der Meta-Daten.

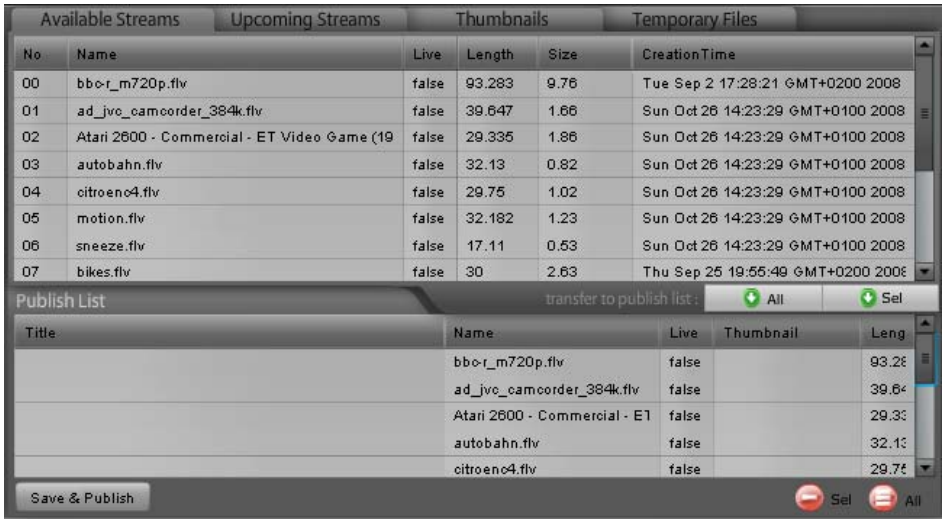


Abbildung V-4: Bereich der Verwaltungseinheit in der ControlCenter.fla Benutzeroberfläche. Aktivierter Reiter: „Available Streams“

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

Genauere Funktionserklärungen zu diesen wesentlichen Elementen können Kapitel V Abschnitt 2.5.3.b entnommen werden.

1.3.2 Die Endbenutzer Oberfläche

Die Oberfläche der End-Benutzerschnittstelle, das Front-End, besteht aus zwei grundsätzlichen Bestandteilen: Auf der einen Seite die Videofenster mit drei verschiedenen Anzeigemodi und auf der anderen Seite die Auswahlfelder der Streams sowie die Informationselemente. Zunächst soll der rechte Interface-Bereich vorgestellt werden. Dieser ist wiederum unterteilt in drei verschiedene Menüpunkte, die jeweils bestimmte Listen enthalten (s. Abbildung V-6).

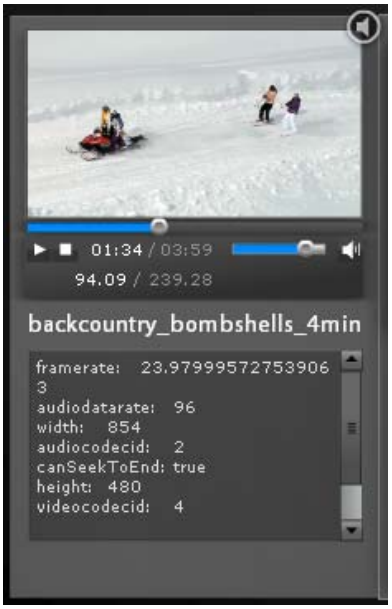


Abbildung V-5: Seitlich an die Verwaltungseinheit anknüpfender Vorschaubereich der Streams

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

Abbildung V-6: Liste der Live-Streams und Live-Events im Front-End.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

Wie im vorigen Abschnitt erwähnt, findet eine Kommunikation zwischen Front-End und Back-End statt. Das Resultat dieser Verknüpfung stellt der Inhalt dieser Listen dar. Ein Stream-Array, welches im Server-side-Skript zwischengespeichert ist, kann nun vom Front-End abgerufen werden. Die XML-Struktur wird geparst[G] und in die erste Liste übertragen. Vorab wird jedoch ein Filter angewandt, der Live-Streams von VoD Dateien separiert. Die obere `TileList` des ersten Menüpunktes wird mit den Live-Streams gefüllt. Alle anderen Streams, die den Live-Status nicht erfüllen, werden in die Liste des zweiten Menüpunkts geschrieben. So kann explizit zwischen „live“ und „gespeichert“ differenziert werden.

Weiterer inhaltlicher Aspekt des ersten Reiters ist die Anzeige der nächsten Events, die „`Next Live Events`“. Diese Elemente werden aus der externen XML-Datei `upclist.xml` geladen, welche sich nach der Erstellung durch das Back-End im Applikationsverzeichnis befindet.

Alle Informationen, die dem XML-Array bzw. den XML-Dateien aus dem Back-End hinzugefügt wurden, sind im Front-End verfügbar (s. Abbildung V-6 u. Abbildung V-7). Der Pfad zur Thumbnaildatei wird hier in einer verkleinerten Grafikdarstellung umgesetzt (gerendert[G]). Der nebenstehende Titel bezeichnet jedes einzelne Element. Zusätzlich werden Erstellungsdatum und –Zeit angezeigt. Der genaue Quellpfad zur Streamdatei, welcher für das Abspielen unerlässlich ist, befindet sich ebenfalls hinter jedem Listenelement, wird dem Nutzer jedoch

verborgen, da diese Angabe nur für die Funktionalität nicht jedoch für den Informationsfluss zum Enduser relevant ist.

Das gleiche Prinzip findet sich auch im zweiten Reiter, der „`Replay`"-Funktion, wieder (s. Abbildung V-7). Die Liste der gespeicherten Videostreams beinhaltet neben den vergleichbaren Informationen der Live-Liste zusätzlich eine Auskunft über die Dauer des Videos. Wesentliches Unterscheidungskriterium bildet die Clip-Auswahl, welche mit einem seitlichen Schieberegister realisiert wurde. Hier findet eine Kombination aus der Stream-XML-Datei und der `cuelist.xml`, also den CuePoints eines Videos, Anwendung. Die Einträge der Streamliste werden mit den Einträgen aus der CuePoints-XML abgeglichen. Sind relationale CuePoints einer Datei vorhanden, werden diese geladen und im Schieberegister dargestellt. Die Visualisierung der CuePoints enthält den Titel, den Einstiegszeitpunkt und bei Klick auf das Element die ersten Frames zum angewählten Zeitpunkt im Video. Die CuePoints stehen immer in Relation zum ausgewählten Element in der Streamliste.

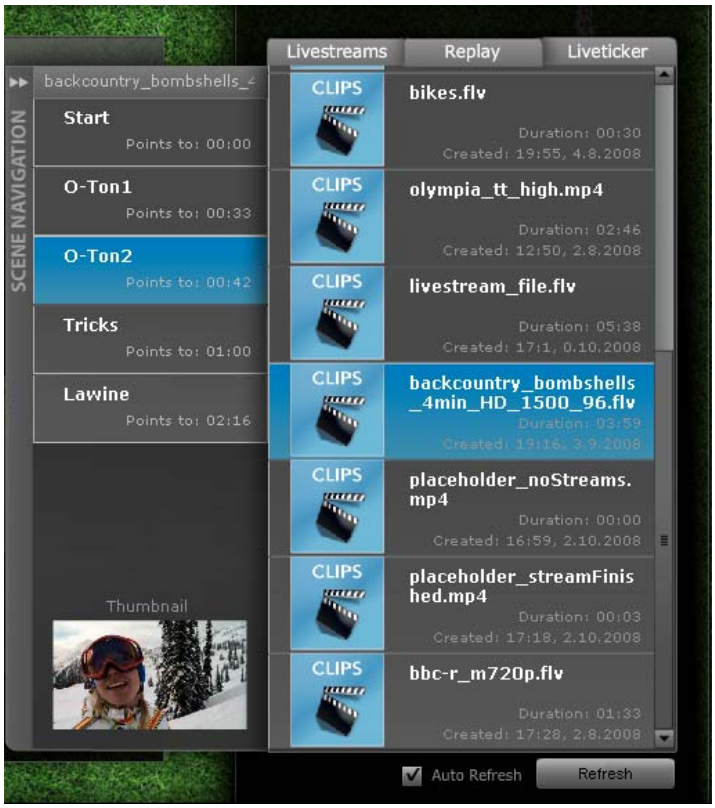


Abbildung V-7: Liste der VoD-Streams und den zugehörigen CuePoints im Front-End.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

Der letzte Menüpunkt bietet die Option eine externe RSS-Datei aus dem Internet einzubinden, um z.B. einen Liveticker zu offerieren. Prinzipiell könnten die Inhalte auch aus jeder anderen Datei im XML-Format geladen werden.

Um das User-Interface stets auf einem aktuellen Stand zu halten, läuft kontinuierlich ein Timer[G]-Ereignis in einem bestimmten Intervall ab, welches die Inhalte der Listen regeneriert und mit denen auf dem FMS bzw. im temporären Array ersetzt. Dieser Automatismus kann über die untere Auswahlbox deaktiviert bzw. durch eine Schaltfläche manuell initiiert werden.

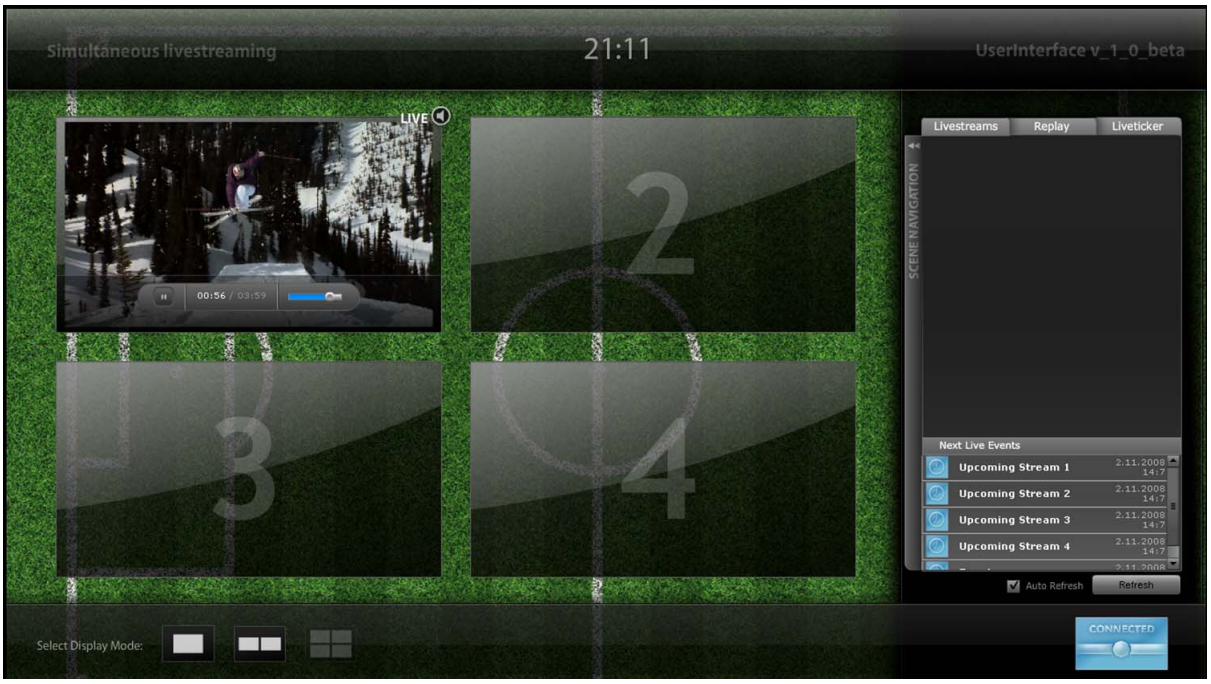


Abbildung V-8: Oberfläche des Front-Ends. Aktiver Darstellungsmodus: 4 Videofenster.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

Der Darstellungsbereich für die Videocontainer ist so skaliert, dass im dritten Anzeigemodus vier Videofenster über- und nebeneinander bei ausreichender Videogröße dargestellt werden können (s. Abbildung V-9). Die zweite Anzeigevariante reduziert die Anzahl auf zwei Videofenster. Der erste Anzeigemodus skaliert das aktuell laufende Video, welches durch den aktiven Tonkanal erkannt wird, annähernd auf die Größe des gesamten Darstellungsbereiches.

Fährt man mit dem Mauszeiger über ein Videofenster, so erscheint zentral ein Icon. Bei einem Mausklick auf selbiges wird der ausgewählte Stream in die Videoinstanz geladen und abgespielt. Handelt es sich hierbei um einen Live-Stream, wird dies durch eine Anzeige in der rechten oberen Ecke signalisiert. Der aktive Tonkanal wechselt automatisch auf das gerade ausgewählte Fenster.
Zusätzlich zeigt ein Maussensitives Feld im unteren Bereich des Videorahmens die aktuelle Spielzeit des Streams im Verhältnis zur Gesamtdauer (nur bei VoD), einen für jede Videoinstanz unabhängigen Lautstärkeregler, eine Play-Pause Schaltfläche und den Titel des aktuellen Streams an.

Abbildung V-9: Einzelnes Videofenster im Front-End mit aktivierter Videokontrolleinheit.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

Es handelt sich bei den soeben erwähnten Elementen ausschließlich um die essentiellen Videokontrollen, auf weitere Funktionen wurde aus Gründen der Übersichtlichkeit und der postulierten Simplizität der Bedienung verzichtet.

1.3.3 Erweiterungsoptionen

Grundsätzlich kann der Aufbau der Applikation als modular bezeichnet werden, da ein wesentlicher Knotenpunkt der Funktionalität im Skript auf Seiten des FMS liegt. Die Liste der Streams sowie aller anderen Informationen werden dynamisch generiert und sind somit modifizierbar. So könnte bspw. eine weitere Flashapplikation den Zugriff auf eine Webcam o.ä. ermöglichen und dieses Video auf dem FMS publizieren. Der Stream ist dann für den Administrator verfügbar und publizierbar.
Es sei an dieser Stelle auf die Schlussbetrachtung (Kapitel VI) verwiesen, in welcher bislang ungelöste Problemstellungen der Applikation diskutiert und optionale Funktionserweiterungen angeführt werden.

Der Stand der Applikation zum Zeitpunkt dieser Arbeit wird als Beta-Status festgelegt und sollte somit nach Beseitigung bestehender Fehler im Weiteren einer Evaluierungsphase unterzogen werden, die bislang unerkannte Fehler sowie unberücksichtigte Anwendungseinschränkungen aufdeckt und im Rahmen einer Contextual Inquiry, die als grundlegendes Prinzip für das Usability Design erklärt wurde, notwendig ist.

2 Realisation

Die technische Umsetzung der Applikation setzt einige Grundstrukturen voraus, welche die Lauffähigkeit einer Streaming Anwendung ermöglichen. Da dieses System auf einem lokalen Rechner aufgesetzt wird, ist zunächst die Installation eines php-fähigen Webservers erforderlich, der die lokale Dateistruktur über das HTTP-Protokoll oder andere Web-Protokolle, wie z.B. RTMP, erreichbar macht.

Die operative Umgebung stellt in diesem Fall ein MacBook Pro dar, welches mittels einer BootCamp^{G} Partition und einer Parallels Desktop Virtual-Machine^{G} eine native Microsoft Windows XP Plattform zur Verfügung stellt. Dadurch waren die Grundvoraussetzungen für die Einrichtung des nur unter Windows und Linux lauffähigen FMS geschaffen. Der FMS liegt für den konkreten Anwendungszweck in der Developer Edition 3.0 vor, die nahezu alle Funktionalitäten des kommerziellen FMS 3.0 bietet (vgl. Kapitel III Abschnitt 1). Der Flash Media Encoder 2.5 ist ebenfalls kostenfrei erhältlich und wird hier eingesetzt.
Einziges benötigtes kommerzielles Produkt ist die Software Adobe Flash, in diesem Fall Version CS3, die zur Erstellung der Applikation unabdingbar ist.

2.1 Installation des Apache Webservers

Die Installation des Webservers, in diesem Fall „Apache“, gestaltet sich dank fertiger Installationspakete äußerst simpel. In MS Windows XP ist kein Webserver vorinstalliert, daher ist dieser Schritt notwendig.
Unter der Adresse http://www.apachefriends.org/de/xampp.html kann der Open Source Webserver mit Installationspaket „XAMPP“ heruntergeladen werden. Weitere Systeme im Paket sind unter anderem MySQL^{G} für die Datenbankverwaltung und phpMyAdmin zur Bearbeitung der PHP Einstellungen. Auf eine manuelle Einrichtung, die unlängst aufwendiger ist kann also für diesen Zweck verzichtet werden.

Nach der Installation von XAMPP sollte in den Konfigurationseinstellungen der Autostart des Webservers bei jedem Systemstart aktiviert werden, um sicher zu stellen, dass der Apache Server kontinuierlich aktiv ist. Sind diese Schritte getan lässt sich das eigene Rechnersystem als Webserver verwenden. Geprüft werden kann dies über den Aufruf der URL: http://localhost/. Die URL verweist bei korrekter Installation auf das eigene Webserversystem und das XAMPP-Kontrollzentrum. Hier können ggf. die meisten Einstellungen des Serversystems vorgenommen werden.

2.2 Installation des Flash Media Servers

Auch die Installation des Flash Media Servers gestaltet sich dank fertigem windowstypischem Installer[G] äußerst einfach. Das Downloadpaket ist unter: http://www.adobe.com/products/flashmediainteractive/http://www.adobe.com/products/flashmediainteractive/ verfügbar.

Die Installation erfolgt über eine automatische Routine und erleichtert die meisten Schritte auf Grund der Voreinstellungen. Die hier geforderte Seriennummer (s. Abbildung V-10) ist nur für den Fall einer kommerziell erworbenen FMIS oder FMSS Version einzugeben. Da jedoch für den Rahmen dieser Arbeit die Development Edition genutzt wurde ist keine Seriennummer verfügbar und notwendig. Die Installation kann auch ohne Seriennummer fortgeführt werden, es wird folglich automatisch die Developer Edition installiert und konfiguriert (s. Abbildung V-10).

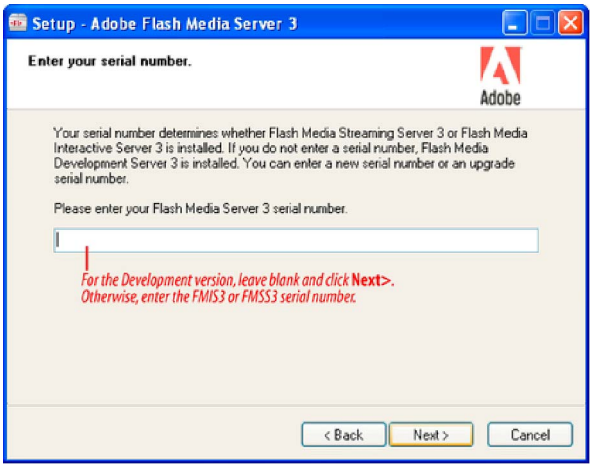


Abbildung V-10: FMDS3 Installation. Seriennummer-Dialogbox.

[Quelle: SANDERS, B. (2008), Kapitel 1 Abschnitt 1.3]

Der nachfolgende Schritt (s. Abbildung V-11) erfordert die Eingabe eines Benutzernamens inkl. Passwort für den Zugang zur Flash Media Administrations-Konsole genannt.
Die Administration-Konsole stellt sämtliche Informationen über den Flash Server zur Verfügung, visualisiert alle laufenden Prozesse und erlaubt die Kontrolle über Benutzer und Applikationen (vgl. Kapitel III Abschnitt 1). Über die Administration-API kann auf Funktionen der Konsole über Actionscript zugegriffen werden.
□

Der nächste Installationsbildschirm zeigt die Einstellung für die Netzwerk-Ports (s. Abbildung V-12). Die Voreinstellungen sollten hier beibehalten werden, sofern keine Kenntnis darüber besteht, dass eine andere Applikation bereits dieselben Ports nutzt. Hier könnten auch mehrere Ports vergeben werden. Dies ist z.B. sinnvoll wenn mit anderen Protokollen als dem RTMPE (RTMP) gearbeitet wird, wie dem RTMPT oder dem RTMPS (s. dazu Kapitel II Abschnitt 1.4).

Für die Zwecke dieser Arbeit ist es jedoch nicht von Nöten eine Vielzahl von Ports zu verwenden. Für den Verlauf der Programmierung ist jedoch der Administration Server port relevant (s. Abbildung V-12). Der hier verwendete Wert `1111` wird für die Verbindung aus der Back-End Applikation zur Administration-API benötigt. Da es sich bei dieser Verbindung um eine andere und unabhängig zu betrachtende Variante als die Verbindung zum FMS3 handelt, ist der differierende Port nun von Bedeutung.

[SANDERS, B. (2008)]

Die letzten Schritte bis zum Finalisieren der Installation des Flash Media Development Servers sind unroblematisch und können zügig durchgeführt werden.

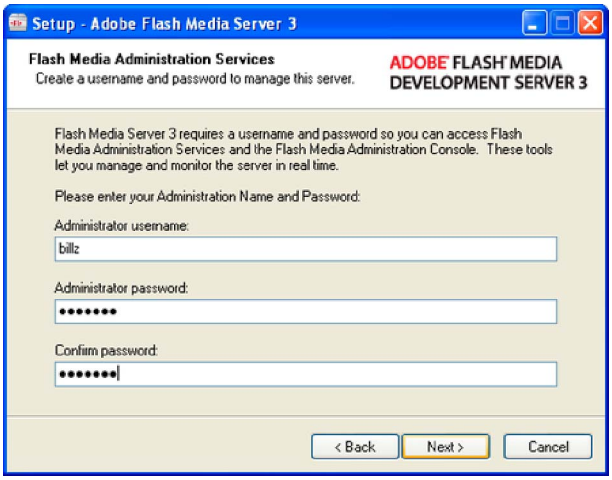


Abbildung V-11: FMDS3 Installation. Seriennummer-Dialogbox.

[Quelle: SANDERS, B.. (2008), Kapitel 1 Abschnitt 1.3]

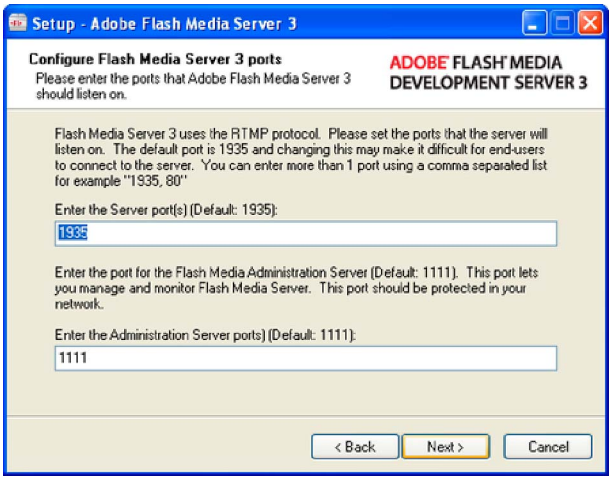


Abbildung V-12: FMDS3 Installation. Porteingabe-Dialogbox.

