Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

Konzeption, Implementation und Erprobung eines multimedialen streamingfähigen Dateisystems für das Internet

Diplomarbeit

Leipzig, 25.08.2003

vorgelegt von
Ulrich Hanff
geb. am:
11.09.1978
Studiengang Informatik

Inhaltsverzeichnis

I. Einleitung

1. Zielstellung

In der vorliegenden Arbeit soll ein offenes System zur Verwaltung, Archivierung und Übertragung von multimedialen Daten konzipiert, implementiert und erprobt werden, das in der Lage ist Streaming-Übertragungen zu realisieren. Das System soll im Sinne eines Dateisystems oder Content-Management-Systems Dateien sowie multimediale Inhalte über das Internet bereitstellen.
Dazu müssen entwickelt werden:

- ein Hash-Algorithmus zur Identifizierung der Daten,
- eine offene Schnittstelle für Frontend-Applikationen,
- ein Lizenz-, Berechtigungs- und Abrechnungssystem,
- Clients zum Bereitstellen und Abspielen von Streams,
- eine Web-Oberfläche zum Benutzen des Systems.

Zielstellungen für die Konzeption und Entwicklung sind:

- Flexible Einsetzbarkeit,
- Entwicklung offener Schnittstellen,
- Verwendung offener Standards,
- Erweiterbarkeit.

Das System soll nach der prototypischen Implementation erprobt werden. Dabei sollen auch Leistungsparameter gemessen und für die Weiterentwicklung des Systems evaluiert werden.

2. Motivation

Die Welt der Informationstechnik in den neunziger Jahren des zwanzigsten Jahrhunderts wurde vorwiegend durch 3 Begriffe geprägt: Internet, Multimedia und Mobilität. Gegen Ende des zwanzigsten Jahrhunderts wurden die Computer klein und leistungsfähig genug, um den Anforderungen der Mobilität und Multimediafähigkeit zu entsprechen. Mit dieser Möglichkeit wurde der Softwaremarkt mit immer mehr Software zum Bearbeiten, Abspielen und zum Übertragen von Audio und Video

überströmt. Es wurden viele Verfahren zum Kodieren und Komprimieren von Video und Audio für die unterschiedlichsten Einsatzgebiete entwickelt. Es wurden Verfahren entwickelt, Video und Ton per Kamera und Mikrophon aufzuzeichnen und über das Internet im Sinne der Telefonie zu übertragen. Andere Ideen sahen vor, Konferenzen oder Tagungen im Internet zu veröffentlichen und zugänglich zu machen. Diese Ideen hatten zur Folge, dass sich einige leistungsfähige Firmen, die in der Lage waren, Ideen schnell umzusetzen, heraus kristallisierten und umfangreiche Softwarepakete und Verfahren zum Komprimieren und Kodieren von Audio und Video entwickelten. Diese Firmen sind zum Beispiel Real Networks, Apple und Microsoft. Sie haben den Massenmarkt entdeckt und nutzen diesen zum Etablieren ihrer Standards in der multimedialen Welt. Sollen heute multimediale Daten, zum Beispiel einen Radiostream (Audiodatenstrom für Internetradio), im Internet veröffentlicht oder über dieses übertragen werden, so gibt es kaum einen Weg, wenn viele Menschen damit erreicht werden sollen, dieses ohne den Einsatz der Software einer dieser Firmen zu realisieren.

Mit Anfang des 21. Jahrhunderts wurde das Internet immer mehr zu einem "Medium für Alles", das überall zu finden ist. Die Kosten der Internetnutzung sanken in Bereiche, die sich nahezu jeder Mensch (in einem relativ wohlhabenden Industrieland wie Deutschland) leisten kann. Diese Entwicklung, die zu dem Zeitpunkt des Entstehens dieser Arbeit, immer noch anhält, hat zur Folge, dass immer öfter nicht mehr vom Industriezeitalter sondern vom Informationszeitalter gesprochen wird. Informationszeitalter bedeutet, dass Informationen und Daten fast ohne Zeitverlust (weniger als 1 Sekunde Verzögerung) an fast jeden Ort, der über entsprechende Empfangs- und Übertragungsmechanismen verfügt, übertragen werden können. Eine Schlussfolgerung daraus ist, dass Daten und Informationen von jedem Ort, sofern dieser ebenfalls über entsprechende Empfangs- und Übertragungsmechanismen verfügt, aus abgerufen werden können (Nutzermobilität). Möglich wurde diese Entwicklung durch die starke Verbreitung von Mobilfunk und den immer leistungsfähigeren Mobilfunkendgeräten.

Eine weitere Entwicklung der letzten Jahre ist die OpenSource-Community, einer weltweiten Gemeinde von Softwareentwicklern, die gemeinsam an Projekten arbeiten, um sie zum Wohle und Nutzen Aller mit Anderen zu teilen. Die Philosophie hinter dieser Bewegung soll hier nicht hinterfragt oder überhaupt betrachtet werden, jedoch zeigt diese Gemeinde die Fähigkeit, mit der immer schneller werdenden

Entwicklung von Soft- und Hardware und der immer größer werdenden Informations- und Wissensflut, schritt halten zu können. So zeigt sich auch, dass einige wichtige Standardisierungen, wie zum Beispiel das HyperText Transfer Protocol (HTTP), der letzten Jahre meist aus nicht kommerziellen Projekten und Bewegungen entstanden. Auch im multimedialen Bereich haben sich solche „Quasi-Standards“ entwickelt, wie zum Beispiel das Videoformat DIVX, welches sich ursprünglich aus dem MPEG IV Standard entwickelte.

Es gibt Software wie Microsofts Netmeeting oder Microsofts MSN Messenger, mit denen Videotelefonie betrieben werden kann. Jedoch fehlt allen Anbietern von Videotelefoniesoftware die Flexibilität alle Video- und Audioformate (im Sinne von Formatunabhängigkeit) übertragen zu können und sich an bestimmte Benutzerwünsche anpassen zu können – so ist es zum Beispiel nicht möglich im Microsoft MSN Messenger das Videobild zu skalieren oder zu vergrößern, es wird lediglich ein nur Briefmarken großes Bild angezeigt. Aus diesem Grund wurde im Rahmen von [BEHA_2001] mit der Hilfe von Java und dem Java Media Framework eine Videokommunikationssoftware entwickelt, mit der unter anderem verschiedene Audio- und Videoformate übertragen werden können und das Videobild frei skaliert werden kann. Dabei kamen Formate wie H.263, MJPEG und DIVX zum Einsatz. Als Übertragungsprotokoll wurde das Realtime Protocol (RTP) eingesetzt. Bei der Erprobung der Kommunikation zwischen einer Firma und dem öffentlichen Netz gab es jedoch Probleme mit der Auslieferung der UDP-Pakete, da diese ohne größeren Aufwand die Rechner in Subnetzen aufgrund einer fehlenden virtuellen Verbindung (wie bei TCP) nicht adressieren können. Es fehlt also ein Ansatz, der es ermöglicht, multimediale Daten aus jedem Netzwerk heraus und in jedes Netzwerk hinein zu übertragen. Mit „jedem Netzwerk“ sind Netzwerke gemeint, die Bandbreiten von mindestens 10 kbit/s besitzen.

Nun ist es zwar möglich mit heutigen Mitteln Videotelefonie und Video-On-Demand zu betreiben, jedoch existieren keine offenen und flexibel einsetzbare Lösungen. Es existieren lediglich Projekte und Produkte, die für den illegalen Tausch von Musik und Film missbraucht werden. Es gibt zwar Firmen wie Arcor, die Video-On-Demand-Plattform betreiben, allerdings sind diese Dienste bis zum heutigen Tage nur proprietär und meist inkompatibel zu Nicht-Microsoft Betriebssystemen. So verlangen die Video-On-Demand-Dienste von Arcor und T-Online-Vision den Microsoft Mediaplayer und entsprechende Digital Rights Management Mechanismen.