[Quelle: SANDERS, B.. (2008), Kapitel 1 Abschnitt 1.3]

2.3 Konfiguration und Modifikation des FMS

2.3.1 Virtuelle Verzeichnisse

Damit Flash-Applikationen vom FMS erkannt und somit freigegeben werden können, ist es notwendig die Flash-Dateien inkl. der zugehörigen Actionscriptdateien in die Unterordner `applications` des FMS-Installationsvezeichnises zu kopieren. Jede Applikation muss ihr eigens Unterverzeichnis erhalten, das alle Dateien beinhaltet.
Die Konfigurationsstruktur des FMS erlaubt es jedoch, dieses Anwendungsverzeichnis über das sog. `Virtual Host` System in jeden anderen beliebigen Ordner des Rechners zu delegieren. So können übersichtliche und informative Ordnerstrukturen erzeugt bzw. erhalten werden.
Im Unterordner `conf` der FMS-Installation lassen sich in der Datei `fms.ini` die Grundeinstellungen des FMS vornehmen. Neben den Benutzerangaben und Portinformationen findet sich

darin auch die Definition des `Virtual Host`, welcher den Pfad zu den Applikationsverzeichnissen deklariert.
In der vorliegenden Testumgebung wurde bspw. der `Virtual Host` wie folgt festgelegt:
`VHOST.APPSDIR = C:\DATEN\DIPLOMARBEIT\FMS\Server-side\applications.`

In der Datei `Vhost.xml` im relativen Unterverzeichnis `conf/_defaultRoot_/_defaultVHost_/` sind die Voreinstellungen für den `Virtual Host` gespeichert. Wird der jeweiligen Anwendung im eigenen Verzeichnis keine individuelle `Vhost.xml` hinzugefügt, die bspw. alternative Pfade definiert, erfolgt ein Rückgriff auf diese Standarddatei.
In Zeile 26 der `Vhost.xml` findet sich auch der zuvor definierte Applikationspfad in folgendem Code wieder: `<AppsDir>${VHOST.APPSDIR}</AppsDir>`.
Zusätzlich hierzu kann der Pfad zu den Streams innerhalb einer Applikation neu gesetzt werden. Im Abschnitt `<VirtualDirectory>` lässt sich über das `<Stream>` Tag einer Variablen, die über SSAS abrufbar ist, ein bestimmter Pfad zuweisen.
Der Aufbau ergibt sich wie folgt:

```
<Streams>/approot;C:\...\DIPL_streamwork\streams\studio\</Streams>
```

„...“ steht hierbei für den absoluten Pfad des Applikationsverzeichnisses, aus Gründen der Übersichtlichkeit wurde er durch „...“ ersetzt.
Prinzipiell ist folgende Struktur einzuhalten: Zu Beginn wird ein eindeutiger Variablenname definiert, hier `/approot`, der im weiteren Verlauf über die Actionscriptprogrammierung verfügbar ist. Nach dem trennenden Semikolon wird der absolute Pfad zum Streamordner angegeben. Dieser entspricht dem Livemountpoint bzw. dem Speicherort der VoD Dateien. Ab Kapitel V Abschnitt 2.4 wird die Nützlichkeit dieses Systems deutlich.

Analog zur `Virtual Directory` in der `Vhost.xml` ist auch die Deklaration einer solchen *Directory* in der `Application.xml` notwendig. Diese Datei befindet sich im selben Ordner und unterliegt den gleichen Bedingungen wie die `Vhost.xml`, kann also individuell mit jeder Applikation neu geladen werden. Die Angabe des Pfades dient hier jedoch nicht den Streams, sondern den Zugriffsparametern der `FileObject`-Klasse, dessen Funktion in Kapitel V Abschnitt 2.4.2 spezifiziert wird. Da sich alle relevanten Dateien und Ordner dieser Applikation aus Gründen der Übersichtlichkeit innerhalb des Streamordners befinden, sind die Pfade beider *Virtual Directories* identisch. Die Syntax ergibt sich wie folgt:

```
<FileObject>
      <VirtualDirectory>/approot;C:\...\DIPL_streamwork\streams\studio\
      </VirtualDirectory>
</FileObject>
```

Über das `FileObject` erlangt man somit Zugriff auf alle Unterordner dieses Verzeichnisses.

2.3.2 Zugriffsrechte

In der `Users.xml` im Ordner `conf/` findet sich die Vergabe der Zugriffsrechte zur Administration-API, die über das HTTP-Protokoll erreichbar ist. Hierbei geht es im Besonderen um die Administration-API-Befehle, die im Back-End von Bedeutung sind.
In den folgenden Codezeilen werden die für diese Applikation benötigten Befehle der Freigabeliste hinzugefügt:

```
<AdminServer>
<HTTPCommands>
<Enable>${USERS.HTTPCOMMAND_ALLOW}</Enable>
<Allow>ping,getLiveStreams,getLiveStreamStats,getApps,getAppStats
</Allow>
<Deny>All</Deny>
<Order>Deny,Allow</Order>
</HTTPCommands>
</AdminServer>
```

Der `<Allow>` Tag umklammert die freigegebenen Befehle. Im `<Deny>` Tag werden die verweigerten Befehle festgelegt, in diesem Fall zunächst *alle*. Der Abfolgepramater `<Order>Deny, Allow</Order>` sorgt dafür, dass zuerst die Verweigerung und anschließend die Freigabe ausgeführt werden. Ein wesentliches Element ist zudem der Tag `<Enable>${USERS.HTTPCOMMAND_ALLOW}</Enable>`,
welcher die Verwendung der Administration-API-Befehle erst ermöglicht. Dieses Element referenziert wiederum zur `fms.ini`, die im vorigen Kapitel bereits bearbeitet wurde. In der letzten Zeile der `fms.ini` verbirgt sich die wichtige Kommandozeile `USERS.HTTPCOMMAND_ALLOW = true`, die den Wert `true` erhalten muss, um die Kommunikation mit dem Server über das HTTP-Protokoll zu aktivieren.

2.4 Die Basis: Das Server-side-Skript

Das Server-side-Skript stellt den Knotenpunkt der Applikation dar. Hier werden die beiden Client-side-Interfaces untereinander und mit dem FMS verknüpft. Grundlegender Unterschied ist zudem die Syntax und der Dateityp. Das Server-side-Skript wird als Server-side Actionscript-Datei mit der Endung `.asc` immer im Ordner der Applikation auf Seiten des FMS abgespeichert. Die Actionscript-Syntax ähnelt der von Actionscript 1.0. Wohingegen die restlichen Actionscriptdateien dieser Applikation in AS 3.0 geschrieben sind und in den Client-side-Ordnern, die an beliebiger Position gespeichert werden können, zu finden sind.

```
application.onAppStart=function()
{
    trace(application.name + " loaded !");
};
application.onConnect=function(newClient,name,passwd)
{
    application.acceptConnection(newClient);
    var appName = application.name;
    var streamFolder = "studio";
    var CuePointXML;
```

Abbildung V-13: Quellcode-Auszug der main.asc. Initialisierung der Applikation

[Quelle: main.asc, Stand: 10.12.2008]

Initiiert wird die `main.asc` mit der `onAppStart`-Funktion, die mit dem Laden der Applikation aufgerufen wird. Verbindet sich ein Client mit der Applikation über `nc.connect` wird die Funktion `onConnect` aufgerufen, innerhalb welcher die meisten Funktionen dieses Server-side-Skriptes eingebaut sind (s. Abbildung V-13). Zunächst ist es jedoch von Nöten, dass der

```
// Generate 1. Stream. Uses Filter to validate playable filetypes.
//
Client.prototype.selectSource1 = function(actMediafile1,me_enabled)
{
    if (me_enabled == true) {
        if( (actMediafile1.substr(actMediafile1.indexOf("."), 4) == ".mp4") || (actMediafile1.substr(actMediafile1.indexOf("."), 4) == ".f4v"))
        {
              actMediafile1 = "mp4:"+actMediafile1.substring(0, actMediafile1.length-4);
        }
        else if( actMediafile1.substr(actMediafile1.indexOf("."), 4) == ".flv")
        {
              actMediafile1 = actMediafile1.substring(0, actMediafile1.length-4);
        }
        this.outgoingStream = Stream.get("stream1");
        this.outgoingStream.setBufferTime(0);
        this.outgoingStream.play(actMediafile1, -2);
        trace("Now playing @ Stream 1: "+actMediafile1);

    } else {trace("Stream 1 disabled"); this.outgoingStream.close();}
```

Abbildung V-14: Quellcode-Auszug der main.asc. Funktion selectSource1 zur Erzeugung des öffenltichen Streams „stream1".

[Quelle: main.asc, Stand: 10.12.2008]

verbindende Client von der Applikation zugelassen wird und somit Zugriff auf die Funktionen innerhalb des Skriptes erhält.
An die `onConnect`-Funktion wird die ID (automatisch vom FMS generierter numerischer Code) des Clients übergeben, sowie der Name der Applikation und optional ein Passwort, was in dieser Applikation jedoch nicht verwendet wurde.

2.4.1 Die Server-side Stream-Klasse

Im Rahmen der zuvor beschriebenen `onConnect`-Funktion werden die vier Streams erzeugt. Der Aufbau ist für alle vier Streams identisch, daher hier repräsentativ ein Codeauszug des ersten Streams (s. Abbildung V-14).
Es wird zunächst die Funktion `selectSource1` erzeugt, die zwei Übergabeparameter erwartet: Einerseits den Dateinamen des abzuspielenden Streams und anderseits einen boole'schen Wert, der angibt ob ein Stream generiert oder geschlossen werden soll.
Über eine `if`-Abfrage wird die Dateiendung des Videos herausgefiltert. Dies ist von Notwendigkeit, da für die Wiedergabe eines Stream über den FMS angegeben werden muss, welches Format gestreamt werden soll. Dateien mit MPEG-4 Codec müssen das Präfix `mp4:` erhalten, normale Flash-Videos bleiben ohne Präfix.

Der Befehl `outgoingStream.Stream.get(„stream1")` erzeugt eine Streaminstanz mit dem Namen `stream1` und ordnet diese der Variablen `outgoingStream` zu. Über diese Variable kann dem Stream jede beliebige Video- oder MP3-Datei zugeordnet werden. Angezeigt wird für den Stream jedoch immer der Name der Streaminstanz, nicht der der Mediendatei. Das macht diese Funktion besonders für Broadcasting-Zwecke adäquat.

Die Methode `setBufferTime(0)` setzt den Puffer serverseitig auf `0`, da die Puffer-Einstellungen in dieser Applikation im Client-Skript gesetzt werden. Schlussendlich wird die Videodatei über den Stream wiedergegeben, was über `outgoingStream.play(actMediafile1, -2)` realisiert wird. Der Parameter `-2` gibt hierbei an, dass die Datei von Beginn an abgespielt wird.

Wird dem 2. Parameter der `selectSource1`-Funktion der Wert `false` übergeben, so wird der bestehende Stream über den Befehl `outgoingStream.close()` geschlossen.

2.4.2 FileObject-Klasse - Methoden und Funktionen

Die `FileObject`-Klasse zählt neben der `NetStream`-Klasse zu den wichtigsten Elementen dieser Applikation. Alle Informationen über verfügbare Videodateien sowie die XML-Verarbeitung wird über die `FileObject`-Klasse in der `main.asc` realisiert.
Ein `FileObject` wird mit dem Befehl `new File(name)` erzeugt und einer Variablen zugewiesen. Der Parameter `name` gibt an, von welcher Datei bzw. von welchem Dateiordner das Actionscript Objekt erzeugt werden soll. Die `FileObject`-Klasse ermöglicht sowohl das Auslesen und Editieren von Dateien als auch von Ordnern.

```
newClient.createFileObj = function(name) {
    trace("Creating file object " + name );
    this.myFile = new File(name);              //Create new file, in this case its a directory

    if ( this.myFile != null ) return this.myFile.toString();
    else return "Failed";
}
```

Abbildung V-15: Quellcode-Auszug der main.asc. Funktion createFileObj zur Erzeugung eines FileObject.

[Quelle: main.asc, Stand: 10.12.2008]

Die Funktion `createFileObj` zum Erstellen des `FileObject` wird vom Back-End aufgerufen und dient nachfolgend der Zusammenstellung einer Liste aller Videodateien im Streamordner (s. Abbildung V-15). Zunächst wird hierfür der Inhalt der Datei, in diesem Fall der Inhalt des Ordners, dem Objekt `myFile` zugewiesen.

```
//Filter playable Videofiles from the gathered files
function filter(name) {
    if ( name.lastIndexOf( ".flv") != -1 || name.lastIndexOf(".mp4") != -1 || name.lastIndexOf(".f4v") != -1){
        return true;
    }
        return false;
}

//create list of the videostreams and return the results to the invoking client (nc.call from ControlCenter)
newClient.dir = function()
{
    trace("newclient.dir is working");
    //set dirList to the array with a filter attached
    var dirList = this.myFile.list(filter);

    trace("RESULTS FROM FileObject .dir");
    //show processing files in admin-console and get stream length
    for( var i = 0; i < dirList.length; i++)
    {
        var index           = dirList[i].name.lastIndexOf(".");
        var stream_name     = dirList[i].name.substring(index + 1, dirList[i].name.length) + ":" + dirList[i].name.substring(0, index);
        var vidExtension    = dirList[i].name.substring(index + 1, dirList[i].name.length);

        trace("Class:\t\t VIDEO\nType:\t\t " + vidExtension + "\nFilename:\t " + (dirList[i].name) +"\n--");
        dirList[i].streamlength = Stream.length(stream_name);

    }
    return dirList;
}
```

Abbildung V-16: Quellcode-Auszug der main.asc. Funktionen zur Filterung und Auflistungen der Dateien des FileObject.

[Quelle: main.asc, Stand: 10.12.2008]

Nachdem das `FileObject` erzeugt wurde, kann aus dem Back-End die Funktion `dir` aufgerufen werden, die alle gesammelten Dateien in `myFile` in ein Array schreibt und den Inhalt an das Back-End zurückgibt. Dies geschieht über die Befehlszeile `var dirList = this.myFile.list(filter)`. Der Befehl `list` erzeugt aus dem `FileObject` das Array der Dateien (s. Abbildung V-16). Ohne den Parameter `filter` würden hier jedoch alle Dateien aufgeführt, die der Ordner bzw. das Objekt beinhaltet, so auch z.B. die windowstypische `Thumbs.db` oder andere Non-Video-Dateien. Daher müssen die verwendbaren Videodateien

aus der Dateiliste extrahiert und von den anderen Dateien separiert werden. Dies geschieht in der Funktion `filter` über eine `if`-Abfrage, die jene Dateien herausfiltert, deren Endung eine Eignung für Videostreaming impliziert.
Dieses Prinzip wird auch für die Erzeugung der Thumbnail-Liste und der Liste der temporären Dateien angewandt und ist somit für eine weitere Erläuterung redundant.
Der Funktionsumfang für die temporären Dateien umfasst jedoch noch einen weiteren Punkt, den Transfer der Dateien in ein anderes Verzeichnis.
Die Funktion `moveTempFile` startet wiederum nur nach Aufruf aus dem Back-End und leitet den Verschiebungsvorgang ein (s. Abbildung V-17). Die übergebenen Parameter sind durch die Variablennamen weitestgehend selbsterklärend: Die zu transferierende Datei `thisFile`, der Zielordner `targetFolder` und der Quellordner `sourceFolder`.

```
//Move File if streaming is finished (invoke from ControlCenter)
Client.prototype.moveTempFile = function(thisFile, targetFolder, sourceFolder) {
    trace("Creating file object " + thisFile );
    this.myTempFile = new File(sourceFolder+thisFile);

    trace("Moving "+thisFile+" to "+targetFolder);
    var copyToTarget = targetFolder+"/"+thisFile;
    trace("copyToName: "+copyToTarget);
    this.myTempFile.copyTo(copyToTarget, false);
    this.myTempFile.remove();
    this.myTempFile.close();
}
```

Abbildung V-17: Quellcode-Auszug der main.asc. Hier die Funktion moveTempFile, die die ausgewählte Datei in den regulären Streamordner verschiebt.

[Quelle: main.asc, Stand: 10.12.2008]

Ausgangsbasis ist auch hier die Erzeugung eines `FileObject`, hier `myTempFile`. Der Befehl `this.myTempFile.copyTo(copyToTarget,false)` kopiert die Datei in den Zielordner, `false` verhindert ein eventuelles Überschreiben einer gleichnamigen Datei, ggf. kann hier der Parameter `true` sinnvoller sein. Die Methode `myTempFile.remove()` löscht die Datei im Quellordner nach dem Verschieben und `myTempFile.close()` schließt das `FileObject` wieder, was grundsätzlich bei jedem `FileObject` nach der Bearbeitung ausgeführt werden sollte, da die Datei so für andere Zugriffe wieder freigeben ist.

Die Verarbeitung der XML-Dateien findet auf zwei unterschiedliche Arten statt:

1. Die Übergabe des XML-Arrays aus dem Back-End an eine Funktion im Server-side-Skript, die dieses Array wiederum in einer globalen Variable zwischenspeichert, welche über eine zweite Funktion aus dem Front-End abrufbar ist.
2. Die 2. Variante arbeitet nicht mit einem zusätzlichen Array sondern speichert das übergebene XML-Array in einer externen XML-Datei. Hierzu kommt abermals das `FileObject` zum Einsatz, welches auch zum „Schreiben“ eingesetzt werden kann.

Der folgende Quellcode zeigt das Skript zum Transfer der *„PublishList“* und beinhaltet beide oben beschriebenen Varianten (s. Abbildung V-18).

Die Funktion `setPublishList` erwartet zwei Parameter, zum einen das XML-Array mit der Liste der Streams und zum anderen den Zielspeicherort inkl. Dateiname für die externe XML-Datei. Das Array wird zunächst in die server-seitige globale Variable `PublishListXML` übertragen, die über die `getPublishList` Funktion an den Client zurückgegeben wird.
Der weitere Code dient lediglich der Erzeugung der XML-Datei. Hier wird zuerst ein `FileObject` erstellt, welches auf dem vorab übergebenen Zielordner samt Dateinamen basiert und daraufhin über den Befehl `this.XMLFileIbjPub.open(„utf8“,“create“)` eine neue Datei mit dem angegebenen Namen generiert. Der erste Parameter der Methode `open` definiert den Zeichensatz, mit welchem die neue Datei formatiert wird, der zweite Parameter sorgt in diesem Fall dafür, dass die Datei neu erzeugt wird.
Durch die Abfrage `if(this.XMLFileObjPub.exists == true)` wird überprüft, ob eine gleichnamige Datei bereits besteht, was zur Folge hat, dass diese zunächst gelöscht und daraufhin neu erstellt wird. Dies kommt einem Überschreibungsprozess gleich.
Die Inhalte des `PublishListXML`-Arrays werden letztendlich über `this.XMLFileObjPub.writeln(PublishListXML)` zeilenweise in die neu kreierte Datei geschrieben. Auch hier wird abschließend das `FileObject` wieder geschlossen. Das Resultat ist eine XML-Datei, die im angegebenen Server Verzeichnis abgelegt und beschrieben wurde.

```
//set global variable for access in both funtions below
var PublishListXML;

Client.prototype.getPublishList = function() {
        return PublishListXML;
}
Client.prototype.setPublishList = function(PublishList, saveDestPub) {
        PublishListXML = PublishList;
        this.XMLFileObjPub = new File(saveDestPub);
        trace("Creating XML file object " + saveDestPub);

        if (this.XMLFileObjPub.exists == true) {
            //Overwrites the existing XML file !
            this.XMLFileObjPub.open("utf8","create");
        } else {
            this.XMLFileObjPub.open("utf8","create");
        }
        this.XMLFileObjPub.writeln(PublishListXML);
        this.XMLFileObjPub.close();
}
```

Abbildung V-18: Quellcode-Auszug der main.asc. Funktionen zur Übertragung der Arrays vom Back-End zum Front-End.

[Quelle: main.asc, Stand: 10.12.2008]

Das Verfahren ist redundant für die Generierung der CuePoints- und Temporary-XML-Dateien. Das soeben Beschriebene stellt einen Auszug der relevantesten Funktionen der `main.asc` dar und ist für die Kommunkation zwischen Front- und Back-End unerlässlich. Eine Erweiterungsoption liegt ggf. in einer dynamischen Kontrolle der Datenübertragunsraten (bandwidth-control). Dies schließt eine Analyse der Übertragungsrate des jeweiligen Clients und eine nachfolgende Anpassung der Datenrate des Streams ein.

2.5 Das Administrationsmodul

Das Back-End wird wie das Front-End von der Client-Seite aufgerufen, kann also von jedem beliebigen Rechnersystem mit dem Adobe FlashPlayer ab Version 9 sowie einem Internetbrowser gestartet werden. Das grundlegende Prinzip beider Interfaces (Front- und Back-End) ist identisch. Die grafischen Elemente werden in einer `.fla` Datei erstellt und platziert, alle Funktionen und Methoden sind in der zugehörigen Dokumentenklasse (`ControlCenter.as` und `UserInterface.as`) im AS3.0 Code definiert.
Die nachfolgenden Erläuterungen stellen auf Grund des sehr umfassenden Quellcodes daher nur die wichtigsten Funktionen heraus und sollen einen Überblick über die generelle Systematik der Applikation bieten. Im Hinblick auf den begrenzten Zeitrahmen des Projektes wurde

darauf verzichtet, Kernfunktionen in externe Actionscript Pakete auszulagern, was den Quellcode zugegebenermaßen stark anfüllt. Hier sollten künftige Optimierungen ansetzen.

2.5.1 Grundlegende Funktionen und Eigenschaften

2.5.1.a Konstruktor und Initialisierungen

Zu Beginn der `ControlCenter.as` Datei gilt es, alle global verwendeten Variablen zu deklarieren sowie alle zusätzlich von Flash benötigten Pakete einzubinden (s. Abbildung V-19). Denn der Zugriff auf Elemente wie die `NetStream`-Klasse oder das `Video`-Objekt ist nur nach vorheriger Einbindung des entsprechenden Paketes möglich, da in einem leeren Actionscriptdokument die Verarbeitung dieser Funktionen zunächst nicht vorgesehen bzw. möglich ist.
Wie bereits vorab angedeutet wurden noch keine Auslagerungen der umfangreichen Funktionen in externe Actionscriptdateien vorgenommen, die wiederum als Pakete eingebunden werden könnten. Daher ist der Aufbau des *Konstruktors* verhältnismäßig lang.
Die Definition der Verzeichnisse dient der Variabilität des Programms und ist besonders für die Verarbeitung der XML-Dateien wichtig.

```
//Directories & File Locations
        private var serverName          = "localhost";                          //name/IP of the Apache webserver
        private var appName             = "DIPL_streamwork";                    //name of the server-side application (-folder)
        private var streamFolder        = "studio";                             //name of the instance (=streamfolder) inside the app
        private var folderName          = "/approot";                           //virtual directory name for the app
        private var imgfolderName       = "/approot/thumbs";                    //virtual directory for the thumbnails
        private var tempfolderName      = "/approot/temp";                      //virtual directory for the temporary streams
        private var SfinishedVideo      = "placeholder_streamFinished.mp4";     //filename of the placeholder video
        private var thumbFol            = "thumbs/";                            //directory name of the thumbnails, used for the XMLs
        private var thumbUPC            = "ThumbUpc.png";                       //standard thumbnail-file used for the upcoming events
        private var thumbStored         = "ThumbStored.png";                    //standard thumbnail-file used for stored videos
        private var thumbLive           = "ThumbLive.png";                      //standard thumbnail-file used for live streams
        private var instanceName        = "DIPL_streamwork/studio";             //combination of appname & instancename
        private var appRootAbs          = "C://DATEN/DIPLOMARBEIT/FMS/server-side/applications/DIPL_streamwork/streams/studio/";    //abs
        private var XMLFolder           = "/approot/xml/";                      //virtual directory for the XML files
        private var pubXMLFilenmae      = "streams.xml";                        //filename of the PublishList XML
        private var cueXMLFilename      = "cuelist.xml";                        //filename of the CuePoint XML
        private var upcXMLFilename      = "upcominglist.xml";                   //filenmae of the Upcoming XML
        private var saveDestPub         = XMLFolder + pubXMLFilenmae;           //save target for the PublishList XML
        private var saveDestCue         = XMLFolder + cueXMLFilename;           //save target for the CuePoint XML
        private var saveDestUpc         = XMLFolder + upcXMLFilename;           //save target for the Upcoming XML
```

Abbildung V-19: Quellcode-Auszug der CotrnolCenter.as. Variablendeklarationen im Konstruktor .

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Die nebenstehenden Funktionen werden automatisch mit dem Start der Applikation innerhalb der Hauptfunktion der Dokumentenklasse geladen und bilden die Voraussetzung für die Lauffähigkeit der Anwendung (s. Abbildung V-20). Es werden die Formatierung der Textdarstellungen festgelegt, die Objekte der Benutzeroberfläche ausgerichtet und aus- bzw. eingeblendet (bspw. die Ladeanzeigen etc.) und zuletzt die Verbindung mit dem FMS über die `connectHandler`-Funktion eingeleitet.

```
//Initial functions
    formatting();                   //Define Textformats
    setGUI();                       //User Interface Set up
    connectHandler();               //FMS Connection Handler
```

Abbildung V-20: Funktionsaufruf der Initialisierungsfunktionen aus der Konstruktorfunktion.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Die Reaktion der Anwendung bei Interaktion des Benutzers wird über sog. `EventListener` festgelegt. Diese definieren die Verhaltensweise nach einem Ereignis, das von einer anderen Funktion oder vom Benutzer ausgelöst wurde. Da in dieser Anwendung das Interagieren des Benutzers mit der Anwendung von essentieller Bedeutung ist, müssen eine Vielzahl dieser

EventListener erzeugt werden, ein Auszug dieser Funktionen findet sich im Anhang (s. Abbildung A-2).

2.5.1.b NetConnection und NetStream als Fundament

Die Konnektivität des Back-Ends mit dem FMS wird durch die Funktion `connectHandler` hergestellt. In der Variablen `rtmpNew` wird der Pfad zum FMS gespeichert (s. Abbildung V-21). Die globale Variable `nc` ist vom Typ `NetConnection` und das Bindeglied zwischen Applikation und FMS. Über `nc.connect(rtmpNew)` wird die Verbindung aufgebaut. Der `EventListener` `nc.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler)` speichert die Statusmeldungen der Verbindung, was im Detail in Kapitel V Abschnitt 2.5.1.c erörtert wird.

```
/*-------------------------------------------------------
    CONNECT Handling functions: Connect / Close / Clear
-------------------------------------------------------*/
            /*  Connect to the server.  */
            public function connectHandler():void {
                trace("- Okay, let's connect now");
                setVid(6);                  //6 = forces visibility of all video instances
                //Connect to FMS
                rtmpNew = "rtmp://"+serverName+"/"+appName+"/"+streamFolder;
                nc      = new NetConnection();
                nc.connect(rtmpNew);
                nc.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

                //RefreshTimer for Buffer & Time Display for each Channel (1-4)
                var chInfoTimer:Timer = new Timer(DISPLAY_TIMER_UPDATE_DELAY, 0);
                chInfoTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                chInfoTimer.start();

                //Connect to Administration Interface to get list of currently running streams (Livelist)
                nc_admin = new NetConnection();
                nc_admin.connect("rtmp://localhost:1111/admin", "admin", "1qayse4RFV");
                nc_admin.addEventListener(NetStatusEvent.NET_STATUS, ncadminStatusHandler);

                //FileObject Handling
                createFileObj();            //Creat server-side FileObjects
                createImgFileObj();
                createTempFileObj();
                dir();                      //get stored Video file list from server-side
                tempDir();                  //get list of currently saving stream files
                loadUpcomingList();         //load upcoming streams XML file to DataGrid
            }
```

Abbildung V-21: connectHandler()-Funktion. Aufbau der Verbindung zum FMS .

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Die zweite Verbindung, `nc_admin`, dient der Kommunikation mit der Administration-API. Hier ist es erforderlich auch den Port, der bei der Installation des FMS festgelegt wurde, in die Adresszeile aufzunehmen, sowie als weitere Parameter den Benutzernamen und das Passwort des Administrators zu übergeben (vgl. Kapitel V Abschnitt 2.2). Andernfalls sind die Verbindung mit der Konsole und das Ausführen der Funktionen in diesem Zusammenhang nicht möglich. Auch für `nc_admin` wird der Status der Verbindung protokolliert, um eventuelle Fehlschläge der Anmeldung zu detektieren. Abschließend werden die `FileObject`-Funktionen ausgeführt, die die zugehörigen Methoden in der `main.asc` aufrufen, sowie deren Wiedergabewerte verarbeitet. Mehr dazu in Kapitel V Abschnitt 2.5.2.

```
/*----------------------------------------
    GENERATE INITIAL STREAMS TO FMS
----------------------------------------*/
private function connectStream(nc:NetConnection):void
{
    ns1=new...

    ns2=new NetStream(nc);
    bufferInitialize(ns2);
    ns2.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
    ns2.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
    ns2.play("stream2");
    vidStream2.attachNetStream(ns2);
    lbl_stream2.text = "Channel 2";

    ns3=new...

    metaCheck(6);
    setActiveAudio(intLastVolume,5);        //Define the Preview Stream a:
    // switch play/pause button visibility
    mcVideoControls.btnPause.visible    = true;
    mcVideoControls.btnPlay.visible     = false;

}
```

Abbildung V-22: connectStream()-Funktion. Aufbau der Streams zum FMS über die NetStream Klasse.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Sind die Voraussetzungen nach Kapitel V Abschnitt 2.5.1.c erfüllt, kann die Verbindung der Streams aufgebaut werden. Dies erfolgt über die `connectStream()`-Funktion (s. Abbildung V-22). Analog zur `NetConnection`-Klasse wird eine Variable des Typs `NetStream` über `ns2=new NetStream(nc)` erzeugt, obgleich dieser zusätzlich ein Parameter übergeben wird, das `NetConnection`-Objekt `nc`. Denn der Stream muss einer bestehenden Verbindung zwischen Applikation und FMS zugeordnet werden. Bereits obligatorisch sind die `EventListener`, hier eingesetzt zur Verarbeitung der Statusmeldungen des Streams und eventueller Fehlermeldungen. Zur Wiedergabe einer Mediendatei über den Stream `ns2` dient die Funktion `ns2.play(`*`dateiname`*`)`. Als Dateiname wird entweder der Stream-Name der Server-side Streams (`stream1 - stream4`) oder der Name einer beliebigen anderen Mediendatei als String[G] übergeben, wobei auch hier das Präfix `mp4:` für Medieninhalte mit MPEG-4 Kodierung berücksichtigt werden muss. Optional können der Methode `play(arguments...)` noch weitere Parameter übergeben werden, so zum Beispiel Integerwerte[G], die Aufschluss darüber geben, ob es sich um Live- (`-2`) oder VoD-Inhalte (`-1`) handelt. Wird auf diese Argumente verzichtet, wie in diesem Fall, geht das `NetStream`-Objekt zunächst von Live-Inhalten aus und stellt andernfalls automatisch auf die Wiedergabe von VoD-Inhalten um. Als dritter Parameter kann der Einstiegszeitpunkt der Wiedergabe (funktioniert nur für VoD-Inhalte) eingestellt werden.
Der Stream `ns2` kann nun über `vidStream2.attachNetStream(ns2)` der Video-Instanz `vidStream2` zugeordnet werden, welche den Inhalt des Streams abspielt, sofern es sich um audiovisuelle Daten handelt. Dies wird für alle fünf Streams (die vier öffentliche Videokanäle und der Vorschau-Kanal) in gleicher Weise ausgeführt.
Der Aufruf `metaCheck(6)` leitet zu den Meta-Daten verarbeitenden Funktionen, die in Kapitel V Abschnitt 2.5.1.d im Detail betrachtet werden. Abschließend wird der aktive Audio-Kanal über den Befehl `setActiveAudio(intLastVolume, 5)` festgelegt (s. Abbildung V-23). Um mehrere aktive Audiokanäle und das daraus resultierende Klanggemisch zu vermeiden, ist es nötig, nur einem Stream die Audiowiedergabe zu ermöglichen. Die Funktion `setActiveAudio` leistet dies und nutzt dazu das `SoundTransform`-Objekt. Sie kann sowohl aus einer anderen Funktion, also auch durch Mausklick auf ein Videofenster der Benutzeroberfläche aufgerufen werden. Hierzu extrahiert die Funktion `setAudio()` den Index des anklickten Videofensters, bestimmt hieraus welcher Kanal aktiviert werden soll und übergibt diesen Wert als zweiten Parameter der Funktion `setActiveAudio()`. Das erste Argument definiert die Audiolautstärke, die auf ein Intervall von `0` bis `1` normiert ist.

```
/*----------------------
    AUDIO Handler
----------------------*/
        private function setAudio(e:MouseEvent):void{
            var Index:int = (e.target.parent.name.substr((e.target.parent.name.length-1), 1));
            if (root["ns"+Index]!=null) setActiveAudio(1,Index);
        }
        private function setActiveAudio(vol:Number,index:int):void {
            var sndTransform2 = new SoundTransform(0);
            var sndTransform = new SoundTransform(vol);
            for (var i:int=1; i <= sAmount+1; i++) {
                if ((i != index) && (root["ns"+i] != null)){
                    root["ns"+i].soundTransform = sndTransform2;
                }//end if
            }//end for
            if (root["ns"+index] != null) root["ns"+index].soundTransform = sndTransform;
            var actAudCh = index;
            posAudioIcon(actAudCh);
        }//End function setActiveAudio

        private function posAudioIcon(index:int):void {
            mc_ActiveAudio.x = root["video_thumb"+index].x + root["video_thumb"+index].width - (mc_ActiveAudio.width/2);
            mc_ActiveAudio.y = root["video_thumb"+index].y - (mc_ActiveAudio.height/2);
        }
```

Abbildung V-23: setAudio()- und posAudioIcon()-Funktion. Festlegen des aktiven Audiokanals und Positionierung des Audio-Icons.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Um die Audiodaten eines Streams zu bearbeiten, kommt die `NetStream`-Eigenschaft `soundTransform` und die Klasse `SoundTransform()` zum Einsatz. Mit ihr lässt sich die Lautstärke sowie das Panning[G] zwischen den Stereo-Kanälen einstellen. Der erste Parameter legt die Lautstärke fest, der zweite die Kanalgewichtung. In dieser Funktion wird dem aktiven Stream der mit der Variablen `vol` übergebene Lautstärkewert zugewiesen, die anderen Streams erhalten in der `for`-Schleife den Wert `0`, was der Stummschaltung gleicht. Die `SoundTransform()`-Parameter werden in der Variablen `sndTransform` zwischengespeichert und über `root[„ns"+index].soundTransform = sndTransform` dem zu aktivierenden Stream zugewiesen.
Um den aktiven Audiokanal zu kennzeichnen, wird mit der Funktion `posAudioIcon()` an der oberen rechten Ecke des entsprechenden Videofensters ein Lautsprechersymbol eingeblendet, so dass der Benutzer stets in Kenntnis darüber ist, zu welchem Video der Ton korreliert.

Ändert der Benutzer zur Laufzeit die Inhalte eines Streams, so wird hierfür die Funktion `changeStream` genutzt (s. Abbildung V-24). Grundgerüst ist abermals die Extraktion des Index' der angeklickten Instanz, um zu definieren, welcher Stream bearbeitet werden soll. Dies setzt eine Instanzbenennung voraus, die als letzten Teilstring immer eine Ziffer von 1-5 enthält.

```
/*-----------------------------------------
    CHANGE STREAM HANDLER
-----------------------------------------*/
//Listitem Change Handler
private function changeStream (e:MouseEvent):void
{
    var targetItem:String = e.target.name;
    var itemIndex = (targetItem.substr((targetItem.length-1), 1));  //get last string = IndexNumber
    var currentResource:String = tab1.dg_StreamList.selectedItem.Name;
    if (root["ns"+itemIndex] != null) root["ns"+itemIndex].close();

    if (parseInt(itemIndex) != 5) {
        setVid(parseInt(itemIndex));                        //make Video Instance visible
        //root["ns"+itemIndex] = new NetStream(nc);         //Build up new NetStream (Clears the Buffer)
        root["ns"+itemIndex].play("stream"+itemIndex);
        root["vidStream"+itemIndex].attachNetStream(root["ns"+itemIndex]);
        root["ns"+itemIndex].addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
        root["ns"+itemIndex].addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
        root["ns"+itemIndex].bufferTime = startBufferLength;    //Initialize Buffer for Quick-Play

        root["lbl_stream"+itemIndex].text = currentResource;    //Call server-side NetStream
        nc.call(("selectSource"+itemIndex),null,currentResource,true);
    }
    //If Preview Button was pressed, dont publish via nc.call, only build NetStream!
    else changePreviewStream();

    metaCheck(parseInt(itemIndex));                     //Check new MetaData
}//END Function changestream
```

Abbildung V-24: changeStream()-Funktion. Hier wird der Inhalt des ausgewählten Streams geändert.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Die ersten beiden Zeilen der `changeStream()`-Funktion dienen der zuvor beschriebenen Teilstringgewinnung. Der nachfolgende Befehl
`var currentResource:String = tab1.dg_StreamList.selectedItem.Name`
liest den Dateinamen des aktuell ausgewählten Elementes in der Stream-Liste in eine Stringvariable ein. Wird über die `if`-Abfrage festgestellt, dass der zu ändernde Stream bereits mit Inhalten belegt ist, so stoppt der Datentransfer und alle Daten im Puffer werden verworfen. Der restliche Teil der `changeStream()`-Funktion gleicht bis auf die dynamische Zuweisung der Variablenindizes annähernd der `connectStream()`-Funktion. Wesentlicher Unterschied ist jedoch die konditionale Umklammerung, welche differenziert, ob die öffentlichen Streams oder der Vorschau Stream geändert werden sollen. Tritt der erste Fall in Kraft, so muss nach der bereits bekannten Streamerzeugung ein Aufruf der jeweiligen Server-side-Funktion `selectSource1` bis `selectSource4` erfolgen. Der Befehl
`nc.call(("selectSource"+itemIndex),null,currentResource,true)` ruft die entsprechende Methode in der `main.asc` auf und übergibt ihr anhand der Variablen `currentResource` den Namen der abzuspielenden Mediendatei. Es wird kein `Responder`-Objekt erwartet, daher erhält der zweite Parameter den Wert `null`. Die Funktion soll jedoch aktiv sein und den Stream aufrecht erhalten, der letzte Parameter `true` sichert dies.
Soll der Stream im Vorschaubereich geändert werden, so muss kein Zugriff auf eine Server-side-Funktion erfolgen, daher wird hier die gesonderte Methode `changePreviewStream()` aktiviert, die auf den `nc.call`-Befehl verzichtet, jedoch einige marginale Eigenheiten des Vorschau-Fensters berücksichtigt.

Mit diesen Methoden ist die grundlegende Funktionalität der Stream-Verarbeitung gesichert.

2.5.1.c Das NetStatus-Objekt

Die Statusmeldungen der beiden `NetConnection`-Objekte werden in den folgenden Funktionen verarbeitet. Die Meldungen des FMS folgen einem konsistenten Schema (im XML-Format), so dass der einzelne Statustyp direkt über einen String abgefragt werden kann. War

die Verbindung erfolgreich, wird „`NetConnection.Connect.Success`“ ausgegeben, was Anlass und Voraussetzung dafür ist, die Funktion `connectStream(nc)` aufzurufen und die Streams zum FMS aufzubauen.

```
/*--------------------------------
    Get NETCONNECTION status
--------------------------------*/
public function netStatusHandler(event:NetStatusEvent):void
{
    eventcode = event.info.code;
    info_container1.holder_msg_netstatus.mc_msg_netstatus.appendText(eventcode + "\n");
    bufferHandler(event);
    switch (eventcode) {
    case "NetConnection.Connect.Success":
        info_container1.mc_msg_ncstatus.msg_ncstatus.text=("\nconnected is: " + nc.connected + "\nevent.info.level: " +
        connectStream(nc);
        info_container1.mc_msg_ncurl.msg_ncurl.text = rtmpNew;
        info_container1.mc_btn_connecthandler.btn_connecthandler.label="Disconnect";
        break;
    case "NetConnection.Connect.Failed":
        info_co...
    case "NetConnection.Connect.Rejected":
        info_co...
    case "NetConnection.Connect.Closed":
        info_co...
    case "NetStream.Play.Stop":
        info_co...
    case "NetStream.Play.StreamNotFound":
        info_co...
    case "NetStream.Publish.BadName":
        info_co...
    }
}
```

Abbildung V-25: netStatusHandler()-Funktion. Registriert und visualisiert die Meldungen des NetStatusEvent-Ereignisses.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Treten Fehlermeldungen auf, werden diese gesondert behandelt und in der Statusanzeige visualisiert. Die über eine `netStatusHandler()`-Methode (s. Abbildung V-25) aufgerufene `bufferHandler()`-Funktion wird in Kapitel V Abschnitt 2.5.1.e näher betrachtet.

Analog zur Statusverarbeitung der `NetStreams` und des `NetConnection`-Objekts `nc` wird auch der Status der Verbindung zur Administration-API anhand der Variable `nc_admin` überwacht (s. Abbildung V-26). Konnte die Konnektivität erfolgreich hergestellt werden, sind die Voraussetzungen gegeben, um Informationen über aktuelle Live-Streams und deren Status einzuholen. Die boole'sche Variable `APIconnect` fixiert eine erfolgreiche Verbindung mit dem Wert `true`, der in der `getLiveStreams()`-Methode, weiteres hierzu in Kapitel V Abschnitt 2.5.2, konditional für die Abarbeitung der Funktionen genutzt wird.

```
public function ncadminStatusHandler(event:NetStatusEvent):void {
    if (event.info.code == "NetConnection.Connect.Success") {
        APIconnect = true;
        var liveListTimer:Timer = new Timer(liveListRefreshInt,0);
        liveListTimer.addEventListener("timer", liveListTimerHandler);
        liveListTimer.start();          //Start Timer for continous liveList refresh
        info_container1.mc_ncAPIStatus.lbl_ncAPIStatus.text = "NetConnection.Connect.Success"
    } else {APIconnect=false; info_container1.mc_ncAPIStatus.lbl_ncAPIStatus.text = event.info.code;}
}
```

Abbildung V-26: ncadminStatusHandler()-Funktion. Registiert und visualisiert die Meldungen des NetStatusEvent-Ereignisses für die Verbindung zur Administration-API.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

2.5.1.d Video-Objekt und Meta-Daten Handling

Voraussetzung für die professionelle Videoübertragung über das Internet sind Mediendateien mit zusätzlichen beschreibenden Inhalten, den Meta-Daten. Denn diese geben Aufschluss über wesentliche Informationen der Mediendatei wie die Bildgröße, die Framerate[G], die Datenrate, den Codec und Weiteres.

Somit ist es auch für diese Applikation von Bedeutung, dass die Videodateien Meta-Daten beinhalten. Für jeden Stream existiert eine eigene `metaCheck()`-Funktion, die die Zusatzinformationen sammelt (s. Abbildung V-27). Da für das Ereignis `onMetaData`, welches beim Empfang von beschreibenden Informationen innerhalb eines Streams eintritt, kein `EventListener` existiert, muss hilfsweise mit einer Zwischenvariablen gearbeitet werden. Dabei wird die `client`-Eigenschaft des Streams auf eine Variable des Typs `Object` referenziert. Über dieses Objekt können nun die Ereignisse `onMetaData` und `onPlayStatus` abgefangen werden. Über den Befehl `metaSniffer.onMetaData=getMeta1` werden folglich die empfangenen Meta-Daten an die Funktion `getMeta1()` übergeben, was gleichzeitig den Aufruf dieser Funktion impliziert. Selbiges geschieht mit dem `onPlayStatus`-Ereignis, das u.a. das Ende der Wiedergabe eines Streams signalisiert.

```
/*----------------------------------------
   CREATE METADATA Receive Objects
----------------------------------------*/
        private function metaCheck(streamnumber:int):void {
            switch (streamnumber)
            {
            case 1:     //Check MetaData of Stream 1
                info_container2.holderMeta1.msg_meta1.text = "Sorry. No Meta-Data available.";
                metaSniffer=new Object();
                ns1.client=metaSniffer;
                metaSniffer.onMetaData=getMeta1; metaSniffer.onPlayStatus=onPlay1;
                break;
```

Abbildung V-27: metaCheck()-Funktion. Erzeugt eine Objekt zur Übergabe der Meta-Daten an die getMeta()-Funktionen.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Verarbeitet werden die Meta-Daten in den `getMeta()`-Funktionen, die für jeden Stream gesondert existieren (s. Abbildung V-28). Als Parameter wird ein Objekt, hier `mdata1`, übergeben. Dieses trägt den Inhalt des Objektes, auf das zuvor `ns.client` referenziert wurde. In ihm sind alle verfügbaren Meta-Daten in dem bereits bekannten XML-Format gespeichert. Es könnten also über die XML-Knoten einzelne Meta-Daten direkt abgefragt werden. In dieser Anwendung werden jedoch alle enthaltenen Meta-Daten angezeigt, um den maximalen Informationsgrad zu erreichen.

```
/*----------------------------------------
  Get METADATA for Stream 1 - 4
----------------------------------------*/
        public function getMeta1 (mdata1:Object):void {
            if (mdata1 != null) info_container2.holderMeta1.msg_meta1.text = "";
            var aspect1:Number = mdata1.width / mdata1.height;
            scalevideo(1, aspect1);
            for (var i in mdata1) {
              info_container2.holderMeta1.msg_meta1.appendText(i + ":\t" + mdata1[i] + "\n" );
            }
        }
```

Abbildung V-28: getMeta1()-Funktion. Verarbeitet die Meta-Daten des Streams.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Die Schleife `for(var i in mdata1){…}` durchläuft alle Knoten in `mdata1`, deren Bezeichnung über `i` sowie der Inhalt über `mdata1[i]` angezeigt wird. (Zur Erläuterung: Der Knoten `<width>320</width>` würde in der Textbox als „`width: 320`" wiedergegeben.)
Dass die Meta-Daten von weit reichender Bedeutung sind, wird endgültig am Befehl `var aspect1:Number = mdata1.width / mdata1.height` sichtbar. In der globalen Variable `aspect1` wird das Bildseitenverhältnis des Videos anhand der Breite und Höhe bestimmt. Dieses fließt später in die Funktion `scalevideo()` (s. Abbildung A-3 im Anhang) zur Berechnung der Größe der Videoinstanzen auf dem Interface ein. Es ist also essenziell, dass dem Video Meta-Informationen mit der Breite und Höhe beigefügt sind.

Das `onPlayStatus`-Ereignis dient, wie oben angedeutet, der Detektion der Beendigung eines Streams (s. Abbildung V-29). Den Meta-Daten-Funktionen entsprechend existiert für jeden einzelnen Stream eine `onPlay()`-Funktion. Des Weiteren wird auf gleiche Weise ein Parameter des Typs `Object` transferiert, der das Status-Ereignis dokumentiert. Für diese Anwendung interessant ist jedoch lediglich der Meldungs-Code „`NetStream.Play.Complete`". Erscheint diese Meldung, so wird der Untertitel des Streams zurückgesetzt und die Wiedergabe eines Platzhalter-Videos eingeleitet.
Ähnlich dem Testbild beim Fernsehen kann als Stream ein Platzhalter-Video wiedergegeben werden, welches das Ende der Übertragung visualisiert. In diesem Fall ist dies eine Videodatei, die zu Beginn der Applikation der Variablen `SfinishedVideo` zugewiesen wurde, eine kurze Sequenz die den Betrachter über den finalen Status informiert.

```
/*---------------------------
    onPlayStatus Handler
---------------------------*/
private function onPlay1(onPlay:Object):void {
    info_container1.holder_msg_netstatus.mc_msg_netstatus.appendText("NETSTREAM 1: \n" + onPlay.code + "\n" );
    if (onPlay.code =="NetStream.Play.Complete") {
        initLblChannels(1);
        nc.call("selectSource1",null,SfinishedVideo,true);
        ns1.play("stream1");
    }//endif
}
```

Abbildung V-29: onPlay1()-Funktion. Verarbeitet die onPlayStatus-Informationen des Streams.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

2.5.1.e Duales Puffern

Für die Nutzung von Streams als Daten/Video-Übertragungsmethode ist es wichtig, die Daten zwischenzuspeichern. Dies wird mit dem Puffer realisiert, der einen Anteil der Daten in den lokalen Zwischenspeicher (RAM) lädt. Die Größe bzw. Länge dieses Puffers lässt sich über die Eigenschaft `bufferTime` der `NetStream`-Klasse einstellen. Es existieren drei Statusmeldungen des `NetStatusEvent`-Objektes, die den Puffer betreffen:

1. „`NetStream.Buffer.Full`"
2. „`NetStream.Buffer.Empty`"
3. „`NetStream.Buffer.Flush`"

```
//Handle the Buffer-size
private function bufferHandler(infoObject:NetStatusEvent):void {
   // Define onStatus event handler
    if ((infoObject.target != NetConnection) ) {
        switch (infoObject.info.code) {
        case "NetStream.Buffer.Full":
            flushdetected = false;
            infoObject.target.bufferTime = expandedBufferLength;
            break;
        case "NetStream.Buffer.Empty":
            if (flushdetected) {
                infoObject.target.bufferTime = startBufferLength;
                flushdetected = false;
            } else {
                infoObject.target.bufferTime = startBufferLength;
            }
            break;
        case "NetStream.Buffer.Flush":
            flushdetected = true;
            break;
        } //End Switch
    } //End if

} //End Function bufferHandler

private function bufferInitialize(callingstream:NetStream):void {
    callingstream.bufferTime = startBufferLength;
}
```

Abbildung V-30: bufferHandler()-Funktion. Dynamische Pufferdefinition.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Diese Ereignisse werden im `bufferHandler()` abgefangen (s. Abbildung V-30). Beim Start der Applikation wird jedem Stream zunächst ein kurzer Puffer von 2s über die `bufferInitialize()`-Funktion zugewiesen. So wird sichergestellt, dass der Puffer schnell gefüllt ist und die Wiedergabe zeitnah einsetzt. Ist der Puffer voll, meldet das `NetStatusEvent` „`NetStream.Buffer.Full`". Ist die Videowie-

dergabe bereits gestartet, kann der Puffer auf einen größeren Wert von bspw. 20s umgestellt werden. Dies garantiert, dass der Puffer groß genug für die meisten Videoübertragungen ist und nicht kontinuierlich unterbrochen und neu gepuffert werden muss. Sollte es dennoch auftreten, dass mehr Daten übertragen werden als im Puffer zwischengespeichert werden können, so kommt es zu einer Leerung und der Meldung „`NetStream.Buffer.Empty`". Die `if`-Kondition erkennt dies und setzt den Puffer wieder auf die anfängliche Länge von 2s, so dass der Puffer schnell gefüllt und die Videowiedergabe fortgesetzt werden kann. Die Unterbrechungszeiten werden somit möglichst kurz gehalten.
Endet der Datentransfer eines Streams tritt zunächst das Ereignis „`NetStream.Buffer.Flush`" ein, d.h. der Puffer läuft über und wird nicht neu gefüllt, hierauf folgt die Meldung „`NetStream.Buffer.Empty`". Der Puffer ist geleert und der Initialstatus von 2s wird gesetzt.