Die vorliegende Arbeit hat ein Dateisystem (nachfolgend System genannt) entwickelt, das es ermöglicht, multimediale Daten, fester Länge und live erzeugte Daten, zum Beispiel Videostreams, unabhängig vom ihrem Format zu übertragen, sie zu archivieren und sie zu verwalten.

Da schon Systeme existieren, die Ähnliches oder Teile realisieren, wurde kein weiteres System dieser Art entwickelt, sondern vielmehr eines, das offen und flexibel ist. Offen bedeutet dabei, dass die Quelltexte im Sinne des OpenSource frei verfügbar sind, so dass prinzipiell jeder in der Lage ist, das System zu benutzen, zu verbessern und auf der Grundlage der Quellcodes neue Schnittstellen und Clients zu entwickeln. Das System ist dabei so flexibel, dass es in sämtlichen Bereichen, sei es in der privaten Videotelefonie in der kommerziellen Musikindustrie oder in der betrieblichen Videoüberwachung, einsetzbar ist.

– Um im privaten Bereich bestehen zu können, muss es den Massenmarkt überzeugen können, darf den privaten Anwender also finanziell nicht belasten (dabei ist auch die benötigte Hardwareumgebung zu beachten).
– Um im kommerziellen Bereich Fuß fassen zu können, hat es Funktionen zum Lizensieren, zum Abrechnen und zum Vergeben gewisser Verwaltungsberechtigungen zur Verfügung stellen.
– Damit das System zum Beispiel von der Musikindustrie akzeptiert wird, muss es den Missbrauch von urheberrechtlich geschützten Daten verhindern können.
– Weiterhin besitzt es offene Schnittstellen, damit Benutzerschnittstellen im „Corporate Identity Look“ oder Ähnlichem geschaffen werden können.
– Ein weiterer Gesichtspunkt für den Einsatz im kommerziellen Bereich ist die Hochverfügbarkeit und Skalierbarkeit des Systems, es wurde deshalb mit Techniken und Methoden aus dem Hochverfügbarkeitsbereich implementiert.
– Um im betrieblichen Einsatz zu überzeugen, bietet das System Kompatibilität zu anderen Standards.
– Ein anderer wichtiger Punkt sind die Betriebskosten des Systems, also dem so genannten Total Cost of Ownership (TCO), der letztendlich festlegt, ob sich eine Software auch lohnt. Das System lässt sich deshalb leicht bedienen und administrieren und ist mit wenigen Handgriffen in andere Systeme integrierbar.

Des weiteren ist es möglich, mit dem entwickelten System Videokonferenzen über das Internet zu führen, Daten online zu verwalten, Daten mit anderen Menschen zu

teilen (im Sinne von Filesharing), Musikstücke zu verwalten und live anzuhören, Internetfernsehen zu realisieren und, im Sinne einer Videothek, gegen Bezahlung Filme herunterzuladen. Eine wichtige Eigenschaft des Systems, die die Einsetzbarkeit grundlegend bestimmt, ist die Plattformunabhängigkeit die dieses System selber und der Benutzerschnittstelle bietet. Das heißt, dass das System selber auf vielen unterschiedlichen Computer- und Betriebssystemen lauffähig ist und es möglich ist, weitere Clients für das System zu entwickeln, die auf jedem Computer- und Betriebssystem, das entsprechende Hardwareanforderungen erfüllt, einsetzbar sind. Alle diese Eigenschaften des Systems sollen es ermöglichen einen Standard für die Adressierung, Verwaltung und Übertragung von Daten im Internet zu etablieren, einen Standard, den jeder nutzen, weiterentwickeln und für seine Zwecke benutzen kann.

Im Kapitel 2 der vorliegenden Arbeit wird das grundlegende Konzept geklärt werden. Es wird darauf eingegangen werden, warum es als OpenSource entwickelt wurde, was die Eigenschaften eines streamingfähigen Dateisystems im Einzelnen sind und wie das System eine größtmögliche Flexibilität erreicht. Im dritten Kapitel klärt eine Anforderungsanalyse die Einsetzbarkeit vorhandener Lösungen und deren Fehler. Des weiteren werden dort grobe Parameter des Systems schon festgelegt. In der Spezifikation des Systems, im Kapitel 4, werden die Struktur des Systems, detaillierte Parameter und spezifische Workflows definiert. Unter anderem werden Streaming- und Datenaustauschprotokolle spezifiziert. Das Kapitel 5 klärt grundlegendes zur Referenzimplementation des spezifizierten Systems, dabei wird kurz auf Methoden und Werkzeuge, die bei der Implementation benutzt wurden, eingegangen. Das sechste Kapitel demonstriert mit Screenshots die Benutzbarkeit des Systems und klärt kritisch einige Fragen zur Leistungsfähigkeit und Einsetzbarkeit des Systems. Im siebten und letzten Kapitel folgt eine Zusammenfassung und der Ausblick für das System.

2. Konzeption

In diesem Abschnitt wird ein Modell für das in dieser Arbeit entwickelte System

konzeptionell dargestellt werden. Es wird geklärt, mit welchen Parametern und Anforderungen das Dateisystem entwickelt werden soll. Die Idee des streamingfähigen Dateisystems ist Folgende:

Es wird ein System entwickelt werden, mit dem Daten nicht auf einem lokalen Rechner, sondern im Sinne eines Portals, im Internet zentral gespeichert werden können. Diese „Online-Speicherung“ hat zum Vorteil, das im Falle eines lokalen Systemabsturzes noch alle Daten vorhanden sind. Ein weiterer Vorteil ist die hohe Mobilität der Daten, denn sie sind nicht mehr an ein lokales Rechnersystem gebunden. Diese Eigenschaft macht das System jedoch noch nicht einzigartig. Dieses System zusätzlich die Möglichkeit bieten auch Daten nicht bekannter Länge und Größe zu verwalten, so genannte Streams. Diese Streams können dabei Audio- und Videodaten sein, die durch ein Aufnahmegerät zur Laufzeit des Systems erst erzeugt werden. Es ist auch denkbar Steuerinformationen für Geräte über das System zu übertragen. Die Multimediafähigkeit des Dateisystems stellt dabei hohe Ansprüche an die Übertragungs- und Kodierungseffizienz des Systems.

Sehr wichtig ist die Eigenschaft der Erreichbarkeit des Systems. Es soll möglich sein, die Daten auf dem gleichen Wege wie eine Website im Internet zu erreichen. Das heißt also, dass ein zentralisiertes Konzept ermöglichen muss, auch in Netzen mit stark eingeschränkten Interaktionsmöglichkeiten, wie zum Beispiel in gesicherten Firmennetzwerken, die den Zugang zum Internet nur über Proxyserver erlauben, das System benutzen zu können.

Das in diesem Kapitel zu konzipierende System soll dabei nicht vordergründig neue Techniken und Protokolle entwickeln, sondern aufbauend auf bewährten Techniken und Protokollen eine flexible, frei verfügbare und effiziente Methode zur Verwaltung von Daten im Internet spezifizieren, die die oben genannten Eigenschaften realisieren kann. Dabei soll von Anfang an zwischen Spezifikation und Implementation unterschieden werden. In den folgenden Kapiteln wird dieses zu entwickelnden Konzept als System bezeichnet werden, da es sowohl Spezifikation als auch Referenzimplementation beeinhaltet. Im Abschnitt 2.1 wird geklärt, welche Eigenschaften einer Softwareentwicklung wichtig für eine starke Verbreitung und hohe Qualität sind. Im Abschnitt 2.2 werden die Parameter des Dateisystems grob skizziert, Details dazu werden erst nach einer Anforderungsanalyse (Kapitel 3) in der Spezifikation (Kapitel 4) definiert. Abschnitt 2.3 legt ein grundlegendes Vorgehen bei der Entwicklung fest, um eine größtmögliche Flexibilität zu erreichen.