2.5.2 XML-Strukturen und Server-side-Funktionen zur Dateiverwaltung

Wie bereits in den Ausführungen zum Server-side-Skript deutlich wurde sind die XML-Dateien und die `FileObject`-Klasse von wesentlicher Bedeutung für die Verarbeitungen der Informationen dieser Applikation.
Um Auskunft über verfügbare Videodateien, Thumbnails sowie die temporären Streams zu erhalten, muss zunächst Server-seitig das jeweilige `FileObject` erzeugt werden.
Die `FileObject`-Methoden erfüllen diesen Zweck (s. Abbildung V-31). Dazu wird in jeder Funktion über den Befehl `nc.call()` die korrespondierende Methode im Server-side-Skript aufgerufen. Der `call`-Befehl ist eine Methode der `NetConnection`-Klasse, bezieht sich also stets auf die Verbindung zwischen Client und Server-side-Applikationsskript. Als Parameter werden der Funktionsname im `String` Format, ein sog. `Responder` und ggf. weitere individuelle Variablen übergeben. In diesem Fall der jeweilige Zielordner der XML-Datei. Der `Responder` dient der Verarbeitung von returnierten Werten der Server-side-Funktion und wird hier durch den Parameter `null` zunächst nicht aktiviert, ist jedoch im folgenden Code das Kernelement der `nc.call()`-Anweisung (s. Abbildung V-32).

```
//Create the FileObjects for the specified functions (videofiles, imagefiles, temporaryfiles)
private function createFileObj():void {
    tab1.dataGrid_Loader.visible = true;              //Show Loading MC during List Refresh
    nc.call("createFileObj", null, folderName);      //Call server-side function
}
private function createImgFileObj():void {
    tab3.dataGrid_Loader.visible = true;
    nc.call("createImgFileObj", null, imgfolderName);
}
private function createTempFileObj():void {
    tab4.mc_tab4.dataGrid_Loader.visible = true;
    nc.call("createTempFileObj", null, tempfolderName);
}
```

Abbildung V-31: Aufruf der FileObject-Funktionen.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Um die Ergebnisse der Server-side-Filter- und Auflistungsfunktionen zu erhalten, sind weitere `nc.call()`-Befehle nötig, denen diesmal allerdings ein `Responder` zugewiesen wird. Die Server-seitig aufgerufen Funktionen schließen alle mit einer `return`-Anweisung ab, die ein Array zurückgibt. Die Verarbeitung des `return`-Wertes geschieht mit dem `Responder`. Hierzu wird in

```
//get the results of the server-side FileObject Listings
    private function dir():void {
        //Call server to obtain directory list and give values to dirResult
        nc.call("dir", new Responder(dirResults, dirFaults));
        nc.call("imgDir", new Responder(imgDirResults, imgDirFaults));
    }
    private function tempDir():void {
        nc.call("tempDir", new Responder(tempDirResults, tempDirFaults));
    }
```

Abbildung V-32: Funktionen zur Verarbeitung der Rückgabewerte der FileObject-Listen.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

der `nc.call()`-Anweisung, nach dem Namen der Funktion als zweites Argument ein `Responder` erzeugt, der zwei Parameter inkludiert. Zum einen die beschriebenen Rückgabewerte und zum anderen eventuell aufkommende Fehler bei der Rückgabe. Für beide Zustände wird jeweils eine eigene Funktion herangezogen und dies individuell für jedes `FileObject`, also insgesamt drei Funktionen für die Ergebnisverarbeitung sowie drei Funktionen für die Fehlermeldungen.

Exemplarisch und aus Gründen der Übersichtlichkeit wird nachfolgend die kompakteste der drei Ergebnisverwaltungsmethoden behandelt. Abbildungen der beiden anderen Funktionen befinden sich im Anhang (s. Abbildung A-3 u. Abbildung A-4). Bei dieser Funktion handelt es sich um die Verarbeitung der Ergebnisse des Thumbnail-`FileObject`.

```
//////////////////////////////////////////
// Get List of Imagefiles in thumb Folder
//////////////////////////////////////////
        function imgDirResults(retVal) {
            //
            tab3.list_ThumbList.setStyle("cellRenderer", rendererCCThumb);  //Set CellRenderer For Thumblist

            if (imgArray.length > 0) imgArray.length = 0;           //initialize Array
            for (var i = 0; i<retVal.length; i++) {
                //takes the "/approot/thumbs/" out of the retVal to resolve the pure filename
                var img_name = (retVal[i].name).substr((retVal[i].name).lastIndexOf("/")+1);
                //Sets the array to your FLV
                //Adds each stream information to imgArray which is then put into TileList
                imgArray.push({name:img_name, source:appRootAbs+thumbFol+img_name, lastmod:retVal[i].lastModified});
            }
            //Add results to Images TileLis
            addColumnImgList();
        }
        function imgDirFaults(errorVal) {
            trace("Error : "+imgfolderName+" "+errorVal.code+"\n");
        }
```

Abbildung V-33: imgDirResuts()-Funktion. Sortiert die Rückgabewerte der FileObject-Liste in ein Array.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Wie ersichtlich ist, wird der Funktion `imgDirResults(retVal)` der Parameter `retVal` übergeben. Dieser beinhaltet den Wert der Server-side `return`-Anweisung, welcher über den `Responder` an diese Funktion übergeben wurde. Da es sich wissentlich um eine Liste von Bilddateien handelt, wird ein Array, hier mit Namen `imgArray`, zur Speicherung der Daten verwendet. Dieses wird zunächst auf die Länge `0` gesetzt, da bei jedem erneuten Aufruf das Array neu aufgebaut werden soll. Ein einfaches `removeAll()` des Arrays funktioniert an dieser Stelle nicht. Es würden zwar die alten Einträge gelöscht, jedoch bliebe die Anzahl der Positionen im Array erhalten, es könnte also unter Umständen passieren, dass das Array bei erneuter Wertezuweisung mehr Positionen als eigentliche Werte besitzt. Dies hätte wiederum bei der Weiterverarbeitung des Arrays Fehler zur Folge. Mit der Reduktion der Länge auf den Wert `0` ist man hier also auf der sicheren Seite.
Die anschließende `for`-Schleife arbeitet sukzessive jeden einzelnen Rückgabewert ab. Im Verlauf der Schleife wird anfänglich der reine Name der Bilddatei aus dem kompletten Dateinamen, welcher auch den Pfad inkl. `Virtual Directory` beinhaltet, zurück gewonnen. Nachfolgend werden dem Array über den Befehl `push` die Elemente zugewiesen. Es entsteht hierbei ein mehrdimensionales Array, welches neben Zeilen auch Spalten besitzt. Die Spalten enthalten in diesem Fall die statischen Werte `name`, `source` und `lastmod`. Diesen Werten werden relativ der pfadlose Dateiname, der absolute Pfad sowie Informationen zum Erstellungszeitpunkt der Datei zugeordnet.

Wurden alle Elemente in `retVal` abgearbeitet und das Array `imgArray` mit selbigen gefüllt endet die Schleife und die Elemente des Arrays können über die Funktion `addColumnsImgList()` der entsprechenden `List-` bzw. `DataGrid`-Instanz des Interfaces hinzugefügt werden.
Nach dem gleichen Prinzip verarbeiten die `dirResults()`- und die `tempDirResults()`-Funktion die Server-side-Rückgabewerte. Die Informationen sind hier allerdings detaillierter und müssen auf andere Weise gefiltert werden, was in aufwendigeren Code-Strukturen resultiert, daher befinden sich die Screenshots dieser Funktionen im Anhang (Abbildung A-3 und Abbildung A-4).

```
//Move finished TempFiles to Streamfolder
private function moveTempFiles(e:MouseEvent):void {
    tab4.mc_tab4.dialog_MoveTemp.visible = false;
    if (tab4.mc_tab4.dg_TempFilesListFine.length > 0) {
        var forTransferArr:Array = new Array();
        var count:int = 0;
        for (var i:int = 0; i < tab4.mc_tab4.dg_TempFilesListFine.length ;i++){
            var actTempItem = tab4.mc_tab4.dg_TempFilesListFine.getItemAt(i);
            //(Responder,fileToCopy,targetFolder,sourceFolder)
            nc.call("moveTempFile",null,actTempItem.name,folderName,tempfolderName+"/");
            count++;
        }//endfor
        createFileObj();
        createImgFileObj();
        createTempFileObj();
        dir();
        tempDir();
        tab4.mc_tab4.lbl_TempFilesStatus.text = count + " Items moved to: " + instanceName;
    }//endif
}
```

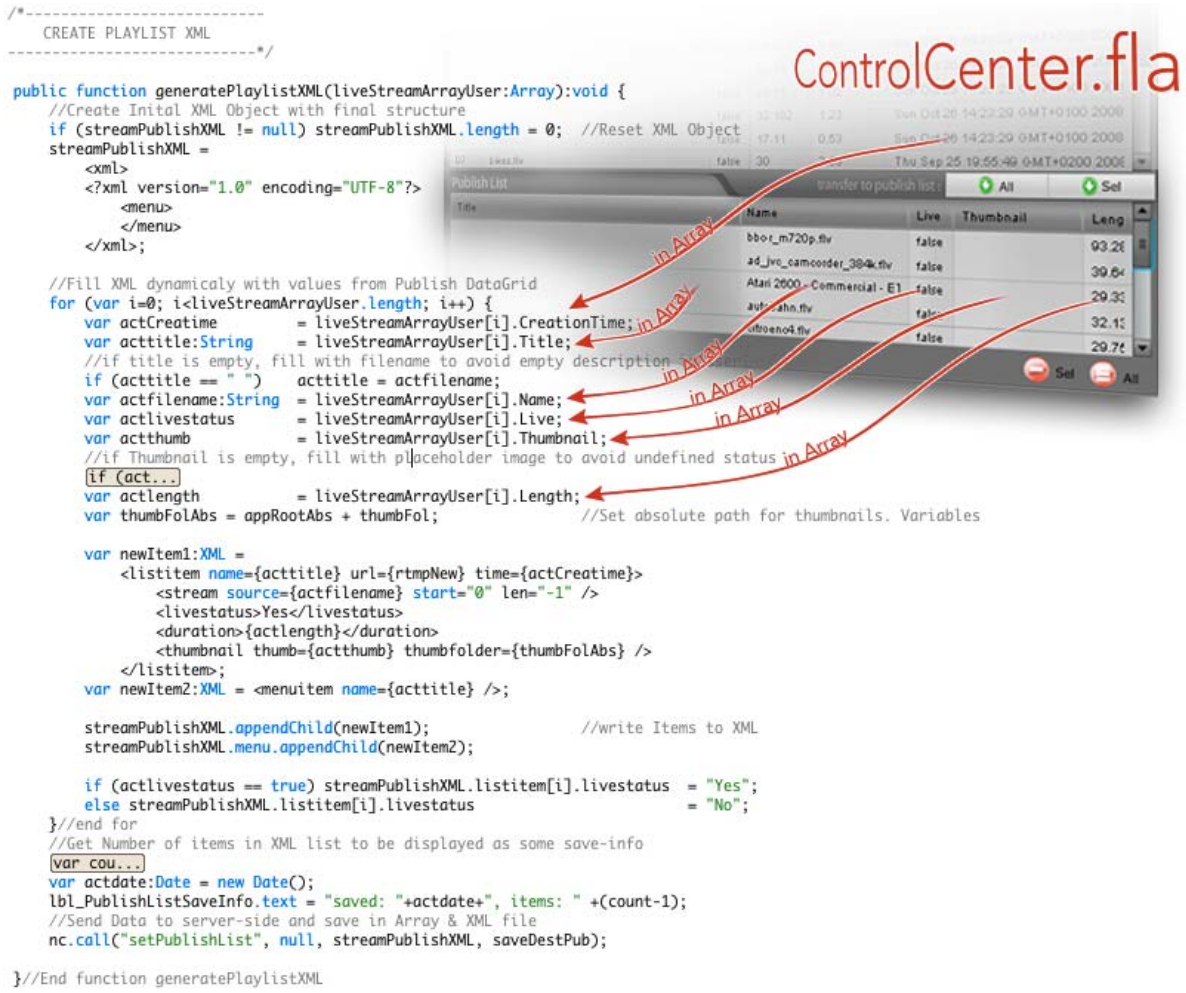
Abbildung V-34: moveTempFiles()-Funktion. Leitet die Verschiebung der gespeicherten temporären Streams in den regulären Streamordner ein.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Eine erwähnenswerte Eigenheit der Funktion `tempDirResults()` ist der zusätzliche Funktionsaufruf `moveTempFiles()` (s. Abbildung V-34). Ausgelöst durch die Betätigung einer Schaltfläche auf dem Interface wird zunächst überprüft, ob in der Liste der abgeschlossenen Streams Elemente vorhanden sind, die verschoben werden können. Bestätigt sich diese Bedingung beginnt eine `for`-Schleife, die alle Elemente der Liste einzeln abarbeitet und diese einer Zwischenvariablen `actTempItem` zuweist. Anschließend wird über `nc.call(„moveTempFile“,null,actTempItem.name,folderName,tempfolderName+“/“)` die Server-side-Funktion `moveTempFile`, welche dem Namen gemäß das angegebene Element, `actTempItem.name`, in den allgemeinen Streamordner verschiebt. Details zum Server-side-Skript finden sich in Kapitel V Abschnitt 2.4.2.
Zuletzt müssen nach dem Transfer alle Stream- resp. Dateilisten neu geladen werden, um die veränderte Dateistruktur zu visualisieren. Dies realisieren die Funktionsaufrufe am Ende der `moveTempFiles()`-Methode.

Letztes herauszustellendes Element der `ControlCenter.as` Codestruktur ist die Speicherung der XML-Daten. Dies soll repräsentativ an der `generatePlaylistXML()`-Funktion verdeutlicht werden (s. Abbildung V-35).
Anfänglich wird die Grundstruktur der XML-Datei über die globale Variable `streamublishXML` festgelegt. Flash sieht für die Verarbeitung von XML-Dateien eine eigene Klasse des Typs `XML` vor, welche bei den XML-Methoden dieser Applikation Anwendung findet. Durch Betätigung der Schaltfläche „`Save & Publish`“ unterhalb der Anzeige werden die Daten aus dem `DataGrid` der `PublishList` in dem globalen mehrdimensionalen Array `liveStreamArrayUser` zwischengespeichert und die einzelnen Parameter anschließend individuellen Variablen zugewiesen. Dies geschieht in der `generatePlaylistXML()`-Funktion im Rahmen einer `for`-Schleife jeweils für jedes Element des Arrays. Nachfolgend werden zwei

temporäre XML-Objekte, `newItem1` und `newItem2` erzeugt und mit den Inhalten der Variablen gefüllt. Diese zwei XML-Knoten werden mit dem `appendChild()`-Befehl der finalen `streamPublishXML` hinzugefügt. Ist die `for`-Schleife mit allen Elementen durchlaufen und die XML-Datei gefüllt kann sie über `nc.call(„setPublishList", null, streamPublishXML, saveDestPub)` an das Server-side Array übergeben werden. Abbildung V-35 verbildlicht diesen Prozess.




```
/*---------------------------
    CREATE PLAYLIST XML
---------------------------*/

public function generatePlaylistXML(liveStreamArrayUser:Array):void {
    //Create Inital XML Object with final structure
    if (streamPublishXML != null) streamPublishXML.length = 0;  //Reset XML Object
    streamPublishXML =
        <xml>
        <?xml version="1.0" encoding="UTF-8"?>
            <menu>
            </menu>
        </xml>;

    //Fill XML dynamicaly with values from Publish DataGrid
    for (var i=0; i<liveStreamArrayUser.length; i++) {
        var actCreatime         = liveStreamArrayUser[i].CreationTime;
        var acttitle:String     = liveStreamArrayUser[i].Title;
        //if title is empty, fill with filename to avoid empty description
        if (acttitle == " ")    acttitle = actfilename;
        var actfilename:String  = liveStreamArrayUser[i].Name;
        var actlivestatus       = liveStreamArrayUser[i].Live;
        var actthumb            = liveStreamArrayUser[i].Thumbnail;
        //if Thumbnail is empty, fill with placeholder image to avoid undefined status
        if (act...
        var actlength           = liveStreamArrayUser[i].Length;
        var thumbFolAbs = appRootAbs + thumbFol;                //Set absolute path for thumbnails. Variables

        var newItem1:XML =
            <listitem name={acttitle} url={rtmpNew} time={actCreatime}>
                <stream source={actfilename} start="0" len="-1" />
                <livestatus>Yes</livestatus>
                <duration>{actlength}</duration>
                <thumbnail thumb={actthumb} thumbfolder={thumbFolAbs} />
            </listitem>;
        var newItem2:XML = <menuitem name={acttitle} />;

        streamPublishXML.appendChild(newItem1);                 //write Items to XML
        streamPublishXML.menu.appendChild(newItem2);

        if (actlivestatus == true) streamPublishXML.listitem[i].livestatus  = "Yes";
        else streamPublishXML.listitem[i].livestatus                        = "No";
    }//end for
    //Get Number of items in XML list to be displayed as some save-info
    var cou...
    var actdate:Date = new Date();
    lbl_PublishListSaveInfo.text = "saved: "+actdate+", items: " +(count-1);
    //Send Data to server-side and save in Array & XML file
    nc.call("setPublishList", null, streamPublishXML, saveDestPub);

}//End function generatePlaylistXML
```

Abbildung V-35: generatePlaylistXML()-Funktion. Erzeugt die XML-Struktur der PublishList und überträgt diese an die main.asc.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Dieses Prinzip ist identisch für alle anderen Funktionen zur XML-Verarbeitung. Das Auslesen der XML-Daten wird im Rahmen des Kaptitels V Abschnitt 2.6.2 näher beschrieben. Deshalb wird dies hier nicht weiter vertieft.

Die Kernfunktionen des Back-Ends wurden hiermit im Detail betrachtet. Weitere Code-Fragmente finden sich - wie aus den Textverweisen hervorgehend - im Anhang.

2.5.3 Visuelle und administrative Bereiche der Benutzeroberfläche

Das programmiertechnische Konstrukt hinter dem Back-End Modul wurde im vorigen Kapitel in den Grundzügen erklärt, im Weiteren folgt nun der Wechsel zur visuellen Umsetzung der

Funktionen im grafischen Interface. Da die verfügbaren Elemente des GUI bereits in Kapitel V Abschnitt 1.3 präsentiert wurden, sind im folgenden Unterkapitel nur die funktionsbezogenen Ausschnitte näher ausgeführt.

2.5.3.a Vier öffentliche Streaming-Kanäle

Eine Eigenheit der vier öffentlichen Streaming Kanäle ist die Unabhängigkeit vom gestreamten Inhalt. Die vier Streams sind fortwährend mit der Applikation sichtbar und stellen alle Medieninhalte dar, die Ihnen der Administrator zuweist. Der End-User hat hierauf keinen Einfluß, es handelt sich also um regiegesteuerte Streams.
Wählt der Administrator einen Stream (Live oder VoD) aus der Streamliste aus und klickt anschließend auf die Schaltfläche „`Publish`" wird die zum Kanal gehörige `nc.call()`-Funktion ausgeführt und die ausgewählte Datei resp. Stream unter dem Namen `stream1` (1-4) über das Server-side-Skript veröffentlicht. Die „`Stop`" Schaltfläche beendet die Übertragung des aktuellen Streams und aktiviert das Platzhaltervideo, welches dem End-User das Ende des Streams signalisiert

Abbildung V-36: Screenshot eines Videofensters der öffentlichen Kanäle.

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

In der linken Textbox werden die Zusatzinformationen angezeigt, die Aufschluss darüber geben, auf welche Zeit der Puffer (`BufferTime`) des Streams gesetzt ist und wie die Pufferauslastung (`BufferLength`) der aktuellen Übertragung ist (s. Abbildung V-36). Die `Playtime` visualisiert die vergangene Zeit seit der Veröffentlichung des Streams. Der Darstellung weiterer Informationen steht hierbei nichts im Wege. Es ist jedoch zu beachten, dass es sich bei diesen Informationen um die Verbindungsdaten zwischen Back-End und FMS handelt. Der Wert der Pufferlänge hat also keinen Einfluss auf die Länge des Puffers im Front-End. Die Darstellung dient also eher der Einschätzung der Puffergrößen im Verhältnis zur Datenübertragung.

Als Optimierungsansatz wäre hier eine interaktive Anpassung des „Longtherm-Buffers", also dem Langzeit-Puffer, von Seiten des Administrators denkbar, die auch für das Front-End übernommen wird. Zudem könnte die Visualisierung um die aktuelle Datenübertragungsrate erweitert werden.

2.5.3.b Verwaltung, Visualisierung und Publizierung der Streams respektive A/V Daten

Der in Kapitel V Abschnitt 1.3.1 vorgestellte Verwaltungsbereich in der `ControlCenter.fla` stellt den Dreh- und Angelpunkt der Applikation dar. Hauptsächlich kommen hier die `DataGrid`- und `List`-Klasse von Flash zum Einsatz, die alle Inhalte möglichst effizient darstellen sollen. Die `DataGrid`-Klasse unterscheidet sich von der `List`-Klasse u.a. in dem Punkt, dass in ihr mehrdimensionale Daten bereits ohne zusätzlichen `CellRenderer` dargestellt werden können. Eine Liste ist hier auf eine sog. `CellRenderer`-Funktion angewiesen, die defi-

niert wie mehrspaltige Inhalte in der Liste dargestellt werden. Die `CellRenderer` kommen besonders in der Thumbnails- und in den Back-End-Listen zum Einsatz.