2.1. Entwicklung als OpenSource Projekt

Dieser Abschnitt soll einige Gründe der Offenheit des Systems untersuchen und diskutieren.

In den 70er Jahren begann in der Welt der Informations- und Computertechnik die Entwicklung eins Konzepts, das heute OpenSource genannt wird. Nach [FREYERMUTH_2001] war es der Angebotsmangel, der in dieser Zeit Forscher und Studenten zur Entwicklung des Time-Sharing-Verfahrens für terminalbasiertes Arbeiten an den Mainframes und somit zu den ersten Schritten der OpenSource Bewegung führte. Es heißt dort auch weiter, dass nichts den technischen und ökonomischen Fortschritt eines Produktes so sehr beschleunigt, wie die Einbeziehung derer, die mit dem Produkt arbeiten und leben müssen. Tatsächlich ist es auch so, dass die großen und wichtigen Konzepte der letzten Jahre und Jahrzehnte, die zum Beispiel auch das Internet am Leben erhalten, fast ausschließlich durch die OpenSource Gemeinschaft entwickelt wurde. Als Beispiel sei das große Serverbetriebssystem Unix genannt, dass als Weiterentwicklung des aufgegebenen Multix Betriebssystems durch die Entwickler Ken Thompson und Dennis Ritchie entstand. Ein weiterer Eckpfeiler des Internets ist das offene Protokoll TCP/IP. Weitere Beispiele für OpenSource sind SMTP, HTTP, BIND, der mit über 60% (nach [NETCRAFT_2003]) am häufigsten eingesetzte Webserver Apache, die im Internet in den 90er Jahren fast ausschließlich eingesetzte Scriptsprache Perl und noch viele mehr. Ein wichtiger Punkt, der den Erfolg dieser OpenSource Entwicklungen erst möglich machte, ist die Fähigkeit der verteilten Kooperation in den Entwicklerteams, die aus Entwicklern bestehen, die sich meist noch nie gesehen haben.

Eine nicht zu vernachlässigende Eigenschaft von OpenSource, ist der Kostenfaktor. So ist es nach [CTOS_2003] die OpenSource Software, die den Total Cost of Ownership (TCO) besonders stark in großen (mehr als 50 Mitarbeiter) aber immer noch maßgeblich in kleineren Unternehmen reduziert. Das liegt nicht nur an den fehlenden Lizenzkosten, denn kostenlose Software heißt nicht zwangsläufig, dass der Unterhalt und die Administration kostenlos oder kostengünstiger ist als bei kostenpflichtigen Konkurrenzprodukten. Vielmehr kann die schnelle Anpassbarkeit an

zum Beispiel entdeckte Sicherheitslücken oder die Anpassbarkeit an individuelle Bedürfnisse überzeugen. Das ist auch in den meisten Fällen einer der größten Nachteile bei OpenSource Software. In der Regel wird viel Know How benötigt, um OpenSource Software anzupassen und „up to date“ zu halten. Für eine kleine Firma (unter 50 Mitarbeiter) heißt das, dass extra Kosten für zum Beispiel einen Linuxadministrator anfallen, um die kostengünstigeren Linux-Server am Leben zu erhalten.
In [MASCHU_1996] wird für die fortschreitende Globalisierung voraus gesagt, dass immer mehr Menschen arbeitslos sein werden, dafür aber Produkte und Dienstleistungen im Zuge der Überproduktion immer billiger werden. Diese Entwicklung, in den OpenSource Kontext gerückt, bedeutet, dass als Folge günstigerer Software viele Unternehmen auch im materiellen Produktionssektor wesentlich kostengünstiger produzieren können.
Ein für diese Arbeit wichtiger Vorteil von OpenSource ist die Möglichkeit der offenen Standardisierung, das heißt, dass es durch die Entwicklung im OpenSource Umfeld möglich ist, dem Konzept zu einer Standardisierung zu verhelfen. Es gibt viele Beispiele für OpenSource Software, die mit dieser Strategie innerhalb weniger Jahre zum Quasi-Standard avancierten. So sei zum Beispiel das Projekt JBOSS (siehe Kapitel 5) genannt, einem Java Applicationserver, der sich in der E-Commerce Welt fast schon zum Standard für Webapplikationen entwickelt hat.

2.2. Konzeption des streamingfähigen Dateisystems

In diesem Abschnitt soll das Fundament des Systems, das Dateisystem konzipiert werden.
Da das System einfach zu bedienen und auch in andere Systeme integrierbar sein muss, bietet sich die Form des Dateisystems an. In herkömmlichen Dateisystemen kann jedes Datum mehrere Zuordnungen (Ordner) besitzen. Dieses System wird für jedes Datum im System eine eineindeutige Zuordnung haben. Die Daten sollen im Internet gespeichert werden, also für den Nutzer völlig unsichtbar auf einem Server, der wiederum ein eigenes Dateisystem hat. Nun kann es vorkommen, dass zwei Nutzer des Systems die gleichen Daten speichern wollen. Das System muss also entweder die Datei zweimal im eigenen Dateisystem abspeichern oder eine

Möglichkeit finden, die Datei nur einmal zu speichern und einem Besitzer eindeutig zuzuordnen. In diesem System werden die Dateien nur genau einmal gespeichert. Dabei ergibt sich von Anfang an das Problem, dass das System erst einmal erkennen muss, was gleichartige Daten sind. Ein kontrovers diskutiertes Filesharing-System mit dem Namen „eDonkey“ [EDONKEY_2003] benutzt für die Identifizierung der Dateien im verteilten Netzwerk so genannte Hashes. Das sind Werte, die aus den Daten berechnet werden, die in der Datei enthalten sind. Diese Berechnung von Hashes scheint also primär geeignet, Daten zu identifizieren, da sie die Daten der Datei repräsentiert. Für ein großes und skalierbares System muss diese Berechnung sehr effizient sein. Für den kommerziellen Anspruch sollte die Identifikation der Daten, also zum Beispiel von Musikstücken, gegenüber geringfügiger Verfälschung der Daten, zum Beispiel das Abschneiden der letzten zwei Sekunden bei einem Musikstück, tolerant sein.

Der nächste Problem ist die Klärung der Frage nach dem Besitzer der Datei. Angenommen zwei Nutzer besitzen eine Datei, hier sei wieder einmal das Beispiel des Musikstücks genommen. Dann sei der erste Nutzer zum Beispiel die Plattenfirma, der die Rechte an dieser Datei gehören und der zweite Nutzer ein Mensch, der diese Datei legal erworben hat. Die Frage ist nun, ob die Datei beiden Nutzern gehört oder Demjenigen, der sie dem System zuerst zur Verfügung stellt (nachfolgend „hochladen“). Für den Fall, dass ein Benutzer die Datei illegal erworben hat, ist die erste Möglichkeit, also dass beide Parteien die Datei besitzen, ungeeignet. Es muss also eine eindeutige Besitzregelung für Dateien im System geben. Das bedeutet wiederum, dass im System kein Nutzer, der Dateien verwalten, also auch hochladen will, anonym sein kann (und darf). Es muss also eine Nutzerverwaltung im Dateisystem implementiert sein. Da fast jedes Dateisystem eine Nutzer- und Gruppenverwaltung hat, wird aus Kompatibilitätsgründen auch eine Gruppenverwaltung implementiert.