Nachfolgend werden die Funktionen der Registerkarten im Detail erläutert (s. Abbildung V-37).
Beginnend mit dem ersten Register, „`Available Streams`", kann man auch gleichzeitig vom wichtigsten Reiter sprechen. Hier werden alle verfügbaren Streams aufgelistet und nach ihrem Live-Status klassifiziert, `true` oder `false`.
Weitere Merkmale sind die Dauer, die Dateigröße und der Erstellungszeitpunkt des Streams

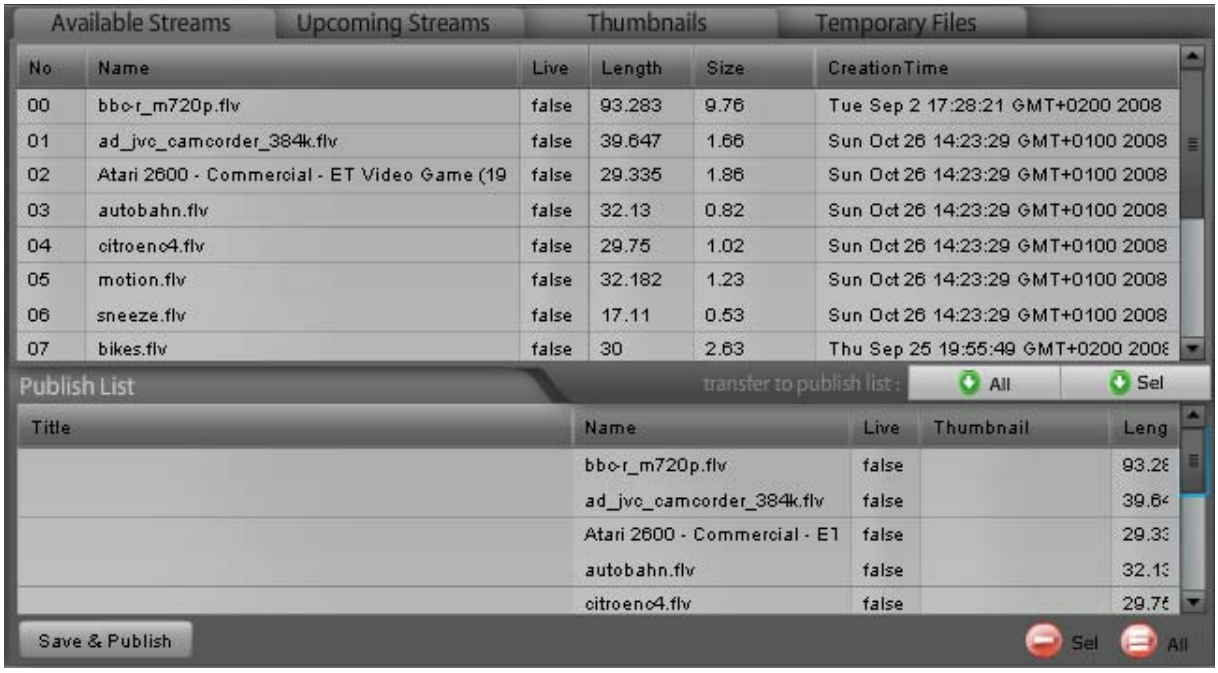


Abbildung V-37: Screenshot des Elements „Available Streams" des Verwaltungsbereiches der Streams.

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

bzw. der Datei. Die beiden erstgenannten Eigenschaften sind nur für die gespeicherten Dateien verfügbar, Live-Ereignisse erhalten hier den Wert `0`. Die Liste der Streams wird direkt aus dem `FileObject`-Array (s. Abbildung A-3) über die Funktion `addColumnStreamList()` dem `DataGrid` zugewiesen. Des Weiteren werden die aktuellen Live-Streams innerhalb der `getLiveStreams()`-Funktion (s. Abbildung A-5) über den an die Administration-API gerichteten Befehl `nc.call(„getLiveStreams")` aufgelistet. Die Namensgleichheit beider Funktionen ist hier von zufälliger Natur.
Über die mittleren grün gekennzeichneten Schaltflächen „`All`" und „`Sel`" können alle oder ein selektiertes Element der oberen Liste in die `PublishList` übertragen werden. Über die rot markierten Schaltflächen „`Sel`" und „`All`" können entweder das markierte oder alle Elemente wieder aus der `PublishList` gelöscht werden.

Die `PublishList` ist editierbar, so dass für jeden Stream ein individueller Titel eingegeben werden kann, der sich vom Dateinamen unterscheidet und einen besseren Wiedererkennungs- und Erläuterungsgrad bietet. Der Button[G] „`Save & Publish`" speichert die Elemente der `PublishList` in eine XML-Variable und überträgt diese an die `main.asc`, die wiederum über eine weitere Funktion die XML-Daten dem Front-End zur Verfügung stellt.

Zukünftige Ereignisse müssen in der Publizierung an das Front-End ebenfalls Berücksichtigung finden.
So bietet der Reiter „`Upcoming Streams`" die Option Ereignisse zu erstellen, die den Enduser

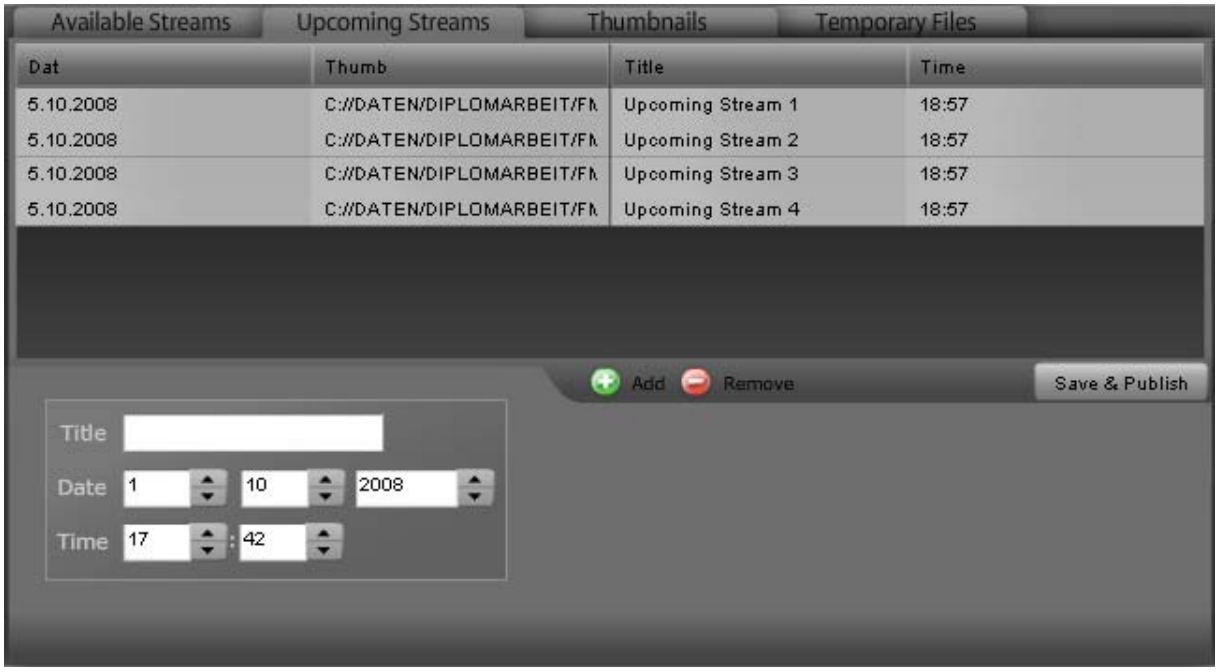


Abbildung V-38: Screenshot des Elements „Upcoming Streams" des Verwaltungsbereiches der Streams.

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

über künftige Streams informieren (s. Abbildung V-38). Dargestellt werden der Titel, das Datum, die Uhrzeit und ein Thumbnail des Events. Das Thumbnail entspricht einer Platzhalter-Grafik die symbolisiert, dass es sich um keinen momentanen Stream handelt, sondern um eine Vorankündigung.
Analog zum vorherigen Menüpunkt findet sich auch hier eine Button „`Save & Publish`", der den Schreibvorgang der Daten zum FMS, in diesem Fall in eine extern gespeicherte XML-Datei, die `upcominglist.xml`, einleitet. Somit soll die Konsistenz der Klickabfolge zur Publizierung der Listen in der Applikation gewahrt bleiben.
Eine Eingabemaske mit Text- und Auswahlfeldern, die auf das jeweilige Format adaptiert sind, erleichtert die Erzeugung eines neuen Eintrags und soll Eingabefehler bereits im Vorfeld unterbinden.

Im Reiter „`Thumbnails`" befindet sich eine Galerie der Bilddateien, die im Ordner `thumb` innerhalb des Applikationsverzeichnisses gespeichert sind. Wie aus Abbildung V-39 hervorgeht, werden die Grafik und der Dateiname dargestellt. Die aus dem Reiter „`Available Streams`" bekannte `PublishList` ist auch hier sichtbar. Wird ein Element aus dieser Liste selektiert und nachfolgend ein Feld der Thumbnails angewählt, wird dem Stream automatisch dieses Thumbnailbild zugeordnet. Der Pfad zur Thumbnaildatei wird nach erneutem Klick auf „`Save & Publish`" in die XML-Struktur integriert. Ist manuell kein Miniaturbild zugewiesen, tritt ein Automatismus ein, der dem entsprechenden Element der `PublishList` eine Patzhaltergrafik zuordnet.

Abbildung V-39: Screenshot des Elements „Thumbnails“ des Verwaltungsbereiches der Streams.

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

Der nachfolgende Screenshot (Abbildung V-40) zeigt den letzten Menüpunkt des Kernmoduls, die Verwaltung der temporären Streams, die „`Temporary Files`“.

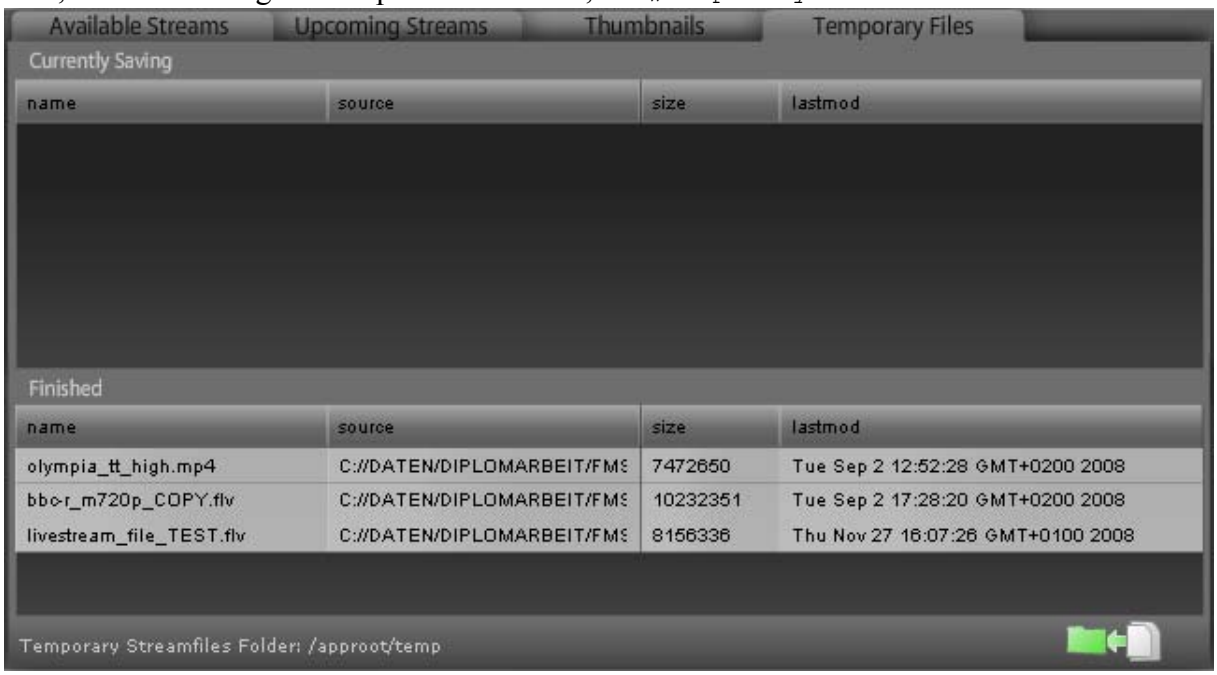


Abbildung V-40: Screenshot des Elements „Temporary Files“ des Verwaltungsbereiches der Streams.

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

Dieser Bereich ist in zwei Felder separiert. Die Darstellung der Inhalte im oberen Feld bezieht sich auf jene Dateien, die zur Laufzeit in den Applikationsunterordner `temp` geschrieben werden. Mit dem FME ist es, wie eingangs erläutert, möglich, sowohl live zu übertragen als auch zeitgleich aufzuzeichnen. Der Aufzeichnungspfad und der Dateiname können über den FME

definiert werden. Für diese Applikation muss ein fester Speicherpfad vorliegen, der in den o.g. Unterordner `temp` verweist.
Da die Dateien in diesem Ordner solange geöffnet sind und beschrieben werden wie der FME Prozess läuft, ist ein Detektionsverfahren nötig, welches abgeschlossene Dateien von solchen, die derzeit editiert werden, unterscheidet. Hierzu wird eine Vergleichsfunktion ausgeführt, die den Status der Dateigröße überwacht. Eine statische Dateigröße lässt auf eine abgeschlossene Datei schließen. Wird dieser Zustand erkannt tritt ein Verschiebungsprozess ein, der die Datei aus der Liste „`currently Saving`" in die Liste „`Finished`" transferiert. Der Administrator hat nun die Möglichkeit über die Symbolschaltfläche im rechten unteren Bereich einen weiteren Verschiebungsprozess einzuleiten, der die finalisierten Dateien in den allgemeinen übergeordneten Streamordner überträgt und somit als Standard VoD Stream lesbar macht.

Dieses Verfahren ist für die in das Front-End integrierte Wiederholungsfunktion essentiell. Anmerkend sei jedoch gesagt, dass, wie aus dem oben erläuterten Prozess deutlich wird, ein Zugriff auf die gespeicherte Version eines Live-Streams nur nach Beendigung des Streaming-Vorgangs im FME möglich ist. Die wiederholte Ansicht des Live-Streams über die parallel gespeicherte VoD Datei ist also erst nach einem kurzen Zwischenstopp des Streamings realisierbar. Hierzu könnte im sportlichen Kontext z.B. eine Unterbrechung (Halbzeit) dienen.

Der Vorschaubereich (s. Abbildung V-5) ist mit dem oberen `DataGrid` aus dem Register „`Available Streams`" verbunden. Die Vorschau bezieht sich stets auf das Videoelement, welches in der Streamliste selektiert ist. Aus diesem Fenster lässt sich über den oberen Schieberegler im Sekunden-Intervall durch das Video scrollen und somit ein Ereignis inkl. zugehörigem Zeitpunkt heraussuchen. Die Funktion `ns.seek()` bietet die Basis für das Suchen eines Zeitpunktes innerhalb eines Streams. Dies ist aus naheliegenden Gründen natürlich nur für eine gespeicherte Videodatei einsetzbar.

Bereits beim Durchklicken der Streamliste wird ein Initialbild (entspricht einem Frame des Videos) des äquivalenten Streams samt Meta-Daten über `ns.play()` geladen. Dies kommt schlussendlich der Schnellbeurteilung der Inhalte zugute.
Obligatorisch sind die Start-/Stopp-Schaltflächen sowie ein Lautstärkeregler und eine Stummschaltung. In Abbildung V-5 ist das Audioicon rechts oben sichtbar, welches den aktiven Soundkanal signalisiert. Die Textbox stellt analog zu Kapitel V Abschnitt 2.5.3.d alle Meta-Daten des zugehörigen Streams dar.

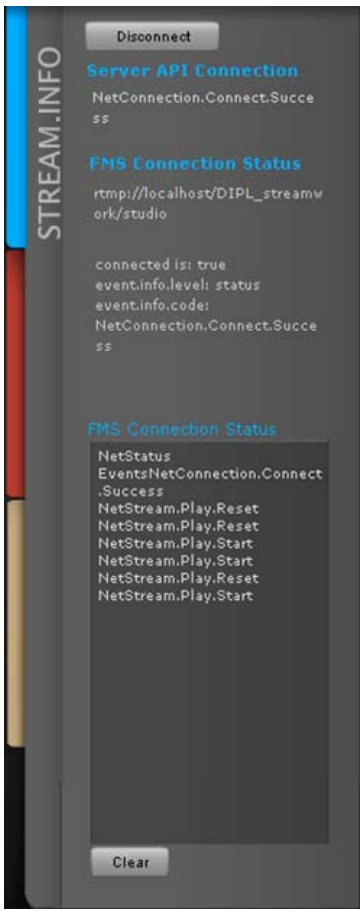


Abbildung V-41: Screenshot der Seitenregister. Hier: „Stream Informationen".

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

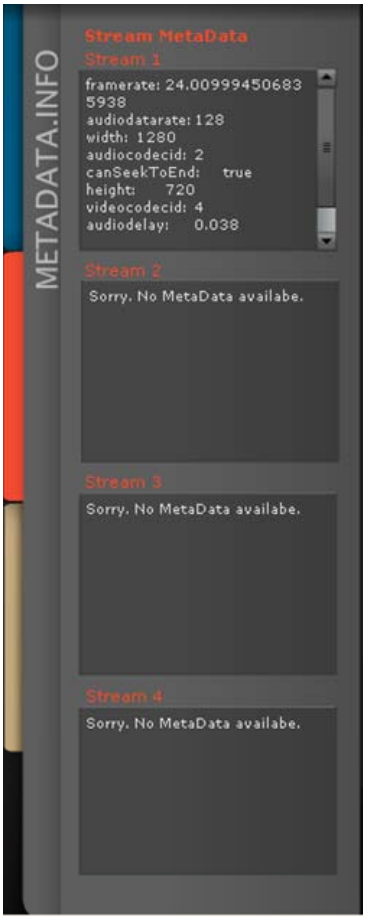


Abbildung V-42: Screenshot der Seitenregister. Hier: „Meta-Daten".

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

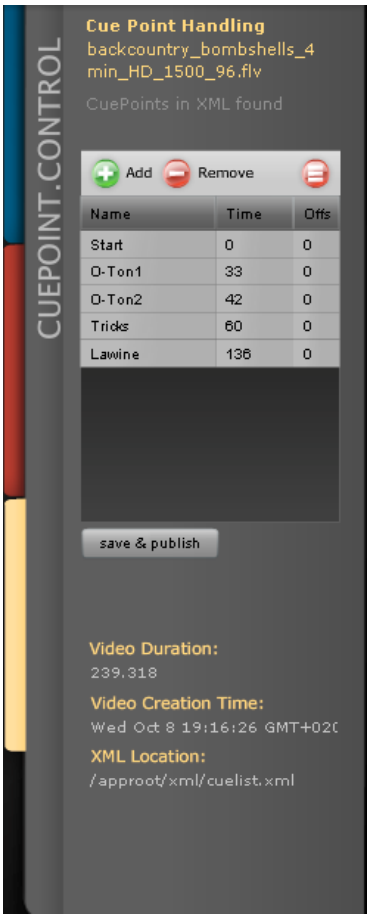


Abbildung V-43: Screenshot der Seitenregister. Hier: „CuePoints".

[Quelle: ControlCenter.fla, Stand: 10.12.2008]

2.5.3.c Status Kontrolle

Der rechte Informationsbereich des Back-Ends ist in drei Registerkarten differenziert, deren Unterscheidung durch unterschiedliche Farbschemen erleichtert werden soll. Die Statuskontrolle, das Register „`Stream.Info`" (s. Abbildung V-41), präsentiert wesentliche Informationen des `NET_STATUS`-Events der `NetConnection` zum FMS und zur Administration-API. Schlägt eine Verbindung Fehl oder wurde sie getrennt, zeigt das entsprechende Textfeld dies an. Über die obige Schaltfläche kann die Verbindung zum FMS aufgebaut oder getrennt werden. Bei Applikationsstart wird die Verbindung automatisch eingeleitet. Im „`FMS Connection Status`" Fenster werden alle Ereignisse der `nc` und `ns1` bis `ns5`-Objekte protokolliert.

2.5.3.d MetaDaten Darstellung

Die Meta-Daten der vier öffentlichen Streams (`stream1 - stream4` resp. `ns1 - ns4`) werden im zweiten Seitenregister (s. Abbildung V-42) angezeigt. Die Meta-Daten des fünften Streams, `ns5`, befinden sich in einer gesonderten Textbox innerhalb des Vorschaubereiches. Jedem Stream steht für die Meta-Daten eine scrollbare Textbox zur Verfügung. Die Funktionen `getMeta1()` bis `getMeta4()`, bekannt aus Kapitel V Abschnitt 2.5.1.d, extrahieren alle

vorhandenen Meta-Informationen des Streams und schreiben diese in die zugehörige Textbox. So sind jederzeit alle beschreibenden Informationen sichtbar und geben Aufschluss über die Beschaffenheit und Qualität der übertragenen Mediendateien.

2.5.3.e CuePoint Handling

Dem CuePoint Handling muss eine gesonderte Bedeutung beigemessen werden, da die CuePoint-Informationen auch im Front-End extrahiert und weiterverarbeitet werden. Daher bietet der dritte und letzte Seitenregister (s. Abbildung V-43) auch die Option die CuePoints gezielt zu editieren. Die Anzeige der CuePoints steht in direkter Beziehung zur oberen Streamliste der Verwaltungseinheit „`Available Streams`". Wird ein Stream der Kategorie VoD ausgewählt, so werden die zugehörigen CuePoints, sofern vorhanden, geladen. Die CuePoint-Verarbeitung ist für Live-Streams nicht aktivierbar. Der Name der aktuell ausgewählten Videodatei wird im oberen Bereich zusätzlich angezeigt. Es erfolgt ebenso eine Übertragung der Dauer des Videos und des Erstellungszeitpunkts aus der Streamliste. Beide Informationen werden in Textfeldern angezeigt. Der Speicherort inkl. Dateiname referenziert auf die im Konstruktor definierte Variable und zeigt wo die CuePoints letztendlich gespeichert werden.

Das `DataGrid`-Element listet die CuePoints auf und splittet die Darstellung in die Spalten `Name`, `Time` und `Offset`. Die `DataGrid`-Instanz ist editierbar, sodass jedem CuePoint ein individueller Titel, ein Zeitpunkt (in Sekunden) und ein Versatz, welcher die Zeit um einen bestimmten Wert vor oder zurück verschiebt, zugewiesen werden kann.
Die oberen Schaltflächen bieten die Möglichkeit einen CuePoint hinzuzufügen, den ausgewählten CuePoint zu entfernen oder alle CuePoints der ausgewählten Datei und den zugehörigen Eintrag in der `cuelist.xml` zu löschen.
Die bereits bekannte Schaltfläche des Typs „`Save & Publish`" speichert die CuePoints in die externe XML-Datei (s. Abbildung A-8). Aus Gründen des Skriptumfangs der Funktion muss an dieser Stelle auf einen Screenshot verzichtet werden.

Grundsätzlich gibt es ein gegenüber gängigen CuePoint Systemen wichtiges Unterscheidungsmerkmal zu der CuePoint Verarbeitung in dieser Applikation. Es existieren allgemein zwei Optionen einem Video CuePoints zuzuordnen: Zum einen werden die CuePoints direkt beim Enkodieren, also auf Seiten des „Videolieferanten", in das Video integriert. Der Vorteil hierbei ist, dass die CuePoints jederzeit mit dem Video verfügbar und fest in die Datei verankert sind. Nachteil ist, dass nur der Herausgeber des Videos CuePoints hinzufügen kann und alle CuePoints erst nach vollständigem Laden des Videos abrufbar sind. Die Variabilität dieser Variante ist also sehr gering. Sie eignet sich eher für Anwendungen mit progressive Download-System.

Eine zweite Möglichkeit bieten die Actionscript CuePoints, die in Verbindung mit der in Flash integrierten `FLVPlayback`-Komponente stehen. Wird ein Video mit dieser Komponente wiedergegeben, lassen sich zur Laufzeit per Actionscript CuePoints zuweisen die nicht in das Video selbst eingebettet sind, sondern als Zusatzinformation gespeichert werden. Der Nachteil bei diesem Verfahren ist jedoch die exklusive Verarbeitung mit der `FLVPlayback`-Komponente. Die *normale* `Video`-Instanz in Flash kann auf diese CuePoints nicht ohne weite-

res zugreifen. Da in dieser Anwendung Instanzen des Typs `Video` und nicht `FLVPlayback` eingesetzt werden, kommen also beide Varianten nicht in Frage.