Ein weiteres Problem ergibt sich bei der eindeutigen Speicherung der Dateien, es kann also im herkömmlichen Sinne eines Dateisystems keine Ordnerstrukturen geben, da die Dateien im System nicht vervielfältigt werden können, bzw. vervielfältigt gespeichert werden können. Wenn ein Nutzer eine Datei besitzt, dann darf ein anderer Nutzer diese Datei nicht hochladen. Diese Einschränkung sichert geistigen Eigentümern von Dateien (im Sinne des Urheberrechts), dass niemand die Daten missbrauchen kann. Es muss dafür eine Berechtigungsfunktion implementiert

werden, die es ermöglicht, anderen Nutzer die Benutzung einer Datei zu gestatten. Diese Berechtigungsfunktion stellt eine Art Kopierfunktion für das System dar. Wenn ein Besitzer einer Datei diese einem anderen Nutzer zur Verfügung stellen will, müsste er im herkömmlichen Sinne diese kopieren und zum Beispiel per Email versenden. Im zu entwickelnden System teilt der Besitzer dem anderen Nutzer nur eine Berechtigung an der Datei zu. Dadurch wird sichergestellt, dass die Datei nur einmal im System vorhanden ist. Die Berechtigungen besitzen Funktionen zum Einschränken des Gebrauchs, also zum Beispiel zeitliche Einschränkungen des Gebrauchs, aber auch Bevollmächtigungen, wie zum Beispiel das Erteilen von Administrationsrechten an einen Mitarbeiter eines Video-On-Demand-Dienstes, der dann für eine Datei Berechtigungen verteilen kann ohne Besitzer der Datei zu sein.
Die nächste zentrale Frage ist die Adressierung der Dateien. Da das System sowohl Dateien fester Länge als auch Daten unbekannter Länge, die also zur Laufzeit noch erzeugt werden (Streams) verwalten muss, wird ein abstraktes System implementiert, das die Unterschiede dieser Daten vor dem Nutzer verbirgt. Um nicht neue Auslieferungs- und Transportprotokolle entwickeln zu müssen, empfiehlt es sich auf verbreitete Standards zurückzugreifen. Für den Transport von Daten fester Länge ist das Hypertext Transfer Protokoll (nachfolgend HTTP) geeignet . Auf den Transport von Streams wird zu einem späteren Zeitpunkt noch ausführlich eingegangen werden. Wenn also schon das HTTP als Transportprotokoll genutzt wird, so liegt es doch nahe auch das URL-basierte Adressierungskonzept aus dem Internet als dem System zugrunde liegendes Adressierungskonzept zu benutzen. Die Frage, die sich dann stellt ist: Was soll eigentlich adressiert werden, die Daten oder deren Berechtigungen? Da das System die Verwaltung der Daten übernehmen soll und der Nutzer aus Sicherheitsgründen keinen Einfluss auf wichtige Verwaltungsfunktionen haben sollte (auch der Besitzer einer Datei) , wird die Adressierung der Berechtigungen verwendet. Somit können als Nebeneffekt auch gleich die Unterschiede zwischen Daten fester Länge und Streams verborgen werden und es wird auch noch eine einfache Integration in vorhandene Systeme ermöglicht. Da ein beliebiger Client allerdings noch Informationen über die Beschaffenheit der Daten haben muss, ist es nötig, dass eine Berechtigung diese Informationen zur Verfügung stellt.
Zusammengefasst verwaltet das System für jeden Nutzer eine unstrukturierte Liste von Berechtigungen, die er einzeln je über eine URL adressieren und bearbeiten

kann. Des weiteren stellt das System eine Downloadfunktion für Daten fester Länger und eine Streamfunktion für Daten variabler Länge zur Verfügung. Für andere Clients stellt das System eine Schnittstelle in XML zur Verfügung. Diese Schnittstelle enthält dann alle nötigen Informationen einer Berechtigung und alle Informationen die Berechtigung zu bearbeiten und die Daten zu beziehen. Jeder Nutzer ist in der Lage anderen Nutzern Berechtigungen an Dateien auszusprechen, bei denen er selber die Berechtigung hat, Berechtigungen auszusprechen.

2.3. Flexibilität des Systems

Eine Anforderung an das System ist die Flexibilität, es muss also ein großer Funktionsumfang mit möglichst wenig Einschränkungen realisiert werden. Das Ziel ist es, dass jeder Nutzer, der Daten verwalten möchte, auf dieses System prinzipiell zurückgreifen kann. Das bedeutet also, dass das System eigentlich alles können muss, was alle anderen Datenverwaltungssysteme zusammen können. Das scheint etwas übertrieben zu klingen, da jedoch alle Datenverwaltungssysteme ein Basissystem zur Speicherung (Serialisierung) ihrer Daten benutzen, also ein Dateisystem, und das zu entwickelnde System zusätzlich noch Streams verwalten können soll, ist diese Anforderung nicht zu hoch, sondern eine Grundvoraussetzung für die Entwicklung des Systems. Wenn das System allerdings so definiert wird, stellt sich schnell die Frage, ob nicht ein herkömmliches Dateisystem oder eine Datenbank zum Einsatz kommen kann. Weitere Möglichkeiten wären mit dem Einsatz eines versionisiertes Dateisystem wie etwa CVS [CVS_2003] oder eines redaktionellen Content-Management-Systems gegeben. Da sicher nicht alle Funktionen in einem System realisiert werden können, liegt es nahe, eine Grundlage für all diese Systeme zu schaffen. Dabei stellt das zu entwickelnde System ein völlig neuartiges Konzept zur Verwaltung, Übertragung und Adressierung von Daten beliebiger Art dar. Im Grunde genommen bietet das World Wide Web schon ähnliche Möglichkeiten, deswegen werden auch viele Konzepte übernommen (siehe Kapitel 3.1). Dieses System stellt ein API zur Verwaltung, Übertragung und Adressierung von Daten zur Verfügung. Um jedoch eine grundlegende Benutzbarkeit des Basissystems zu gewährleisten, wird es eine atomare Benutzerschnittstelle implementieren.
Die Grundlage für die Flexibilität des Systems liegt zum Einen in der Bereitstellung

von ausführlich dokumentierten und einfach zu benutzenden Schnittstellen, zum Anderen aber auch im Angebot von vordefinierten Workflows, die sich ihrerseits wieder Benutzer spezifisch anpassen lassen müssen. Als zentrale Schnittstelle des Systems dient die Berechtigung, welche Informationen über die Daten enthält, über Zugangsschnittstellen zur Bearbeitung und zum Bezug der Daten und Informationen über die Rechte an der Datei selbst. Diese Schnittstelle muss so einfach wie möglich beziehbar, benutzbar und lesbar sein. Da sich das System ohnehin im Bereich Internet und Hypertext Markup Language (nachfolgend HTML) bewegt, liegt es nahe, die eXtensible Meta Language (nachfolgend XML) zu benutzen. Auf diese Weise wird das Entwickeln von Parsern für ein neu entwickeltes Protokoll gespart, des weiteren ist XML "human readable" (also lesbar).

Das System bietet nun also eine im Netzwerk erreichbare Schnittstelle einer Berechtigung einer Datei, die über eine URL angesprochen wird. Wenn diese Überlegung weiter ausgemalt und nebenbei einbezogen wird, dass auch noch vordefinierte Workflows, also eine Benutzerschnittstelle, entwickelt werden muss, so kommt als Schlussfolgerung nur eine Webapplikation als zugrunde liegendes System in Frage, also eine Client/Server basierte Anwendung mit auf HTML und HTTP basierter Benutzerschnittstelle. Allerdings stellt sich hier die Frage, ob die Spezifikation des Systems festlegen sollte, ob es eine Webapplikation sein muss. Es ist auf jeden Fall die am leichtesten zu realisierende Implementationsmöglichkeit für das System, die Spezifikation wird diese aber nicht festlegen. Die Spezifikation wird lediglich die Verwendung von XML als Schnittstellenprotokoll und HTTP als Transportprotokoll für Schnittstellendaten festlegen. Aber dazu mehr im Kapitel 3 Spezifikation.

Ein weiterer Punkt der Flexibilität ist die Erweiterbarkeit des Systems, und genau hier ist die Streamingfähigkeit des Systems eine wichtige Eigenschaft. Wenn heute von Streaming gesprochen wird, so fallen automatisch Begriffe wie das Audioformat MP3 (MPEG Layer 3), das Videoformat MPEG oder das Videoformat DIVX, also heute gängige Medienformate. Um ein solches System erweiterbar zu machen, muss es in der Lage sein, wie ein herkömmliches Dateisystem, auch mit zukünftigen Formaten umgehen zu können. Insbesondere der Streamingserver des Systems darf also keine Einschränkungen in Bezug auf Format und Typ der zu übertragenden Daten machen. Deshalb muss ein Protokoll zum Einsatz kommen, das mit unterschiedlichsten Paketlängen und Paketinhalten zurecht kommt. Zur Übertragung von Datenströme ist

HTTP als Transportprotokoll generell erst einmal geeignet. Wenn jedoch bedacht wird, dass ein Datenstrom auch einmal mehrere Stunden oder Tage Daten transportieren kann, scheint es unnötig, einen HTTP-Header voran zu stellen. HTTP könnte also benutzt werden, muss es aber nicht. Der Einsatz kann also dem Implementierenden überlassen werden. Vielmehr muss ein Protokoll entwickelt werden, das möglichst viele Fremdentwickler benutzen können und das auch noch effizient Mediendaten, also auch Daten mit mehreren hundert Paketen pro Sekunde, verpacken kann.

Ein viel versprechendes Protokoll zur Übertragung ist das Realtime Protokoll, im Folgenden RTP [CASSCHU_1996] genannt. In früheren Entwicklungen [BEHA_2001] und Ausarbeitungen [HANFF_2001] wurden bereits die Möglichkeiten und Fähigkeiten von RTP und HTTP als Streamingprotokolle ausgiebig getestet und dort auch (speziell in [HANFF_2001]) die Überlegenheit von RTP gegenüber HTTP festgestellt. Insbesondere bei der Übertragung von Audiodaten in belasteten Netzen ist RTP dem HTTP in der resultierenden Sprachqualität überlegen.

RTP benutzt dabei Mechanismen zur Flußkontrolle, kann also Rückmeldung über die beim Empfänger erzielte Sprachqualität liefern, so dass der Sender Übertragungsrate und Kodierungsqualität den temporären Möglichkeiten des Netzes anpassen kann. Des weiteren wird bei RTP das User Datagram Protokoll, im Folgenden UDP [POSTEL_1980] genannt, genutzt, welches keine Garantien für Reihenfolge und Auslieferung von Paketen gibt. In stark belasteten Netzen verursacht diese Eigenschaft, dass bei einer Audioübertragung verloren gegangene Pakete nicht als störend empfunden werden. Im Gegensatz dazu wiederholt das beim HTTP eingesetzte Transmission Control Protocol, kurz TCP [TCP_1981], verloren gegangene Pakete. Das hat zur Folge, dass bei einer Audioübertragung diese doppelten Pakete, wenn nicht heraus gefiltert, auf jeden Fall störend wirken, anderenfalls zumindest die Performance beeinträchtigen. In der Arbeit [BEHA_2001] arbeitet das eingesetzte RTP über UDP völlig zufriedenstellend und kann in entsprechend ausgelegten Netzen Videos VHS-Qualität übetragen. Jedoch ist es durch den Einsatz von UDP nicht möglich ohne entsprechende RTP-Gateways oder Multicastrouter eine Kommunikation zwischen zwei einander unbekannten Subnetzen aufzubauen. Es setzt immer direkt adressierbare Kommunikationsparteien voraus.

Da dieses System nun so flexibel wie möglich sein soll, also auch von jedem Punkt

der Welt (sofern erforderliche Übertragungstechnik vorhanden ist), in jeder möglichen Umgebung und an jedem für das System geeignete Computer einsetzbar sein soll, disqualifiziert sich UDP. Aus diesem Grund wird beim Streamingserver TCP eingesetzt werden. Mit TCP kann eine Verbindung aus einem Subnetz in ein öffentliches Netz aufgebaut und damit auch die Übertragung vom öffentlichen in ein Subnetz realisiert werden, so dass also auch die Übertragung von zum Beispiel einem Server in ein Firmennetzwerk möglich ist, denn mit UDP ist das nicht ohne Umwege (Multicast oder entsprechende Gateways) möglich. Einer der größten Vorteile von RTP über UDP ist die Multicastfähigkeit, da aber ohnehin kaum ein öffentliches Netz (im Sinne des Internets), und diese öffentlichen Netze werden für größtmögliche Flexibilität betrachtet, multicastfähig ist und sich UDP schon disqualifiziert hat, disqualifiziert sich RTP ebenfalls für den Einsatz als Streamingprotokoll, auch wenn es standardisiert ist und noch eine Reihe weiterer Vorteile wie zum Beispiel eine Flußkontrolle hat.

Ein großer Vorteil ist die Zentralisierung des Streamingsystems, die Daten werden nicht Peer-To-Peer, also direkt von Sender zu Empfänger, sondern erst vom Sender zum Server und von dort aus an die Empfänger übertragen.

Für den Sender hat dieses Konzept die gleichen Fähigkeiten wie echtes Multicasting, da er seine Daten nur einmal zum System schicken muss und das System dann die Last der Verteilung übernimmt. Ein Multicastrouter erledigt im Sinne der Datenverteilung die gleiche Arbeit, allerdings aufgrund seiner Spezialisierung (Hardwareimplementation) meist ungleich effizienter. Ein nicht zu vernachlässigender positiver Nebeneffekt dabei ist, dass durch die Verwendung von TCP der Sender in der Lage ist, sein System durch eine Firewall komplett abzusichern. Bei der Verwendung bei RTP über UDP wäre dieses aufgrund der Verbindungslosen Übertragung nicht möglich, da der Empfänger immer Ports öffnen muss, um Datagramme empfangen zu können, wohingegen bei TCP der Empfänger die Verbindung nach außen aufbauen kann (siehe Kapitel 4.3). Dieser Sicherheitsvorteil beim Einsatz von TCP ist gerade beim Einsatz in sicherheitsrelevanten Anwendungsszenarien, wie zum Beispiel der zentralen Überwachung von Büros, ein großes Plus an Flexibilität.

Für den Privatnutzer ergibt sich durch die emulierte Multicastfähigkeit die Möglichkeit Videokonferenzen auch mit mehreren Personen zu führen, da er seine Bandbreite zu mindestens für den Upload seines Signals nur einmal belasten muss.

Als Basispfeiler für die Flexibilität soll zusammenfassend die Verwendung von HTTP als Kommunikationsprotokoll, die Verwendung von XML als Schnittstellenprotokoll für alle Verwaltungsfunktionen und die Verwendung eines zentralen Verteilungskonzepts genannt sein. Die in der Referenzimplementation des Systems in dieser Arbeit zu implementierende minimale Benutzerschnittstelle ist dabei zwar als Flexibilitätsmerkmal jedoch nicht als eine in der Spezifikation festgelegten Hauptfunktion zu betrachten.