Einen individuellen Lösungsansatz bot hier das XML-Format. Die CuePoints werden dem Video über den Dateinamen zugeordnet. Sie können beliebige Informationsstrukturen enthalten und sind jederzeit mit XML-kompatiblen Systemen abrufbar. Diese Variabilität erwies sich für den Aufbau der Applikation als nützlich.
Es ist zudem denkbar, dass die CuePoints nicht nur vom Back-End geändert und bearbeitet werden können, sondern auch vom Standort der Videoproduzenten. Ein dezentrales System kann die Effektivität deutlich erhöhen, da die manuelle Arbeit der CuePoint-Editierung auf mehrere Positionen verteilt würde.

2.5.4 Grafische Aufbereitung der Informationen

Wie aus den vorherigen Abschnitten ersichtlich wird, kommen im Back-End wiederkehrend bestimmte Komponenten und Arten der Darstellung zum Einsatz. Dies ist ebenso übertragbar auf das Front-End.
Für die Auflistung der Streams und aller anderen Server-Side-Inhalte werden Komponenten des Typs `DataGrid` und `TileList` eingesetzt. Die `DataGrid`-Instanzen stellen die Listen der Streams und Dateien sowie der CuePoints dar, da sie für die Wiedergabe umfassender Daten verschiedenen Typs angelegt wurden. Die `TileList`-Instanz erweist sich für die Thumbnail Visualisierung als nützlich, da die `TileList`-Komponente von Grund auf für diese Art der Darstellung von Flash vorgesehen wurde. Über den `CellRenderer` (s. Abbildung V-44 u. Abbildung A-6) können die Bilder direkt als solche in der Liste dargestellt und einem nebenstehenden Text zugeordnet werden. Alle ursprünglichen Daten und Werte bleiben wohlgemerkt stets erhalten, auch wenn in der `DataGrid`- oder `TileList`-Komponente nicht alle Daten angezeigt werden.

```
function imgDirResults(retVal) {
    //
    tab3.list_ThumbList.setStyle("cellRenderer", rendererCCThumb);  //Set CellRenderer For Thumblist
```

Abbildung V-44: Zuweisung des CellRenderers rendererCCThumb an die TileList „list_Thumblist“.

[Quelle: ControlCenter.as, Stand: 10.12.2008]

Man kann die Instanzen jedoch zur Editierung freigeben, so dass der Nutzer Eingaben für jede Zelle eines Elementes vornehmen kann, wie es auch bei der `PublishList`-Instanz geschieht.

Die grundsätzliche Layoutbasis des Interfaces bildet die Verwendungen von Reitern zur Menüstrukturierung sowie die Aufteilung der Oberfläche in themenverwandte Bereiche. Reiter bieten den Vorteil, dass die Ansichten und somit die Informationen schnell gewechselt werden können und dennoch eine Platz sparende Anordnung möglich ist. Die dunklen Schwarz-Grauverläufe sollen ein harmonisches zurückhaltendes grafisches Bild erzeugen, das den Fokus auf die Inhalte lenkt. Um die Textinformationen knapp und prägnant zu halten und eine Überflutung des Interfaces mit Text zu vermeiden, wurde mit Farbschemen gearbeitet, die als Unterscheidungsmerkmal dienen sollen, so z.B. im Bereich der Seitennavigation.

2.6 Die Endbenutzer Oberfläche

Das Front-End stellt dem End-User alle Informationen zu Verfügung, die über das Back-End festegelegt und publiziert wurden. Die Strukturen beider Interfaces sind somit sehr ähnlich, da mit den gleichen Streams und XML-Dateien gearbeitet wird. Es sollen an dieser Stelle also nur Eigenheiten und Differenzen des Front-Ends explizit herausgestellt werden.

2.6.1 Analogien und Unterschiede zum Administrationsmodul

Da es sich beim Front-End ebenso wie beim Back-End um Client-Applikationen handelt, müssen beide zur Kommunikation mit dem FMS ein `NetConnection`-Objekt erzeugen, um mit diesem eine Verbindung aufbauen zu können. Des Weiteren greifen beide auf dieselben externen Dateien und Verzeichnisse zu. Somit sind auch Parallelen in der Variablendeklaration vorhanden.

In der gesamten Anwendung kommt für die Übertragungen und Darstellung der Videostreams ausschließlich die `NetStream`-Klasse zum Einsatz, die `FLVPlayback`-Komponente wird auch im Front-End nicht eingesetzt, da das Handling mit der `NetStream`-Klasse für diese Anforderungen besser geeignet erschien.
Wesentlicher Unterschied ist jedoch, dass nur `NetStreams` zum FMS aufgebaut und nicht die `nc.call(„selectSource“)` Anweisungen zum Definieren der Inhalte der vier öffentlichen Streams (`steram1` bis `stream4`) ausgeführt werden. Der End-User ist also nur berechtigt die Inhalte der Streams zu betrachten, nicht zu setzen bzw. zu verändern.
Auch das Audio-Handling der Streams verhält sich identisch zu dem des Back-Ends.
Folglich werden zur Videodarstellung auch wieder Instanzen des Typs Video benutzt, die in die vier Anzeigefenster (s. Abbildung V-8) eingebettet sind.

Abbildung V-45: Screenshot des NetConnection-Status-Icons.

[Quelle: UserInterface.fla Stand: 10.12.2008]

Die Meta-Daten und `NET_STATUS`-Ereignisse werden im Front-End ebenso verarbeitet, dem End-User allerdings nicht bzw. auf andere Weise visualisiert. So dienen die Meta-Daten zunächst nur der Skalierung der Videos mit dem korrekten Bildseitenverhältnis. Andere Anwendungszwecke sind hier ggf. auch denkbar.
Ob der Nutzer erfolgreich mit dem FMS verbunden ist, wird ihm über eine Grafik (s. Abbildung V-45) mitgeteilt, nicht im Textformat.

Der Puffer für die Streamübertragungen wird mit dem bereits bekannten dynamischen zwei-schwelligen System variiert.
Nutzungsäquivalent ist ebenso die `TileList`-Klasse, die im Front-End ausschließlich zur Darstellung der XML-Inhalte Verwendung findet. Auch hier kommen wieder die `CellRenderer` zum Einsatz, obgleich jeder der drei `TileList`-Instanzen ein eigener `CellRenderer` zugewiesen wurde. Hier ist also eine individuelle Adaption nötig.

Die prägnantesten Unterschiede zwischen beiden Interfaces sind wohl in der Oberflächengestaltung zu finden. Das Back-End ist auf umfangreiche und gezielte Informationsvisualisierung ausgelegt, das Front-End ist auf die nötigsten Funktionen zur Videokontrolle und Auswahl der Streams reduziert worden. Logischerweise stehen die Videos selbst im Vordergrund. Nichts desto trotz steckt hinter beiden grafischen Benutzerschnittstellen eine stark verwandte Funktionalität.

2.6.2 Zugriff auf Streams und Informationen

Wie aus den Erläuterungen zum Server-side-Skript und dem Back-End hervorgeht, werden dem Front-End die Streamlisten im XML-Format übergeben.
Die XML-Dateien stehen entweder als externe Datei oder in einem Array des Server-side-Skripts zur Verarbeitung bereit. Um die Liste der publizierten Arrays (Live und VoD) aus der `main.asc` zu empfangen, muss die zugehörige Server-side-Funktion aufgerufen werden, die `getStreamList()`-Funktion (s. Abbildung V-46). Als Rückgabeobjekt wird ein `Responder` definiert, der die Ergebnisse an die `listHandler()`-Methode weiterleitet. Fehler bei der Abfrage werden an den `faultHandler()` übergeben. In der `listHandler()`-Funktion werden zunächst die `CellRenderer` zugewiesen und die Rückgabewerte in die Variable `xmlLoader` geschrieben (s. Abbildung V-47). Die folgende Verarbeitung des Arrays findet in der `initStreamList()`-Funktion (Abbildung A-7) statt und gleicht der einer extern geladenen XML-Datei. Es soll daher an dieser Stelle die Verarbeitung der XML-Dateien im UserInterface am Beispiel der CuePoints (`cuelist.xml`) gezeigt werden.

```
public function getStreamList():void {
        getUPCList();       //load upcominglist.xml
        nc.call("getPublishList", new Responder(listHandler, faultHandler));
}
```

Abbildung V-46: getStreamList()-Funktion. Aufruf der Server-side-Funktion zum Empfang des Arrays der publizierten Streams.

[Quelle: UserInterface.as, Stand: 10.12.2008]

```
/*---------------------------------
    STREAM & CUE LIST Handler
---------------------------------*/
public function listHandler(retVal) {
    if(retVal != null) {
        //Loader for Livestream-List and Stored-List
        list_container1.list_StreamList.setStyle("cellRenderer", rendererThumb);
        list_container2.list_StreamList.setStyle("cellRenderer", rendererThumb_stored);
        mc_CueList.list_CueList.setStyle("cellRenderer", rendererCuelist);
        xmlLoader = retVal;
```

Abbildung V-47: Auszug der listHandler()-Funktion. Zwischenspeicherung der XML-Werte des Server-side Arrays.

[Quelle: UserInterface.as, Stand: 10.12.2008]

Da hier keine Rückgabewerte der `main.asc` übergeben werden, müssen die externen Dateien zunächst geladen und die Inhalte extrahiert werden. Hierzu bietet Flash zwei in Kombination stehende Klassen, `URLRequest` und `URLLoader`. Die `URLRequest`-Funktion sammelt und bündelt zunächst alle übergebenen Informationen und wandelt sie in ein `Object` um. Dieses wiederum beinhaltet einen `data`-Knoten, der alle heruntergeladenen Informationen (unformatiert) beinhaltet. Die Daten aus `data` können sowohl gelesen als auch beschrieben werden. Das `URLRequest`-Objekt wird hier in der Variablen `cueXMLURL` gespeichert und an die `URLLoader`-Funktion übergeben. Diese wandelt die Inhalte in Text, binäre Daten oder URL-kodierte Werte um, was in diesem Fall geeignet für die XML-Daten ist. Im Gegensatz zur `URLRequest`-Funktion können die Daten nur geladen und nicht geschrieben werden. Das Ergebnis der Umwandlung wird in der Variablen `cueLoader` gespeichert. Der Variablen `cueLoader` wird anschließend ein `EventListener` hinzugefügt, der für den Aufruf der Weiterverarbeitungsfunktion `CueXMLLoaded()` zuständig ist, sobald alle Daten vollständig geladen wurden und das Ereignis `COMPLETE` eingetreten ist.

In der `CueXMLLoaded()`-Funktion werden zu Beginn die Daten aus der `cueLoader`-Variablen über `cueXML = XML(cueLoader.data)` in eine XML-Variable geschrieben, die im weiteren Verlauf über eine `for each`-Schleife abgearbeitet wird. Der Befehl `for each (var cpitem:XML in cueXML.citem)` bildet eine Schleife, die alle Knotenpunkte mit dem Bezeichner `citem` innerhalb der XML durchläuft und deren Werte in der Variablen `cpitem` des Typs XML zwischenspeichert. So können alle Knoten und Werte jedes einzelnen Elements über `cpitem` abgefragt werden (Aufbau der `cuelist.xml` s. Abbildung A-8).
Die CuePoints werden jedoch immer nur für das derzeit ausgewählte Element in die VoD-Liste (s. Abbildung V-7) geladen.
Über das Event `CHANGE` der Liste `list_StreamList` wird bei jedem Selektionswechsel die `getCueXML()`-Funktion (s. Abbildung V-48) erneut aufgerufen und als Parameter der Titel sowie Dateiname des Listenelements übergeben. Somit ist eine `if`-Kondition nötig, die Sorge dafür trägt, dass die zum selektierten Element gehörigen CuePoints geladen werden. Ist die Bedingung `if (citem.@name == actitemFile){}` erfüllt, so werden alle Befehle innerhalb der geschweiften Klammern abgearbeitet. Der Wert `actitemFile` trägt den Dateinamen als `String` des im `DataGrid list_StreamList` ausgewählten Elements. Eine zweite verschachtelte `for each`-Schleife durchläuft nun alle CuePoints des aktuellen `citem`-Knotens. Die Werte der CuePoints werden in Variablen zwischengespeichert und in ein mehrdimensionales Array, `cuelist`, geschrieben. Sind alle CuePoints des aktuellen Elements durchlaufen, werden die Daten aus dem Array in die `list_CueList` geschrieben und mittels `CellRenderer` entsprechend formatiert.

```
private function ShowCuePoints (e:Event):void {
    var actitem=list_container2.list_StreamList.selectedItem;
    var actitemFile=list_container2.list_StreamList.selectedItem.data;
    ShowCueBar();
    mc_CueList.lbl_ActCueItem.text = actitem.label;
    ShowCueThumbs2();

    getCueXML(actitem,actitemFile);
}
```

Abbildung V-48: ShowCuePoints()-Funktion. Aufruf zum Laden der cuelist.xml über getCueXML().

[Quelle: UserInterface.as, Stand: 10.12.2008]

```
public function getCueXML(actitem:Object, actitemFile:String):void {
    //should be a relative path or url! couldnt get it working!
    var XML_URL:String = appRoot+"xml/"+cueXMLFilename;
    var cueXMLURL:URLRequest = new URLRequest(XML_URL);
    cueXMLURL.contentType="text/xml";
    var cueLoader:URLLoader = new URLLoader(cueXMLURL);
    cueLoader.addEventListener(Event.COMPLETE, CueXMLLoaded);
    cueLoader.addEventListener(IOErrorEvent.IO_ERROR, XMLErrorHandler);

    function XMLErrorHandler(e:IOErrorEvent):void {
        trace("Error opening cuelist.xml");
        list_container2.dialogCueList.visible = true;
    }

    function CueXMLLoaded(event:Event):void
    {
        cueXML = XML(cueLoader.data);
        var cuelist:Array = new Array();
        //Fill Selectable List with XML Item Titles
        for each(var citem:XML in cueXML.cueitem){
                // Get thumbnail value and assign to cellrenderer.
                if (citem.@name == actitemFile) {
                    // Send data to livelist or storedlist, depending on livestatus node.
                    var cue...
                    for each(var cpitem:XML in cueXML.cueitem.(@name==actitemFile).cuepoint){
                        var cuePName = cpitem;
                        var seekTime:String = cpitem.@time.toXMLString();
                        var Offset:String = cpitem.@offset.toXMLString();
                        var seekTimeNo:Number = parseInt(seekTime) + parseInt(Offset);
                        var seekTimeFormat:String = formatTime(seekTimeNo);
                        cuelist.push({label:cuePName, data:cuename, seekformat:seekTimeFormat, seekto:seekTimeNo});
                    }//endfor2
                }//end if
        }//endfor
        if (mc_CueList.list_CueList.length !=0) mc_CueList.list_CueList.removeAll();
        mc_CueList.list_CueList.dataProvider = new DataProvider(cuelist);
    }//End function CueXMLLoaded
```

Abbildung V-49: getCueXML()-Funktion. Laden der CuePoint Informationen aus der cuelist.xml.

[Quelle: UserInterface.as, Stand: 10.12.2008]

Der Vollständigkeit halber ist zu erwähnen, dass das Laden der `upcominglist.xml` auf ähnlich Weise statt findet. Allerdings werden dabei, wie für das Live-Stream Array, alle XML-Knoten geladen.

2.6.2.a Differenzierung zwischen Live und Video-On-Demand

Die Unterscheidung zwischen Live und VoD Dateien ist nötig, da zwei unterschiedliche Listen für jede Kategorie angewendet werden, die über differenzierte Funktionen verfügen. So sind die Informationen für Live-Streams begrenzt, die Liste (s. Abbildung V-6) gibt ein Miniaturbild, den Stream-Namen sowie den Erstellungszeitpunkt wieder. Wohingegen die Liste der VoD-Streams neben der Liste der gespeicherten Streams bei Mausklick ein seitliches Register öffnet, dass die zugehörigen CuePoints anzeigt. Somit soll der Schnellzugriff auf einzelne Szenen ermöglicht werden. Zu jedem CuePoint wird hierzu bei Auswahl ein Screenshot des Videos an dieser Stelle angezeigt.
Die Differenzierung findet über die `if`-Kondition (s. Abbildung V-50) statt. Hat der Inhalt des XML-Knotens `live` den Wert `true`, wird das Element dem `livelist`-Array hinzugefügt, andernfalls erfolgt eine Addition zum `storedlist`-Array.

```
// Send data to livelist or storedlist, depending on livestatus node.
if (item.livestatus == "Yes") livelist.push({label:actname, data:item.stream.@source.toXMLString(),duration:dur, created:tim
else storedlist.push({label:actname, data:item.stream.@source.toXMLString(),duration:dur, created:timeString, source:thumb,
```

Abbildung V-50: Auszug der initStreamList()-Funktion. If-Kondition zur Differenzierung zwischen Live und VoD.

[Quelle: UserInterface.as, Stand: 10.12.2008]

2.6.2.b Newsticker im RSS-Format

Die `URLLoader`-Klasse bietet die Möglichkeit auch RSS-Dateien aus dem Internet zu laden. Da diese generell im XML-Format vorliegen, ist die Vorgehensweise identisch mit dem Laden der anderen XML-Dateien dieser Applikation. Um die einzelnen Konten der für die Anwendung interessanten RSS-Datei darzustellen, sollte im Vorfeld mit einem Webbrowser der Quellcode der `rss.xml` betrachtet werden, damit die hierarchische Struktur der Datei erkennbar ist. So wird deutlich mit welchen Pfaden auf die einzelnen Knoten zugegriffen werden muss. Der Aufbau der im Web verfügbaren RSS-Dateien ist grundsätzlich normiert, jedoch treten immer wieder Unterschiede auf, auch unter dem Aspekt der verschiedenen Versionen von RSS1.0 und RSS2.0. Eine Kontrolle ist also sinnvoll.

```
public function initRSSList():void {
    var RSS_URL:String = "http://www.sport1.de/de_1/fussball/fussball_bundesliga/rss.xml";
    var myRSSURL:URLRequest = new URLRequest(RSS_URL);
    var rssLoader:URLLoader = new URLLoader(myRSSURL);
    rssLoader.addEventListener(Event.COMPLETE, rssXMLLoaded);

    function rssXMLLoaded(event:Event):void
    {
        var rssXML:XML = XML(rssLoader.data);
        var outXML:Array = new Array();
        for each (var rssitem:XML in rssXML.channel.item)
        {
            var itemTitle:String = rssitem.title;
            var itemDescription:String = rssitem.description;
            var itemLink:String = rssitem.href;
            outXML.push({label:itemTitle, data:itemDescription, source:itemLink});
        }
        list_container3.list_TickerList.selectable = false;
        list_container3.list_TickerList.dataProvider = new DataProvider(outXML);
    }//end function rssXMLLoaded
}//End public function initRSSList
```

Abbildung V-51: Laden der externen rss.xml mit der initRSSList()-Funktion.

[Quelle: UserInterface.as, Stand: 10.12.2008]

Zum Laden der `rss.xml` wird zunächst der absolute Pfad zur Datei angegeben, in diesem Fall eine Webadresse (s. Abbildung V-51). Nun kann von dieser Adresse ein `URLRequest`-Objekt erzeugt werden, das wiederum dem `URLLoader`-Objekt übergeben wird. Der nachfolgende `EventListener` gibt, wie in Kapitel V Abschnitt 2.6.2 anfangs erläutert, das Ergebnis des Ladevorgangs an die verschachtelte Funktion `rssXMLLoaded()` weiter. Diese speichert die Daten in der Variablen `rssXML` des Typs `XML`. Anschließend können über die `for each`-Schleife die Knoten innerhalb des Mutterknotens `channel.item` sukzessive abgefragt werden. Die gewünschten Elemente, in diesem Fall `title`, `description` sowie `href`, also ein Link, werden in ein Array geschrieben, das außerhalb der Schleife der `TileList list_TickerList` hinzugefügt wird.

2.6.3 Darstellungsoptionen und Wiedergabe der Streaminhalte

Wie in Kapitel V Abschnitt 1.3.2 angedeutet gibt es drei verschiedene Darstellungsmodi (s. Abbildung V-52).

Abbildung V-52: Die 3 Darstellungsmodi des Front-Ends.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

Die Symbolschaltflächen links unten im Front-End dienen dem Wechsel der Modi. Der erste Modus aktiviert die Einzeldarstellung eines Videofensters. Hierbei wird herausgefiltert, welches Video (Stream) den derzeit aktivierten Audiokanal „besitzt“ und abhängig davon dieses Videofenster für die maximierte Ansicht vergrößert. Somit wird verhindert, dass versehentlich ein Videofenster vergrößert wird, das im Tonkanal nicht aktiv ist. Lästiges hin und her Klicken soll so vermieden werden. Der zweite Darstellungsmodus positioniert zwei verkleinerte Videofenster horizontal nebeneinander und zentriert diese vertikal. Der dritte und letzte Darstellungsmodus verwendet dieselbe Videofenstergröße wie der sekundäre Darstellungsmodus, verteilt jedoch vier Videofenster gleichmäßig auf der Anzeigefläche.

In den linken Listen können die Live- oder VoD-Streams vom User selektiert werden, was Voraussetzung dafür ist, dass er sie in ein Videofenster laden kann. Abgespielt wird ein in der Live- oder VoD-Liste ausgewähltes Element, indem der User mit der Maus über ein Videofenster fährt und dadurch eine Schaltfläche einblendet (Abbildung V-53), die es dem User ermöglicht, den Stream in das Video-Fenster zu laden. Bei einem Klick auf diese wird automatisch die `NetStream`-Verbindung aufgebaut, die ausgewählte Videodatei resp. der ausgewählte Stream geladen und abgespielt.
Für jedes Videofenster kann dabei unabhängig die Lautstärke geregelt werden. Zudem signalisiert ein Schriftzug, ob es sich beim abgespielten Stream um eine Live-Übertragung handelt oder nicht.

Abbildung V-53: Screenshot eines Videofensters im Hover-Status im UserInterface.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

2.6.4 Grafisches Konzept und Umsetzung unter Usability Gesichtspunkten

Beginnend mit der Farbgebung lassen sich einige Parallelen zum Back-End ziehen. Beide Interfaces sind in dunklen Grau-Schwarz-Verläufen gehalten. Dies soll die kolorierten Inhalte der Videos sowie die farblich hervorgehobenen Schaltflächen stärker betonen. Da es sich um listenbasierte Textdarstellungen handelt und keine langen Fließtexte gelesen werden müssen, schien die Wahl eines dunklen Hintergrundes geeignet. Damit wird die Gesamtleuchtdichte der Applikation reduziert und eine zu große visuelle Überanstrengung des Betrachters ver-

mieden. Auch die weichen Verläufe sollen zur unangestrengten Betrachtung des Interfaces beitragen.
Beurteilt man nun das Front-End im Speziellen, so fällt zunächst ein eher minimalistisches Layout ins Auge. Der Hintergrund ist in thematischer Hinsicht mit den Darstellungsinhalten verknüpft, was den Verschmelzungsgrad von Benutzer und Applikation erhöhen soll. Die Empathie, die Einfühlung des Benutzers in den Kontext, soll so vergrößert werden. Erweiterungsoption an dieser Stelle wäre ein für den User frei wählbares Grafikschema (Theme) bzw. eine Anpassungsoption durch den Administrator.

Die meisten funktionalen Prozesse arbeiten im Hintergrund und bleiben dem User verborgen. Nur die nötigsten Funktionen stehen dem User zur Interaktion zur Auswahl.
So wird die Benutzung der Anwendung auf das Wesentliche reduziert: Die Wiedergabe von Medieninhalten.
Jedoch muss der User stets wissen was er sieht und welcher Kategorie (Live oder VoD) es zuzuordnen ist. Kernpunkt der Interaktion ist die Eigenregie des Users. So stehen ihm neben den vier vom Administrator gesteuerten Kanälen (`stream1` bis `stream4`) auch weitere Streams zur Verfügung, sowohl Live als auch VoD. Er kann also selbst wählen ob er sich der Regie des Administrators anschliesst oder selbst genau definiert, wo welcher Stream läuft.
Die Freiheit der Auswahl soll zudem durch die Kombination aus Live und VoD Streams vergrößert werden. Für die Anwendung ist vorgesehen, dass es zu jedem Live-Stream auch eine zugehörige gespeicherte Videodatei gibt, die dem Benutzer die Möglichkeit bietet, sich verpasste Szenen oder ein verpasstes Ereignis im Ganzen zu einem individuellen Zeitpunkt anzuschauen. Aus den im Vorfeld erläuterten technischen Gründen ist die Betrachtung der VoD-Dateien in Relation zum Live-Stream erst nach einer kurzen Unterbrechung und dem Finalisieren der Videodatei möglich.
Um die Gewohnheit des Benutzers noch stärker anzusprechen bietet ein RSS-Ticker die Verknüpfungskomponente zu bisherigen Liveübertragungs-Portalen in sportlicher Thematik im Internet. Zudem wird der Informationsgrad bei mehreren simultanen Ereignissen erhöht, da über den Text-Ticker alle Kernereignisse protokolliert werden. Die Anzahl der gleichzeitig erfassbaren (Bild-)Informationen ist naturgemäß begrenzt. Der Benutzer ist gezwungen seine Aufmerksamkeit zu fokussieren. Daher soll es auch dem Benutzer überlassen werden, wie viele Videos er zur simultanen Betrachtung nutzt.