3. Anforderungsanalyse

3.1. Konkurrierende Konzepte und Produkte

In den Bereichen Multimediastreaming, Content-Management und Dateisysteme gibt es schon eine Vielzahl Lösungen und Produkte. Als Paradebeispiele seien für Multimediastreaming die Firma Real Networks [REAL_2003] mit den Produkten Helix Producer und Realplayer und die Firma Microsoft [MISOLN_2003] mit dem Produkt Windows Mediaplayer genannt. Für den Bereich Content-Management gibt es wesentlich mehr (auf jeden Fall mehr als zehn) konkurrierende Ansätze und Produkte als im Bereich Multimediastreaming. Das mag zum einen daran liegen, dass dieser Bereich eher im so genannten B2B (Business to Business) Segment anzusiedeln ist, also im Gegensatz zu Multimediastreaming nicht auf einen Massenmarkt zielt. Zum Anderen liegt das sicher auch daran, dass dieser Bereich erst sehr jung ist und sich erst in den letzten vier Jahren mit dem Durchbruch des „dynamischen“ Internets entwickelte. Bei den Content-Management-Systemen werden aufgrund der Vielzahl von Lösungen keine konkreten Produkte genannt, sondern es wird vielmehr auf einige Lösungsansätze eingegangen.
Bei Dateisystemen für das Internet sei zunächst der offene Standard WebDAV (Web-based Distributed Authoring and Versioning) [STEIN_2003] erwähnt, der eigentlich schon die wichtigsten Eigenschaften eines Dateisystems implementiert. Ein weiteres System, das noch weiter geht als WebDAV ist Oracles IFS (Internet File System) [IFSLN_2003] ein Aufsatz für Oracles Datenbanksystem das auch noch diverse Aspekte eines Content-Management-Systems (unter anderem Versioning und Publishing) implementiert.

Um nun ein möglichst effizientes System zu entwickeln, muss nun untersucht werden, welche Anforderungen alle diese Systeme erfüllen und welche nicht.

3.1.1. RealNetworks Realplayer und Helixproducer

Zunächst soll das System Real von Real Networks betrachtet werden, das im Wesentlichen aus zwei Komponenten besteht, dem Realplayer [REALPL_2003] der für die Wiedergabe der Streams verantwortlich und für den Nutzer (in der Standardversion kostenlos) für alle gängigen Betriebssysteme verfügbar ist und dem Helixproducer [REALHP_2003] der für die Produktion der Streams, also die Kodierung, das Packaging und die Auslieferung verantwortlich ist. Der Helixproducer ist in einer eingeschränkten Version für den privaten Bereich kostenlos erhältlich. Selbst mit dieser kostenlosen Version können Streams in passabler Qualität über das Internet übertragen werden. Es können damit drei verschiedene Bandbreiten im Format H.263 erzeugt und genau zu einem Endpunkt übertragen werden. Real verwendet dabei sowohl HTTP als auch RTP und RTSP als Transportprotokolle, RTP dabei immer wenn möglich, sonst HTTP. H.263 [H263LN_2003] bietet dabei keine hervorragende Qualität, dafür aber eine Ressourcen schonende Codierung und Decodierung und eine geringe Anforderung an Bandbreite. Auf die besonderen Qualitäten verschiedener Komprimierungscodecs wird später noch eingegangen werden (Kapitel 3.2). So sind die Vorteile des Real Systems in der Verbreitung, durch die Verfügbarkeit auf allen gängigen Betriebssystemen, in der guten Qualität, die erzeugt werden kann und im niedrigen Preis (kostenlos für Privatanwender) zu finden. Die hohe Qualität begründet sich im Pufferverfahren, dass der Realplayer beim Empfang von Streams verwendet. So speichert der Player immer erst einige Sekunden eines ankommenden Streams, um dann bei eventuellen Störungen im Netz Daten ausgleichen und damit einen kontinuierlichen Strom darstellen zu können. Diese Methode begründet allerdings auch einen Nachteil, das System ist damit nicht echtzeitfähig, es können also keine Videokonferenzen oder Audiogespräche geführt werden. Mit der Einführung des Videoformats Realvideo 9 wurde der Einsatz des reinen H.263 abgelöst, trotzdem ist es mit Real nicht möglich andere Formate als Realvideo und Realaudio zu übertragen.

3.1.2. Microsoft Media

Das zweite System zur Übertragung von Multimedia ist jünger als das Real System, ist jedoch im semiprofessionellen Einsatzbereich sowie im hoch kommerziellen

Segment weit verbreitet. Die Rede ist von Microsofts Windows Mediaplayer und den dazugehörigen Formaten wie WMV, WMA, ASF die sich unter dem Pseudonym „Windows Media 9“ zusammenfassen lassen. Bei ASF (Advanced Systems Format) handelt es sich um eine Spezifikation für die Auslieferung, Codierung und Lizensierung für Windows Mediadaten. Microsoft ist einer der wenigen Anbieter, die zum jetzigen Zeitpunkt ein Digital Rights Management (DRM) implementiert haben, aus diesem Grund ist es auch bis jetzt das einzige eingesetzte System in vorhanden Video-On-Demand Plattformen. Bei den Komprimierungscodecs baut Microsoft auf vorhandene offene Standards wie etwa den MP3-Codec für Audio- und MPEG-4 für Videodaten, allerdings hat Microsoft an diesen Standards noch einiges geändert und durch sein implementiertes Digital Rights Management die Offenheit der eigenen Formate beschränkt und völlig inkompatibel zu den Ausgangsformaten gemacht. Einer der größten Nachteile ist somit die fehlende Unterstützung für andere Betriebssysteme als Microsofts Windows. Der Microsoft Media Player ist für Windows Betriebssysteme, Apple's Mac OS X, Apple's Mac, Sun's Solaris und sogar Palm-Betriebssysteme verfügbar. Für Linux gibt es einige OpenSource Gruppen, die schon funktionierende Programme auf Basis der Microsoft Media Player DLLs (Dynamic Link Libraries) entwickelt haben. Ähnlich wie Real Networks hat Microsoft auch einen Streamingserver und Producer. Diese Server sind optionaler Teil des neuen Serverbetriebssystems Microsoft Server 2003. Eine für den Privatanwender kostenlose Version des Streamingservers gibt es nicht. Um jedoch Microsofts Formate in vollem Umfang (also auch inklusive DRM) erstellen zu können, wird zumindest ein Microsoft Betriebssystem und Microsofts Moviemaker benötigt. Audiodaten lassen sich schon mit dem mitgelieferten Windows Mediaplayer erstellen.

Wenn beide Systeme betrachtet werden, fällt auf, dass sie eher proprietär sind, das heißt, dass sie nur eigene Formate und Systeme unterstützen, sie sind untereinander inkompatibel. So ist es zum Beispiel nicht möglich mit dem Windows Media Player das Realformat abzuspielen. Ein weiterer Nachteil beider Systeme ist die Unterstützung von jeweils nur eigenen unter Verschluss gehaltenen Codecs und Verpackungsformaten.
Zwar bieten beide Systeme eine hohe Übertragungs- und Bildqualität und ein überzeugendes Gesamtkonzept, jedoch müssen bei beiden Systemen Abstriche bei der Kompatibilität zu anderen Produkten und Systemen gemacht werden. Im Kapitel

3.2 werden die Komprimierungsformate beider Hersteller noch näher untersucht.
Bei der Konzeption und Spezifikation des Systems muss darauf geachtet werden, dass die fehlenden Funktionen und Anforderungen, also die Unterstützung aller möglichen auch zukünftigen Formate und Komprimierungscodecs und die Kompatibilität zu anderen Systemen ermöglicht wird.

3.1.3. Content-Management-Systeme

Bei den Content-Management-Systemen geht es in erster Linie um die Verwaltung von Content, das heißt um die Erstellung, die redaktionelle Aufbereitung und die Veröffentlichung von Inhalten. In den Anfangsjahren der CMS (nachfolgend für Content-Management-Systeme) waren diese Inhalte meist textuelle Inhalte für das Internet oder andere publizierende Bereiche. Diese Inhalte wurden meist versionisiert in Textdateien oder Datenbanken gespeichert.
Ein bei Softwareentwicklern beliebtes System zur Versionisierung von Quelltexten ist CVS [CVS_2003], das textbasierte Inhalte nur als Änderungen zur ersten Version, also mit jeder neuen Version lediglich die geänderten Zeilen, abspeichert. Diese Methode funktioniert auch bei den textbasierten Formaten des Internets, also HTML, PHP etc. noch hinreichend gut. Bei multimedialen Inhalten wie Videos und Audiodaten oder sogar Bildern funktionieren die meisten CMS nicht, da sie die Versionisierung dieser Daten nicht gewährleisten können. Für diese Formate gibt es auch keine hinreichenden Kategorisierungsmechanismen also auch keine assoziative Archivierung. Dieses Problem lässt sich verdeutlichen, wenn zum Beispiel mit Hilfe einer gesummten Melodie versucht wird ein ganzes Lied zu identifizieren oder zu finden. Es gibt Ansätze diese Daten mit Algorithmen aus der Fuzzy-Technologie und der neuronalen Netze zu kategorisieren. Zu diesen Mustererkennungsverfahren auf Prinzip der neuronalen Fuzzy-Logik gibt es viele Ausarbeitungen und Papiere auf die hier aber nicht näher eingegangen werden wird.
Ein weiteres Schlagwort im Zusammenhang mit CMS ist MVC (Model-View-Control) , die Trennung von Inhalt, Struktur und Präsentationsoptik. Es geht hierbei also um die getrennte Behandlung und Speicherung des Inhaltes von seinen Darstellungsmethoden. Eine Technik die sich dabei durchgesetzt hat ist XML (eXtensible Markup Language), die Oberklasse von HTML und SGML. Mit dieser Technik ist es möglich die Daten unformatiert innerhalb so genannter Tags zu speichern. Für die Struktur der Daten sorgen dann die verwendeten Tags. Ein

Beispiel für die Strukturierung eines Dokumentes sieht so aus:

<document>

<title>

Der eigentliche Titel des Dokumentes

</title>

<chapter>

<title>

Der Titel des Kapitels

</title>

<paragraph>

Erster Absatz des Kapitels

<paragraph>

</chapter>

</document>

Dieses kleine Beispiel zeigt die Möglichkeiten der Strukturierung. Die Tags werden in so genannten DTD (Document Type Definitions) definiert. Eine DTD für Textdokumente ist die Docbook DTD [WALSH_2003], die die in Dokumenten am häufigsten benutzten Strukturmerkmale als Tags, wie in etwa Kapitel, Titel, Absätze, Unterabschnitte etc. beinhaltet. Die Möglichkeiten für die Präsentation der strukturierten Daten ergibt sich durch den Einsatz von Stylesheets. Im Zusammenhang mit XML sind das so genannte XSL Stylesheets (eXtensible Stylesheet Language), die wiederum auf dem XML-Format beruhen. Mit XSL kann nun die Darstellung eines jeden Tags definiert werden. Dabei umfasst XSL nebenbei noch umfangreiche Standards zum Erstellen von Scripts, XSLT (XSL Transformations) genannt. Mit XSL können sowohl XML-basierte Darstellungsformate als auch komplizierte oder gar binäre Ausgabeformate erzeugt werden. Ein gutes Beispiel ist eine Stylesheetbibliothek mit dem Namen FOP (Formatting Object Processor) [APFOP_2003] der Apache Software Group, die es ermöglicht, aus zum Beispiel Docbook-XML-Dokumenten, PDF-Dokumente zu erzeugen. Ein kleines Beispiel für die Transformation des obigen Beispiels in eine HTML-Datei soll das Model-View-Konzept verdeutlichen:

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="html" indent="yes"/>

```
 <xsl:template match="document">          <!-- die Umsetzungsvorschrift für document -->
   <html>
    <head>
      <title><xsl:value-of select="title"/></title> <!-- fügt den Inhalt des Title-Tags ein -->
    </head>
    <body background="#ffffff">
      <xsl:apply-templates select="chapter"/><br/> <!--ruft die Vorschriften für alle chapter auf -->
    </body>
   </html>
 </xsl:template>
 <xsl:template match="chapter"> <!-- Vorschrift für chapter -->
   <h2><xsl:value-of select="title"></h2>
     <xsl:apply-templates select="paragraph"/> <!--ruft die Vorschriften für alle enthaltenen Absätze auf -->
 </xsl:template>
 <xsl:template match="paragraph"> <!-- Vorschrift für paragraph -->
   <p><xsl:value-of select="."/></p> <!-- fügt den Text des paragraph-Tags ein -->
 </xsl:template>
</xsl:stylesheet>
```

Das Beispiel kann nur begrenzt die Möglichkeiten von XML/XSL aufzeigen, verdeutlicht jedoch, wie mit XML/XSL diese Trennung von Inhalt, Struktur und Präsentation erreicht werden kann. Eines kann XML/XSL allerdings nicht: mit multimedialen Daten umgehen. Für den Bereich nicht bewegter Bilder gibt es das auf XML basierendes Format SVG (Scalable Vector Graphics) [SVG_2003], aber für bewegte Bilder oder für Audiodaten gibt es noch keine verwendbaren Ansätze. Das mag zum Einen daran liegen, dass XML einen starken Overhead (da textbasiert) erzeugt, was bei textuellen Daten oder Bildern noch vernachlässigbar ist, jedoch bei hochqualitativen Videodaten nicht zweckmäßig erscheint. Zum Anderen sind frequenzbasierte Daten wie Audio und Video nicht strukturiert und somit gänzlich ungeeignet für die Darstellung in XML.

Wenn also multimediale Daten wie Audio und Video in einem CMS Kontext verwaltet werden sollen, so bleibt entweder der Einsatz einer Datenbank oder das binäre Abspeichern in einem System wie CVS. So wäre zu mindestens die Versionisierung auf unterster Ebene gesichert. Eine in sich schlüssige Lösung des Problems scheint in nächster Zeit jedoch nicht in Sicht.

Die Grundfunktionen eines jeden CMS sind eine Benutzerverwaltung, ein Versionisierungsmechanismus, ein Redaktionsmechanismus, ein Verteilmechanismus und ein Veröffentlichungsmechanismus. Die Benutzerverwaltung muss dabei auch Berechtigungsfunktionen übernehmen, da nicht jeder Nutzer alle Daten sehen darf oder zum Beispiel Redaktionsrechte besitzen kann. Der Redaktionsmechanismus sorgt dafür, dass Daten erst nach Abnahme durch einen Redakteur veröffentlicht werden können, dass Daten zur Bearbeitung gesperrt werden können usw. Der Versionisierungsmechanismus sorgt für die Versionisierung der geänderten Daten, der Verbreitungsmechanismus für die Verteilung der Daten an ihre vorgesehen Orte, also auch das Deployment. Der Veröffentlichungsmechanismus charakterisiert die Aufbereitung der Daten für ein öffentliches System. Alle diese Mechanismen wird das System dieser Arbeit auch implementieren. Einzig der Redaktionsmechanismus für die Abnahme durch einen Redakteur kann auch auf höherer Ebene realisiert werden und muss hier nicht integriert werden.

3.1.4. Internetdateisysteme

Im Zusammenhang mit den genannten CMS-Mechanismen lassen sich auch die Dateisysteme für das Internet näher betrachten, so ermöglicht die WebDAV-Spezifikation ebenfalls eine Benutzerverwaltung durch die in HTTP implementierten Sicherheitsfunktionen. Hier ist auch die Verteilung der Daten geregelt, die Versionisierung ist laut Spezifikation auch möglich. Der Redaktionsmechanismus wird hier „Authoring“ genannt, ermöglicht also auch das Sperren von Dateien während der Bearbeitung usw. Einzig die Aufbereitung der Daten ist bei WebDAV nicht geregelt, da aber zum Beispiel jeder Apache Webserver das WebDAV Protokoll unterstützt und sich im Apache Webserver auch Servletengines wie der Apache Tomcat einbinden lassen, kann mit Hilfe von Apache Cocoon oder anderen XML/XSL-Techniken auch die Aufbereitung der Daten mittels XML und XSL realisiert werden.