Der Zeitraum zwischen Auswahl eines Streams und Abspielen des Selbigen soll so kurz wie möglich gehalten werden. Ausserdem ist es wichtig, viele Schaltflächen, die das Interface überladen könnten, zu vermeiden. Das Mouse-Over Ereignis bietet hier eine gute Lösungsmöglichkeit. Der Benutzer kennt diese Funktionalität bereits von gewöhnlichen Internetseiten, aus den lokalen Betriebssystemen sowie aus anderen Programmen. Die Schaltflächen sind zudem meist als Symbole umgesetzt und verzichten weitestgehend auf Text, da der Benutzer auf Form- und Farbschemen wesentlich empfindlicher und intuitiver reagiert als auf Textbotschaften. Sollte eine Symbol nicht direkt selbsterklärend oder ubiquitär bekannt sein, tritt abermals das Hover-Ereignis als Erklärungskomponente mit einer kurzen Textzeile in Kraft (vgl. Abbildung V-53). Zum Abspielen eines Streams wird also zunächst das Hover-Ereignis genutzt um die `LOAD` Schaltfläche darzustellen. Ein Klick hierauf und das aktuelle Video wird in das zur Schaltfläche gehörige Videofenster geladen.

Der Hover-Bereich eines Videofensters ist ein zwei Elemente unterteilt. Das untere Drittel visualisiert eine zusätzliche Kontrolleinheit, die Aufschluss über Titel, Dauer, Lautstärke und einen Start-/Stop-Knopf bietet.
Für den Benutzer ist des Weiteren über eine Checkbox^{G} wählbar, ob die Stream-Listen automatisch in einem fest definierten Intervall aktualisiert werden sollen. Eine manuelle Aktualisierung ist über eine nebenstehende Schaltfläche ebenfalls verfügbar.
Mit den bereits beschriebenen Stream-Listen endet die Interaktionsmöglichkeit des Benutzers. Er soll sich somit innerhalb möglichst kurzer Zeit der reinen Videobetrachtung hingeben und die Inhalte auswählen können, die er betrachten möchte.

VI SCHLUSSBETRACHTUNG

1 Ungelöste Probleme und Optimierungsmöglichkeiten

Aufgrund des prototypischen Charakters der Applikation ließen sich noch vorhandene Fehlerquellen nicht ausschließen. Folgende Liste soll Aufschluss über vorhandene Funktionslücken sowie unvollständige Programmroutinen geben:

(01) Dynamische Bandbreitenanpassung abhängig von den Datenübertragungsraten der Clients (betrifft: `main.asc`)

(02) Auslagerung einiger Funktionen in exterene Actionscriptpakete (betr. Back- und Front-End)

(03) Umfangreiche Sicherheitsoptionen und –Einstellungen der Applikation und der Streams [vgl. http://www.adobe.com/devnet/flashmediaserver/security.html, Digital Rights Management] (betr.: `main.asc`, Back-End, eventuelle Auswirkungen auf das Front-End)

(04) Zusätzliche Zugriffs- und Benutzerkontrolle (betr.: Back-End)

(05) Verbesserungen beim Video-Kontroll-Panel (betr.: Front-End)

(06) De-Selektierung der Listenelemente nach Aktualisierung verhindern (betr.: Front- und Back-End)

(07) Zusätzliche Zugriffsmöglichkeiten auf die Voreinstellungen und Variablenwerte über das Benutzer-Interface (betr.: Back-End)

(08) CuePoint-Bearbeitung auch für externe Clients ermöglichen (betr.: neues Zusatzmodul)

(09) Automatisches Löschen veralteter Streams aus den XML-Dateien (betr.: Back-End)

(10) Vollbildmodus für das Front-End ermöglichen (betr.: Front-End)

(11) Dynamische Anpassung der Front-End Hintergründe durch den Administrator (betr.: Back-End und Front-End)

(12) „multi-bit rate“ Kodierungssystem mit dynamischer Anpassung an die Datenübertragungsrate des Clients

Bislang unbekannte Probleme und Verbesserungsansätze sollten über eine Evaluierungsphase mit geeigneten Probanten determiniert und offengelegt werden. Dadurch ergeben sich voraussichtlich weitere Optimierungsschwerpunkte.

2 Fazit & Danksagungen

Die im Rahmen dieser Arbeit entwickelte Applikation besitzt den Status einer Beta-Version und ist daher als Prototyp anzusehen. Somit sind, wie im vorigen Kapitel erläutert, bewusste und unbewusste Fehler vorhanden deren Korrektur erforderlich wäre. Eventuelle Fehleingaben oder ungewollte Abfolgen des Benutzers müssen erfasst und verbessert werden.

Was die Anwendungsperspektive und das Entwicklungspotential der Applikation anbelangt, so unterstützt die voranschreitende Verbreitung von Live-Übertragungen über das Internet die thematische Relevanz. Die Entwicklung des Marktes ist weiter zu beobachten, um Trends zu erkennen und die Ansprüche der Benutzer, sowohl professioneller als auch laienhafter Natur zu erfassen und in die Optimierung der Applikation einfließen zu lassen. So bieten z.B. die Meta-Daten ein großes Ergänzungspotential für Mediendateien. Zusatzinhalte könnten hinzugefügt und der Informationsgrad erhöht werden.

Auch die Systeme und Anwendungen des Web 2.0 sollten verstärkt Berücksichtigung finden. Eine zentrale, noch weiter auszuformende Bedeutung besitzt das Benutzerfeedback und die Interaktionsmöglichkeit des Betrachters. Bewertungsoptionen, Kommentarfelder und weitere Anwendungen sollten auf Eignung und Umsetzbarkeit für dieses System geprüft werden.
Der Flash Media Server wird stetig weiterentwickelt, so wechselte die Version während des Entwicklungsprozesses dieser Applikation von 3.0 auf 3.5. Es sind also weitere neue Funktionen zu erwarten, die größeren Spielraum für neue Systeme bieten. Tendenzen zur Ausweitung des Interessentenkreises und der Anwendungsoptionen sind deutlich erkennbar.

Auch die sukzessive Verschmelzung von Internet und klasischem Fernsehen und die damit verbundene Tendenz der Lean-Forward Nutzung von Medieninhalten sollte im Blickfeld bleiben. Die Tauglichkeit von Internetanwendungen für die Bedienung am „heimischen“ Fernseher sollte mit zunehmender Relevanz einkalkuliert und getestet werden.

Danksagungen

Ich möchte als Autor die Gelegenheit nutzen und mich an dieser Stelle für die unkomplizierte Betreuung bei Prof. Grünvogel bedanken. Ein besonderer Dank gilt auch meinem Bruder, Thomas Lux, der mir durch zahlreiche Denkanstöße und das Korrekturlesen dieser Arbeit sehr geholfen hat. Ein Dankeschön möchte ich auch den Mitgliedern der FlashComGuru.com Mailinglist aussprechen, die durch prompte Antworten einige Problemstellungen auflösen konnten.

VII QUELLENVERZEICHNIS

1 Literatur

BÖSKEN, M. (2007): Streaming-Video und Web-TV. In: Diplomica GmbH, Hamburg (Hrsg): 06/2007, S. 1-32.

COOPER, A., R. REIMANN UND D. CRONIN (2007): About Face 3. The Essentials of Interactive Design. In: Wiley Publishing Inc. (Hrsg): 2007, S. 77-108 u. S. 434-556.

LOUIS, D. (2004): ActionScript. Einfache Programmierung mit Flash MX und MX 2004. Markt&Technik (Hrsg): 2004.

RICHTER, M. und M. FLÜCKIGER (2007): Usability Engineering Kompakt: Benutzbare Software gezielt entwickeln. In: Spektrum Akademischer Verlag (Hrsg): 2007, S. 1-41.

SANDERS, B. (2008): Learning Flash Media Server 3. In: O'Reilly (Hrsg): 3/2008.

2 World Wide Web

ACTIONSCRIPT.ORG — http://www.actionscript.org/, [Zugriff: 24.11.2008]

ADOBE
- Streaming und progressiver Download, http://www.adobe.com/de/products/hdvideo/supported_technologies/streaming.html, [Zugriff: 05.12.2008]
- Flash Media Live Encoder 3, http://www.adobe.com/products/flashmediaserver/flashmediaencoder/features/, [Zugriff: 11.12.08]

ADOBEFORUMS — User to user forums > cue point frustration, http://www.adobeforums.com/webx/.59b6953e, [Zugriff: 26.12.2008]

APPLE — MPEG-4 - the container for digital media, http://www.apple.com/quicktime/technologies/mpeg4/, [Zugriff: 11.12.2008]

BRANCA, FABRIZIO — Phasen, Formate und Techniken zur Videoproduktion, http://www.vikar.de/uploads/media/StA_Fabrizio_Branca.pdf, [Zugriff: 05.12.2008]

FLASH MEDIA SERVER BLOG — Archive for the 'Flash Media Encoder' Category, http://www.flashmediaserver-blog.de/category/flash-media-encoder/, [Zugriff: 26.12.2008]

FLASHCOMGURU
- http://www.flashcomguru.com/, [Zugriff: 20.12.2008]
- http://community.lsoft.com/scripts/WA-LSOFTDONATIONS.exe?A0=flashmedia, Maling List, [Zugriff: kontinuierlich]

FMSGURU.COM — BULL, G. - The basics of the File object on the server side of Flash Media Server , http://fmsguru.com/tutorials.cfm, [Zugriff: 26.12.2008]

HERRLICH & RAMUSCHKAT GMBH — Adobe Consulting > Flash Media Server Consulting > Streaming vs. Progressive, http://www.richinternet.de/index.cfm?uuid=9CE8A873BACCDD0CD3

	06D8C511D3E816&title=Streaming vs. Progressive, [Zugriff: 05.12.2008]
INTERNETWORLDSTATS	http://www.internetworldstats.com/stats4.htm - europe, [Zugriff: 03.12.08]
ITWISSEN	H.264/AVC (H.264 advanced video coding), http://www.itwissen.info/definition/lexikon/H-264-advanced-video-coding-H-264-AVC-H-264.html, [Zugriff: 11.12.2008]
NATIONMASTER.COM	Encyclopedia > Sorenson Spark, http://www.nationmaster.com/encyclopedia/Sorenson-Spark, [Zugriff: 11.12.2008]
ON2 TECHNOLOGIES INC.	Optimized software video codecs for embedded applications, http://www.on2.com/index.php?144, [Zugriff: 11.12.2008]
SENOCULAR.COM	XML (E4X), http://www.senocular.com/flash/tutorials/as3withflashcs3/?page=4, [Zugriff: 26.12.2008]
STREAMING MEDIA WEST	OZER, J. - Pre-Conference: Encoding H.264 Video for Streaming and Progressive Download (09/2008), http://www.streamingmedia.com/west/presentations/SMWest2008-H264.pdf, [Zugriff: 20.12.2008]
STUDIE DEUTSCHLADN ONLINE	http://www.studie-deutschland-online.de/do5/2200.html, [Zugriff: 03.12.08]
WEBHITS	http://www.webhits.de/deutsch/index.shtml?webstats.html, [Zugriff: 18.12.08]
WIKIPEDIA	http://de.wikipedia.org, [Zugriff: 28.12.2008]

Nachfolgende Links geben einen Überblick über kommerzielle strukturell ähnliche Streaming Anwendungen

INFLUXIS	http://www.influxis.com/, [Zugriff: 26.12.2008]
LIGX LIVE MEDIA SERVICE GMBH	http://www.ligx.de/, [Zugriff: 26.12.2008]
TV1	http://www.tv1.eu/de/index.jsp, [Zugriff: 26.12.2008]
WEBZOOMS.TV	http://www.webzooms.tv/, [Zugriff: 26.12.2008]

ADOBE LIVEDOCS - FLASH

ACTIONSCRIPT 2.0 COMPONENTS LANGUAGE REFERENCE	Flash CS3 Documentation > ActionScript 2.0 Components Language Reference, http://livedocs.adobe.com/flash/9.0/main/Part6_AS2_Components_LangRef_1.html, [Zugriff : 26.12.2008]
ACTIONSCRIPT 3.0 LANGUAGE AND COMPONENTS REFERENCE	ActionScript 3.0 Language and Components Reference, http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/index.html, [Zugriff : 26.12.2008]
ON2 VP6- UND SORENSON SPARK VIDEO-CODECS IM VERGLEICH	On2 VP6- und Sorenson Spark Video-Codecs im Vergleich, http://livedocs.adobe.com/flash/9.0_de/UsingFlash/help.html?content=WSd60f23110762d6b883b18f10cb1fe1af6-7ca8.html, [Zugriff : 11.12.2008]

USING VIDEO METADATA	Programming ActionScript 3.0 > Working with video > Using video metadata, http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00000267.html, [Zugriff: 16.12.2008]
WORKING WITH CUE POINTS	Flash CS3 Documentation > Learning ActionScript 2.0 in Adobe Flash > … > Working with cue points, http://livedocs.adobe.com/flash/9.0/main/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Parts&file=00001037.html, [Zugriff: 26.12.2008]
WORKING WITH XML	Flash CS3 Documentation > Programming ActionScript 3.0 > Working with XML, http://livedocs.adobe.com/flash/9.0/main/00000122.html, [Zugriff : 26.12.2008]

ADOBE LIVEDOCS - FMS

CLIENT-SIDE ACTIONSCRIPT LANGUAGE REFERENCE	Flash Media Server > Client-Side ActionScript Language Reference for Flash Media Server 2 > Client-Side ActionScript Language Reference http://livedocs.adobe.com/fms/2/docs/00000523.html, [Zugriff : 26.12.2008]
SERVER-SIDE ACTIONSCRIPT LANGUAGE REFERENCE	Flash Media Server > Server-Side ActionScript Language Reference-Server-Side ActionScript Language Reference, http://livedocs.adobe.com/fms/2/docs/00000630.html, [Zugriff: 26.12.2008]
USING THE ADMIN COMMANDS	Managing Flash Media Server > Deploying Flash Media Server > About configuration levels > Using the admin commands , http://livedocs.adobe.com/fms/2/docs/00000208.html, [Zugriff: 25.12.2008]

ADOBE FLASH MEDIA SERVER DOCUMENTATION

ADMINISTRATION API REFERENCE	http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=Book_Part_35_serv_man_asd_1.html, [Zugriff: 20.12.2008]
CONFIGURATION AND ADMINISTRATION GUIDE	http://livedocs.adobe.com/flashmediaserver/3.0/docs/help.html?content=Book_Part_36_admin_1.html, [Zugriff: 20.12.2008]
DEVELOPER GUIDE	http://livedocs.adobe.com/flashmediaserver/3.0/hpdocs/help.html?content=Book_Part_31_deving_1.html, [Zugriff: 20.12.2008]
INSTALLATION GUIDE	http://livedocs.adobe.com/flashmediaserver/3.0/docs/help.html?content=Book_Part_30_installing_1.html, [Zugriff: 20.12.2008]
TECHNICAL OVERVIEW	http://livedocs.adobe.com/flashmediaserver/3.0/docs/help.html?content=Book_Part_28_tech_overview_1.html, [Zugriff: 20.12.2008]

ADOBE FLASH MEDIA DEVELOPER CENTER

CALCULATING BANDWIDTH NEEDS FOR FLASH MEDIA SERVER 3	http://www.adobe.com/devnet/flashmediaserver/articles/calculating_bandwidth.html, [Zugriff: 02.11.2008]
CREATING A DYNAMIC PLAYLIST FOR STREAMING FLASH VIDEO	LARSON-KELLEY, L. - http://www.adobe.com/devnet/flash/articles/video_player.html,

	[Zugriff: 28.10.2008]
DYNAMIC STREAM SWITCHING WITH FLASH MEDIA SERVER 3	HASSOUN, D. - http://www.adobe.com/devnet/flashmediaserver/articles/dynamic_stream_switching.html, [Zugriff: 29.10.2008]
FLASH VIDEO (FLV) BITRATE CALCULATOR	REINHARDT, R. - http://www.adobe.com/devnet/flash/apps/flv_bitrate_calculator/, [Zugriff: 05.12.08]
IMPLEMENTING A DUAL-THRESHOLD BUFFERING STRATEGY IN FLASH MEDIA SERVER	SONNATI, F. - http://www.adobe.com/devnet/flashmediaserver/articles/fms_dual_buffering.html, [Zugriff: 03.10.2008]
OVERVIEW OF STREAMING WITH FLASH MEDIA SERVER 3	LARSON-KELLEY, L. - http://www.adobe.com/devnet/flashmediaserver/articles/overview_streaming_fms3.html, [Zugriff: 25.09.2008]
SICHERHEITSÄNDERUNGEN IN FLASH PLAYER 8	MEKETA, D. - http://www.adobe.com/de/devnet/flash/articles/fplayer8_security_04.html, [Zugriff: 26.12.2008]
USING FILE OBJECT FOR VIDEO ON DEMAND AND MP3 PLAYBACK	SANDIE, R. - http://www.adobe.com/devnet/flashmediaserver/articles/on_demand_player.html, [Zugriff: 18.10.2008]
WORKING WITH METADATA FOR LIVE FLASH VIDEO STREAMING	LOEFFLER, J. - http://www.adobe.com/devnet/flashmediaserver/articles/metadata_video_streaming.html, [Zugriff: 10.10.2008]

VIII GLOSSAR

Back-End Benutzeroberfläche zur Administratoren eines Systems.

Bildframes Digital gespeicherte Videos bestehen aus einzelnen Bildern, den Frames. Pro Sekunde werden je nach Aufnahme und Wiedergabe Standard ca. 25fps, als 25 Bilder pro Sekunde wiedergegeben.

BootCamp Software die eine native Microsoft Windows Installation an Hand einer gesonderten Partition auf einem Apple Macintosh Rechner ermöglicht.

Broadcaster Ein Rundruf bzw. Broadcast in einem Computernetzwerk ist eine Nachricht, bei der Datenpakete von einem Punkt aus an alle Teilnehmer eines Netzes übertragen werden.

[vgl. http://de.wikipedia.org/wiki/Broadcasting, Zugriff: 28.12.2008]

Button Schaltfläche, die dem Benutzer ermöglicht, eine dem Steuerelement zugeordnete Funktion auszulösen

[vgl. http://de.wikipedia.org/wiki/Button, Zugriff: 28.12.2008]

Checkbox Eine Checkbox (engl. für Auswahlkasten, Kontrollkästchen) ist ein Standardelement einer graphischen Software-Benutzungsoberfläche das meist zwei Zustände definiert: Aktiviert oder nicht aktiviert.

[vgl. http://de.wikipedia.org/wiki/Checkbox, Zugriff: 28.12.2008]

Client Als Client (zu deutsch Kunde) wird ein Computerprogramm bezeichnet, das nach dem Client-Server-Modell Verbindung mit einem Server aufnimmt und Nachrichten mit diesem austauscht.

[vgl. http://de.wikipedia.org/wiki/Client, Zugriff: 28.12.2008]

Content delivery network Ein Content Delivery Network (auch Content Distribution Network, kurz CDN) ist ein Netz lokal verteilter und über das Internet verbundener Server, mit dem Content (insbesondere große Mediendateien) ausgeliefert wird.

[vgl. http://de.wikipedia.org/wiki/Content_Delivery_Network, Zugriff: 28.12.2008]

Content-Management-System Ein Content-Management-System (kurz CMS) ist ein Anwendungsprogramm, das die gemeinschaftliche Erstellung und Bearbeitung des Inhalts von Text- und Multimedia-Dokumenten ermöglicht und organisiert, meist für das World Wide Web. Ein Autor kann ein solches System auch ohne Programmier- oder HTML-Kenntnisse bedienen. Inhalt und Design sind voneinander getrennt.

[vgl. http://de.wikipedia.org/wiki/Content-Management-System, Zugriff: 28.12.2008]

CuePoint	Szenen-Sprungpunkt in einer Videodatei.
Edge server	Server der als zusätzicher Hilfsserver für einen anderen Server genutzt wird um bspw. die Verteilung von Streams zu übernehmen oder weitere Kapazitäten zur Verfügung zu stellen.
Framerate	Wiederholrate der Einzelbilder (Frames).
Front-End	Benutzeroberfläche für die Endbenutzer eines Systems.
High-Speed DSL	DSL Internetverbindung mit Geschwindigkeiten von über 16Mbit/s.
HTTP-Protokoll (Tunnel)	Ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich eingesetzt, um Webseiten und andere Daten aus dem World Wide Web (WWW) in einen Webbrowser zu laden. [vgl. http://de.wikipedia.org/wiki/HTTP, Zugriff: 28.12.2008]
Installer	Dialogfenster gestützte Routine, die der Installation einer Software dient.
Integerwert	Ganzzahliger Wert.
Interface	Schnittstelle, die der Kommunikation zwischen zwei Medien, z.B. Benutzer und Computer dient.
iTV	Interaktives Fernsehen nach dem “Lean-Forward”-Prinzip, also der aktiven Beteiligung des Betrachters.
Mouseover	Bewegen der Maus über eine Bildschirmfläche bzw. ein Bildschirmobjekt.
MPEG	Motion Picture Expert Group. Konsortium, dass den Aufbau MPEG-kodierter Mediendateien standardisiert hat.
MySQL	Datenbanksystem.
Open Source	Open source bzw. quelloffen ist eine Palette von Lizenzen für Software, deren Quelltext öffentlich zugänglich ist und durch die Lizenz Weiterentwicklungen fördert. [vgl. http://de.wikipedia.org/wiki/Open_source, Zugriff: 28.12.2008]
Panning	Wechseln des Tonkanals (bei Stereo-Ton) von Links nach Rechts oder umgekehrt.
Parallels Desktop Virtual-Machine	Software die eine Microsoft Windows Partition auf einem Apple Macintosh Betriebssystem emuliert. Apple Mac-OS und Microsoft Windows laufen hierbei parallel und teilen sich die Ressourcen.
Parsing	Zerlegung und Umwandlung einer beliebigen Eingabe in ein für die Weiterverarbeitung brauchbares Format. [vgl. http://de.wikipedia.org/wiki/Parsing, Zugriff: 28.12.2008]
Progressive Scan Verfahren	Wiedergabe eines vollständigen Bildes (Frames) mittels eines Vollbildes, keine Halbbilder.
Realmedia	RealMedia ist die Sammelbezeichnung für die Dateiformate

	und die damit verbundenen Client- und Serverprodukte des Software-Herstellers RealNetworks. [vgl. http://de.wikipedia.org/wiki/Realmedia, Zugriff: 28.12.2008]
Rendering	Darstellung oder Berechnung von grafischen Inhalten [vgl. http://de.wikipedia.org/wiki/Rendering, Zugriff: 28.12.2008]
SSL	Transport Layer Security (TLS) oder Secure Sockets Layer (SSL) ist ein hybrides Verschlüsselungsprotokoll zur Datenübertragung im Internet. [vgl. http://de.wikipedia.org/wiki/Transport_Layer_Security, Zugriff: 28.12.2008]
String	Eine Zeichenkette oder ein String (englisch) ist eine Folge von Zeichen (z. B. Buchstaben, Ziffern, Sonderzeichen und Steuerzeichen) aus einem definierten Zeichensatz. [vgl. http://de.wikipedia.org/wiki/String, Zugriff: 28.12.2008]
TCP/IP Verbindung	Das Transmission Control Protocol/Internet Protocol (kurz TCP/IP) ist eine Familie von Netzwerkprotokollen. Die Identifizierung der am Netzwerk teilnehmenden Rechner geschieht über IP-Adressen. [vgl. http://de.wikipedia.org/wiki/TCP/IP, Zugriff: 28.12.2008]
Thumbnail	Als Vorschaubild, Bildvorschau, Miniaturbild oder Thumbnail (englisch für Daumennagel) werden kleine digitale Grafiken bzw. Bilder bezeichnet, die als Vorschau für eine größere Version dienen [vgl. http://de.wikipedia.org/wiki/Thumbnail, Zugriff: 28.12.2008]
Timer	Steuerbaustein, der zur Realisierung unterschiedlicher zeitbezogenen Funktionen eingesetzt wird. [vgl. http://de.wikipedia.org/wiki/Timer, Zugriff: 28.12.2008]
User Centered Design	User Centered Design (UCD) beschäftigt sich damit, wie Benutzer systematisch in die Entwickung von Systemen und Produkten einbezogen werden können. [VGL. RICHTER UND FLÜCKIGER, 2007, S. 10]
WMV	Windows Media File. Windows-eigenes Dateiformat für Video und Audio.
Wrapper	Eine Anwendung für Hüllenklassen (engl. Wrapper) in objektorientierten Programmiersprachen ist, Klassen für Grunddatentypen zur Verfügung zu stellen, um die Handhabung zu vereinfachen und zusätzliche Funktionen zur Verfügung zu stellen. [vgl. http://de.wikipedia.org/wiki/H%C3%BCllenklasse, Zugriff: 28.12.2008]
Zeilensprungverfahren	Wiedergabe eines vollständigen Bildes (Frame) mittels zweier Halbbilder die zeilenweise gewechselt werden. Die Trägheit des Auges setzt dies zu einem Vollbild zusammen.

IX ANHANG – SCREENSHOTS

```
/*----------------------------------------
    SCALE & POSITION VIDEO INSTANCES
----------------------------------------*/
            private function scalevideo(vidIndex:int, aspect:Number) {

                if ((aspect != 0) && (aspect >= 16/9))  {
                    if ((vidIndex == 1) || (vidIndex == 2) || (vidIndex == 3) || (vidIndex == 4)) {
                        root["vidStream"+vidIndex].width    = root["video_thumb"+vidIndex].width - 4;
                        root["vidStream"+vidIndex].height   = Math.floor(root["vidStream"+vidIndex].width / aspect);
                        root["vidStream"+vidIndex].x        = root["video_thumb"+vidIndex].x+1;
                        root["vidStream"+vidIndex].y        = root["video_thumb"+vidIndex].y+1;
                    } else if(vidIndex == 5) {
                        video_thumb5.vidStream5.width       = video_thumb5.width - 4;
                        video_thumb5.vidStream5.height      = Math.floor(video_thumb5.vidStream5.width / aspect);
                        video_thumb5.vidStream5.x           = 1;
                        video_thumb5.vidStream5.y           = 1;

                    }//Endif
                } else ...
            }//End private function scalevideo
```

Abbildung A-1: Quellcode der scalevideo()-Funktion. Skaliert die Videoinstanzen proportional zum Bildseitenverhältnis, das aus den Meta-Daten errechnet wird.

[Quelle: ControlCenter.as; Stand: 10.12.2008]

```
    info_container3.dialogCueRemove.btn_CancelCueRemove.addEventListener    (MouseEvent.CLICK, dialogCueRemoveHandler);
    info_container3.dialogCueItemRemove.btn_ConfirmCueItemRemove.addEventListener(MouseEvent.CLICK, dialogCueRemoveHandler);
    info_container3.dialogCueItemRemove.btn_CancelCueItemRemove.addEventListener(MouseEvent.CLICK, dialogCueRemoveHandler);
    info_container3.btn_saveCueList.addEventListener(MouseEvent.CLICK, saveCueList);
//Add EventListeners to the Lists
    for (var a:Number = 1; a <= sAmount+1; a++){
        if (a != 5) root["btn_playStream"+a].addEventListener   (MouseEvent.CLICK, changeStream);
        if (a != 5) root["btn_Stop"+a].addEventListener         (MouseEvent.CLICK, stopPublishStream)
        root["video_thumb"+a].addEventListener                  (MouseEvent.CLICK, setAudio);
        root["video_thumb"+a].addEventListener                  (MouseEvent.MOUSE_OVER, vidFMouseOverHandler);
        root["video_thumb"+a].addEventListener                  (MouseEvent.MOUSE_OUT, vidFMouseOutHandler);
    }
    video_thumb5.btn_play5.addEventListener(MouseEvent.CLICK,changeStream);
//sidbar links
    info_container1.sidebar_link1.addEventListener (MouseEvent.CLICK, sidebarHandler);
    info_container2.sidebar_link2.addEventListener (MouseEvent.CLICK, sidebarHandler);
    info_container3.sidebar_link3.addEventListener (MouseEvent.CLICK, sidebarHandler);
//listbar links
```

Abbildung A-2: Auszug der EventListener-Deklaration.

[Quelle: ControlCenter.as; Stand: 10.12.2008]

```
//////////////////////////////////////////////
er data obtained from the Flash Media Server
//////////////////////////////////////////////

//////////////////////////////////////
 Get List of Videofiles in Streamfolder
//////////////////////////////////////
      function dirResults(retVal) {
          //
          tab1.dg_StreamList.removeAll();                    //Initialize DataGrid
          liveStreamArray.length=0;                          //Initialize Array

          for (var i = 0; i<retVal.length; i++) {
              //Takes FLV link out of File Object that is returned.
              var flv_name = (retVal[i].name).substr((retVal[i].name).lastIndexOf("/")+1);    //takes the "/approot/" out of the retVal
              //Sets the array to your FLV
              var index = flv_name.lastIndexOf(".");
              var stream_name = flv_name.substring(index + 1, flv_name.length) + ":" + folderName + "/" + flv_name.substring(0, index);
              //Gathers the length for each of the streams
              var streamlength = nc.call("getStreamLength", null, stream_name);
              var streamsize = (retVal[i].length)/(Math.pow(2,20));   //Convert Byte to MByte
              //Adds each stream information to liveStreamArray which is then put into datagrid
              liveStreamArray.push({Name:flv_name, Live: false, Length:retVal[i].streamlength, CreationTime:retVal[i].creationTime, Size:streamsize.toFixed(2)});
          }
          //Invoke getLiveStreams() function after finished gathering
          if (APIconnect == false) {addColumnStreamList();}   //If GetLiveStreams failed, add only the FileObject List to DataGrid
          else {APIconnect=true;getLiveStreams();}            //Get initial LiveStream List from Server API
      }
      function dirFaults(errorVal) {
          trace("Error : "+folderName+" "+errorVal.code+"\n");
          //If FileObject fails get only the LiveStreams
          getLiveStreams();                                  //Get initial LiveStream List from Server API
      }
```

Abbildung A-3: Quellcode der dirResults()-Funktion. Verarbeitet die Rückgabewerte der Auflistung der gespeicherten Videodateien.

[Quelle: ControlCenter.as; Stand: 10.12.2008]

```
////////////////////////////////////
/ Get List of Files in temp Folder
////////////////////////////////////
      function tempDirResults(retVal) {
          //
          //tab4.list_ThumbList.setStyle("cellRenderer", CCThumb);    //Set CellRenderer For Thumblist
          var tempFilesArrayOld:Array = new Array();
          if (tempFilesArray.length > 0) {
              //fill compare array with old items
              for (var f:int = 0; f < tempFilesArray.length; f++) {
                  tempFilesArrayOld.push(tempFilesArray[f]);
              }
              tempFilesArray.length = 0;          //initialize tempFilesArray
          }
          FinishedFilesArray.length = 0;      //initialize finishedFilesArray
          for (var i = 0; i<retVal.length; i++) {
              //takes the "/approot/temp/" out of the retVal to resolve the pure filename
              var temp_name = (retVal[i].name).substr((retVal[i].name).lastIndexOf("/")+1);
              var compValue = compareTempItem(retVal[i], tempFilesArrayOld);
              //Sets the arrays of the current files in the studio/temp folder
              //Adds each item to the proper array depending on the compare Value
              if (compValue != true) tempFilesArray.push({name:temp_name, source:appRootAbs+thumbFol+temp_name, size:retVal[i].length, lastmod:retVal[i].lastMo
              else FinishedFilesArray.push({name:temp_name, source:appRootAbs+thumbFol+temp_name, size:retVal[i].length, lastmod:retVal[i].lastModified});
          }
          //if condition ensures unique invoke of the timer event
          if (compareTimerFlag == true) {
              var TempCompareTimer:Timer = new Timer(1000, 1);
              TempCompareTimer.addEventListener(TimerEvent.TIMER, reCompare);
              TempCompareTimer.start();
          } else {
              compareTimerFlag = true;    //Re-Enable the timer for next run
              addColumnTempList();        //Add results to the proper DataGrid
          }//endif
      }
      function tempDirFaults(errorVal) {
          trace("Error : "+tempfolderName+" "+errorVal.code+"\n");
      }
      function compareTempItem(actTempItem, oldArray) {
          var actTempItem_name = (actTempItem.name).substr((actTempItem.name).lastIndexOf("/")+1);
          var compFlag:Boolean = false;
          for each(var oldItem in oldArray) {
              if (oldItem.name == actTempItem_name) {
                  if (oldItem.size == actTempItem.length) {
                      compFlag = true;
                  } else compFlag = false; //endif2
              }//endif1
          }//endfor
          if (compFlag == true) return true;
          else return false;
      }
      private function reCompare(e:TimerEvent):void {
          createTempFileObj();
          tempDir();
          compareTimerFlag = false;   //Disable the timer
      }
```

Abbildung A-4: Quellcode der tempDirResults()-Funktion. Ergebnisverarbeitung der Liste der temporären Streams.

[Quelle: ControlCenter.as; Stand: 10.12.2008]

```
//////////////////////////////////////
// Get List of Streams from Server API
//////////////////////////////////////
        public function getLiveStreams() {
            liveStreamNameArray.length=0;
            nc_admin.call("getLiveStreams", new Responder(resultHandler, faultHandler), appName+"/"+streamFolder);
            function resultHandler(info){
                var retValLive = info.data;
                if (info.code == "NetConnection.Call.Success") {
                    for (var i = 0; i < retValLive.length; i++) {
                        var liveStreamName = retValLive[i];
                        //ADD ITEM, BUT NO DUPLICATES
                        //duplicate control (avoids gathering vod streams which are
                        //actually played from the FileObject Streamlist. getLiveStreams issue,
                        //which returns !every! running stream !)
                            var duplicate_found = false;
                            for(var f=0; f<liveStreamArray.length;f++) {
                                //Check if extension exists
                                //get Names of FileObject Array WITHOUT fileextension
                                var fObjName:String;
                                if (liveStreamArray[f].Live != true) fObjName = (liveStreamArray[f].Name).substr(0, liveStreamArray[f].Name.length - 4);
                                else fObjName = liveStreamArray[f].Name;

                                if(fObjName == liveStreamName){
                                    duplicate_found = true;
                                }//endif
                            }//endfor
                            if(duplicate_found == false){
                                liveStreamNameArray.push(liveStreamName);
                            }//endif duplicate
                        //END ADD ITEM
                    }//end for1
                    //Check gathered LiveStreams for publishing status
                    getPublishStatus(liveStreamNameArray);

                }//end if1
            }//End Function resultHandler

            function faultHandler(fault){
                trace("fault: "+fault)
            }//End Function faultHandler

        }//End Function getLiveStreams
```

Abbildung A-5: Quellcode der getLiveStreams()-Funktion. Ergebnisverarbeitung der Liste der Live-Streams von der Administration-API.

[Quelle: ControlCenter.as; Stand: 10.12.2008]

```
public function rendererCCThumb() {
    super();

    loader.scaleContent = true;     //Resize Content of the Loader Component
    loader.maintainAspectRatio = true;
    useHandCursor = false;
...

    desc = new TextField();
    desc.autoSize = TextFieldAutoSize.LEFT;
    desc.x = 80;
    desc.y = 20;
    desc.width = (TileListWidth/3)-thumbWidth-20;
    desc.multiline = true;
    desc.wordWrap = true;
    addChild(desc);


    textStyle = new TextFormat();
    textSty...

    timeStyle = new TextFormat();
    timeSty...
}//end public function rendererCCThumb

override protected function drawLayout():void {
    // Position cell elements; tweak these for your thumbs if needed
    var imagePadding:Number = getStyleValue("imagePadding") as Number;
    loader.move(5, 1);

    var w:Number = width-(imagePadding*2);
    var h:Number = height-imagePadding*2;


    if (loader.width != w && loader.height != h) {
        loader.setSize(thumbWidth,h); //sets size of the loader content, not the size of the UILoader components itself!
    }

    loader.drawNow();                           //Draws the image in the Cell

    desc.text = data.name + "\n";               //Show Filename
    desc.setTextFormat(textStyle);

    time.text = "Last Modified: " + data.lastmod;
    time.setTextFormat(timeStyle);

    background.width = (TileListWidth/3);       //Set width of the cells as the third part of the whole TileList width,
    background.height = height;                 //to get a 3 column splitting

    textField.visible = false;
}//end override protected function
```

Abbildung A-6: Auszug des Quellcode der CellRenderer-Klasse rendererCCThumb.as. Hier wird festgelegt, wie der Inhalt der Thumbnails-TileList im Back-End angezeigt wird.

[Quelle: rendererCCThumb.as; Stand: 10.12.2008]

```
public function initStreamList():void {

    if ((streamXML) && (streamXML.length != 0)) streamXML.length=0; //Initialize XML Object
    //Fill Array with server-side Items
    var streamXML:XML = xmlLoader;
    //streamXML.ignoreWhitespace = true;

    var count:int =0;
    var livelist:Array = new Array();
    var storedlist:Array = new Array();
    //Fill Selectable List with XML Item Titles
    for each(var item:XML in streamXML.listitem){
            var actname:String = item.@name.toXMLString();
            // Get thumbnail value and assign to cellrenderer.
            var thumb:String;
            var dur = formatTime(Math.floor(item.duration));
            var timeStamp:Date = new Date(item.@time.toXMLString());
            //resolve date format in shorter string
            var tHour = timeStamp.getHours(); var tMinutes =  timeStamp.getMinutes(); //var tSeconds = timeStamp.getSeconds();
            var myDate = timeStamp.getDay() + "."+timeStamp.getMonth() + "." + timeStamp.getFullYear();
            var timeString:String = tHour+":"+tMinutes+", "+myDate;
            //if(item.hasOwnProperty("@thumb")>0)
            thumb = item.thumbnail.@thumbfolder + item.thumbnail.@thumb;
            // Send data to livelist or storedlist, depending on livestatus node.
            if (item.livestatus == "Yes") livelist.push({label:actname, data:item.stream.@source.toXMLString(),duration:dur, created:timeString,
            else storedlist.push({label:actname, data:item.stream.@source.toXMLString(),duration:dur, created:timeString, source:thumb, appurl:it
    }//endfor
    livelist.sortOn("label");
    //Remove last Item: Issue from ControlCenter XML, empty "initial" item
    for (var z=0; z < livelist.length;z++) {
        if (livelist[z].label == "initial_title") livelist.splice(z,1);
    }
    //send the array of videos to the listbox UI
    if (list_container1.list_StreamList.length !=0) list_container1.list_StreamList.removeAll();
    list_container1.list_StreamList.dataProvider = new DataProvider(livelist);

    if (list_container2.list_StreamList.length !=0) list_container2.list_StreamList.removeAll();
    list_container2.list_StreamList.dataProvider = new DataProvider(storedlist);

    root["list_container"+activelist].refreshLoader.visible = false;
}//End public function initStreamList
```

Abbildung A-7: Quellcode der initStreamList()-Funktion. Verarbeitung der Werte des Server-side Arrays.

[Quelle: UserInterafce.as; Stand: 10.12.2008]

```
- <xml>
  - <cueitem name="backcountry_bombshells_4min_HD_1500_96.flv" url="rtmp://localhost/DIPL_streamwork/studio">
      <cuepoint time="0" offset="0">Start</cuepoint>
      <cuepoint time="33" offset="0">O-Ton1</cuepoint>
      <cuepoint time="42" offset="0">O-Ton2</cuepoint>
      <cuepoint time="60" offset="0">Tricks</cuepoint>
      <cuepoint time="136" offset="0">Lawine</cuepoint>
    </cueitem>
  - <cueitem name="olympia_tt_high.mp4" url="rtmp://localhost/DIPL_streamwork/studio">
      <cuepoint time="0" offset="0">Start</cuepoint>
      <cuepoint time="0" offset="10">CuePoint</cuepoint>
    </cueitem>
  - <cueitem name="bbc-r_m720p.flv" url="rtmp://localhost/DIPL_streamwork/studio">
      <cuepoint time="0" offset="0">Start</cuepoint>
      <cuepoint time="0" offset="0">CuePoint</cuepoint>
      <cuepoint time="0" offset="0">CuePoint</cuepoint>
    </cueitem>
  - <cueitem name="bikes.flv" url="rtmp://localhost/DIPL_streamwork/studio">
      <cuepoint time="0" offset="0">Start</cuepoint>
    </cueitem>
  - <cueitem name="livestream_file.flv" url="rtmp://localhost/DIPL_streamwork/studio">
      <cuepoint time="0" offset="0">Start</cuepoint>
      <cuepoint time="0" offset="0">Test1</cuepoint>
      <cuepoint time="0" offset="0">Test2</cuepoint>
    </cueitem>
  </xml>
```

Abbildung A-8: Aufbau der cuelist.xml mit exemplarischen Inhalten.

[Quelle: cuelist.xml; Stand: 10.12.2008]

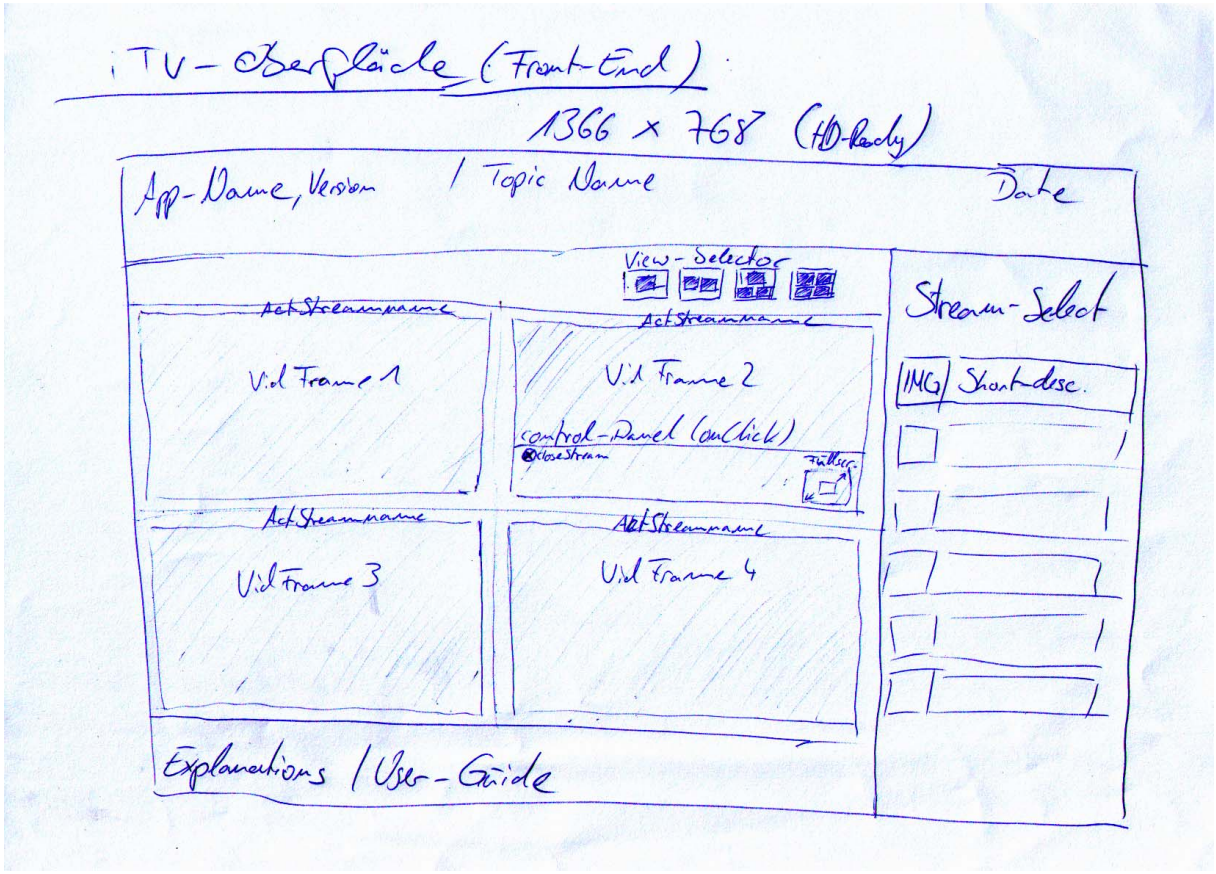


Abbildung A-9: Skizze des Front-End User Interfaces.

[Quelle: eigener Entwurf]

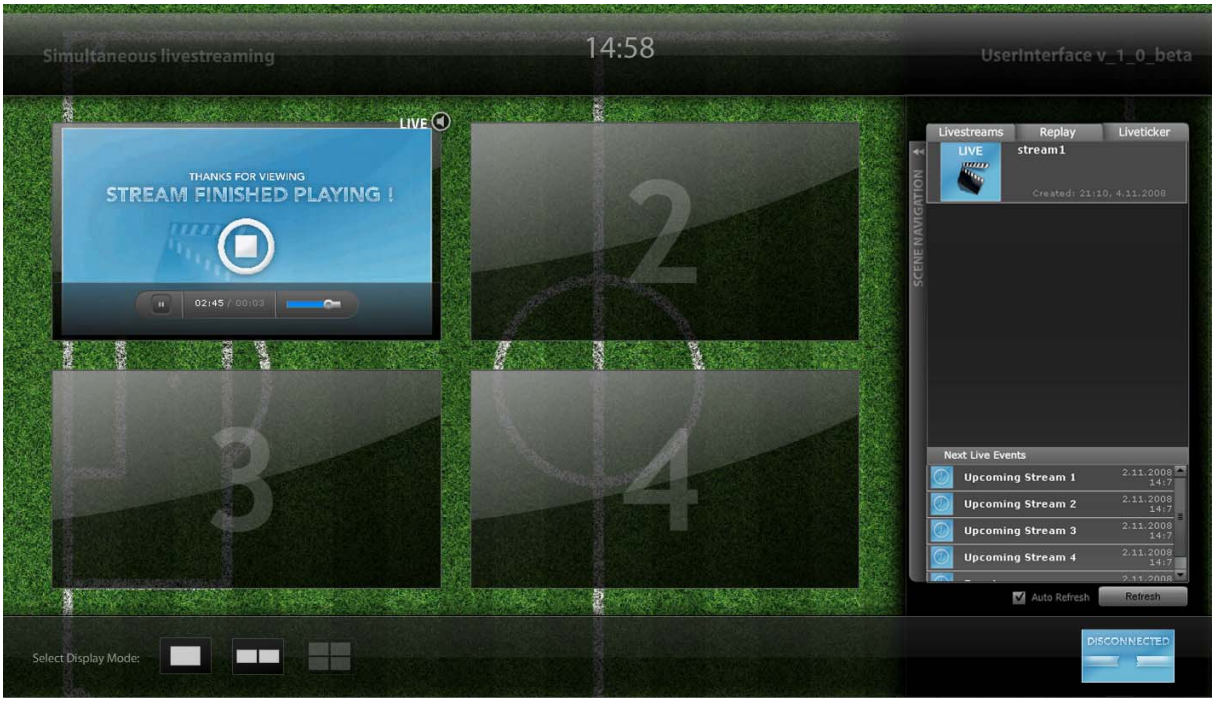


Abbildung A-10: Umsetzung des Front-End User Interfaces.

[Quelle: UserInterface.fla, Stand: 10.12.2008]

Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst habe, dass ich keine anderen Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, in jedem Fall als Entlehnung kenntlich gemacht habe. Das Gleiche gilt auch für beigegebene Zeichnungen und Abbildungen.

Meckenheim, 2. Januar 2009

Voker Lux