Das WebDAV Protokoll kann also zu einem großen Teil die Anforderungen an das zu entwickelnde System bis auf den Umgang mit Streamingdaten erfüllen. Die Hauptmerkmale von WebDAV sind also das unterliegende HTTP-Protokoll, die Möglichkeiten zur Benutzerverwaltung und zur Versionisierung.

Eine hier kurz theoretisch untersuchte Möglichkeit, die fehlende Streamingunterstützung zu kompensieren, würde das Streaming der Daten emulieren, in dem der Senderprozess in eine Datei schreibt und andere Empfängerprozesse gleichzeitig vom Ende der Datei lesen. Damit wären gleich zwei andere Funktionen implementiert, die Archivierung der Daten und eine Art Time-Shifting, also die Möglichkeit den Stream ab der aktuellen Position oder ab Anfang auszulesen. Bei Videokonferenzen könnte ein neuer Teilnehmer so das Geschehen von Anfang an anschauen um dann aktiv am Gespräch teilnehmen zu können. Das Problem bei dieser Technik ist, dass die dem HTTP unterliegenden vom Betriebssystem abhängigen Dateisysteme dieses synchrone Schreiben und Lesen einer einzigen Datei meist nicht unterstützen und somit eine generelle Plattformunabhängigkeit nicht gewährleistet werden kann.

Ein weiteres Problem dabei ist, dass viele Streamingformate, insbesondere Video-Packagingformate wie *.MPG und *.AVI bestimmte Headerinformationen der eigentlichen Videodaten in Clustern in die Datei schreiben und das dann meist erst bei Kenntnis der vollen Länge eines Clusters. Somit können also die Daten nicht in Echtzeit wieder ausgelesen und wiedergegeben werden.

Aus diesem Grund kann das WebDAV Protokoll aufgrund der fehlenden Streamingunterstützung nicht als Grundlage für das zu entwickelnde System zum Einsatz kommen.

Wünschenswert ist jedoch die Kompatibilität zum WebDAV Protokoll, also die Möglichkeit feste Daten über das WebDAV Protokoll zu verteilen, das wird allerdings mit der in Kapitel 2.2. erwähnten eindeutigen Identifizierung durch Hashes im System nicht möglich sein. WebDAV erlaubt es zum Beispiel auch Ordnerstrukturen anzulegen und damit Daten mehrfach zu speichern. Aber das soll ja gerade vermieden werden. Zusammengefasst bedeutet das, dass existierende Systeme meist auf Ordnern basieren, also auf einer Ordnung durch manuelle Kategorisierung. Einzig das CVS- System benutzt für die Hauptordnung eine Versionen-Kategorisierung, arbeitet aber darüber auch wieder mit Ordnern und Dateien um mit existierenden Dateisystemen kompatibel zu sein.

Die Anforderungen an das System ergeben sich also nun in den Begriffen Kompatibilität, Streamingfähigkeit, Unabhängigkeit von Formaten und Codecs, Versionisierungsfähigkeit, Trennung von Inhalt, Struktur und Präsentation (zumindest für die Benutzerschnittstelle), Offenheit, Benutzerverwaltung und Berechtigungssystem (oder auch Redaktionelles System).

3.2. Codecs

Als Anforderung an das System wurde bereits die Erweiterbarkeit und die Unabhängigkeit von bestimmten Streamingformaten genannt. Dennoch muss für die Spezifikation des Systems untersucht werden, mit welchen Formaten und Hardwareanforderungen zu rechnen ist. Dabei sind insbesondere Kodierungseffizienzen und Qualitätsmerkmale für bestimmte Einsatzbereiche zu untersuchen. In diesem Abschnitt werden zu diesem Zweck eine Reihe der heute am häufigsten verwendeten Codecs (nachfolgend für Audio- und/oder Videokomprimierungsverfahren) vorgestellt.

Die bekanntesten Audiocodecs (abgesehen von PCM) haben so geringe Bandbreitenanforderungen, dass meist nur die Codecs verwendet werden, die die beste Qualität liefern. Dennoch gibt es auch bei der Qualität zwei Bereiche unterschiedlicher Einsatzszenarien. Zum Einen der Einsatz in der Telefonie, wo weniger High Fidelity (HiFi) Merkmale als vielmehr Kodierungseffizienz und Bandbreitenschonung als vordergründige Auswahlkriterien gelten. Zum Anderen zielt der Einsatz im Entertainmentbereich, also Musik und Film, eher auf Merkmale höchster Klangqualität als auf Bandbreitenanforderungen.

Im „Low-Bandwith“ Bereich gibt es zwei verbreitete Verfahren: GSM und G.723. GSM [ETSIGSM_1992] ist dabei das Sprachkomprimierungsverfahren des Mobilfunksystems GSM (Global System for Mobile Telecommunication) und kommt mit einer Bandbreite von 13 kbit/s aus. Ein weiterer Vorteil von GSM ist die Kodierungseffizienz, das heißt, dass GSM generell wenig Rechenleistung zur Kodierung und zur Dekodierung benötigt. Dadurch kann GSM auch in schmalbandigen Systemen mit wenig Rechenleistung, wie in Mobiltelefonen, eingesetzt werden. GSM benutzt ein Kodierungsverfahren mit dem Namen RPE-LTP (Regular Pulse Excitation Long-Term Prediction) [SEMPERE_1997], welches

Sprache in 20ms lange Blöcke unterteilt und jeweils vorhergegangene Blöcke benutzt um den Aktuellen vorher zusagen. Nach [HANNAN_1995] ist dieses RPE-QT Verfahren ein Verfahren, das vorrangig für den Telefoniebereich entwickelt wurde und Sprache in einer hohen Qualität übertragen soll. Es benutzt dabei bestimmte Redundanzmerkmale der menschlichen Sprache und ist aus diesem Grund auch für den Entertainmentbereich völlig ungeeignet. Dieses Kodierungsverfahren ist nicht das einzige dieser Art, hat sich jedoch auch aufgrund der nativen Implementation in Microsofts Betriebssystem ab Windows 2000 durchgesetzt um Sprache zu übertragen. Microsoft selbst benutzt zur Sprachübertragung jedoch den G.723 Codec. G.723 ist Teil des H.323 Standards von der ITU. Der Codec gliedert sich in eine ganze Reihe von Audiocodecs der ITU ein, darunter die Codecs G.711, G.722 und G.729. Alle diese Codecs haben ihre Vor- und Nachteile, breite Unterstützung fand allerdings nur der G.723, und das sicher auch aufgrund der bevorzugten Benutzung in Microsofts H.323 basiertem Netmeeting. Tabelle 1 zeigt die Komplexitäten und Datenraten der ITU Audiocodecs und lässt erkennen, warum sich G.723 durchgesetzt hat. Der Hauptgrund dafür dürfte die extrem geringe Bandbreite sein, G.723 kommt mit einer maximalen Bandbreite von 6,4 kbit/s aus, das ist nicht einmal halb so viel wie GSM benötigt. Allerdings liegt die subjektive Qualität auch unter der von GSM. Aber gerade in Anwendungen wie der IP-Telefonie, die schmalbandige Sprachübertragung benötigen, können diese Qualitätsverluste unbeachtet bleiben. Die Arbeit [GOEHR_2001] beschäftigt sich ausgiebig mit den Bandbreiten und Qualitätsmerkmalen der G.700 Audiocodecs der ITU. Allerdings zieht sie keinerlei Vergleich zu GSM oder ähnlichen Sprachcodecs. In der Arbeit [BEHA_2001] wird jedoch darauf hingewiesen, dass der GSM-Codec in Sachen Kodierungseffizienz und Sprachqualität der Bessere ist.

Standard	*Bitrate*	*Framesize / Lookahead*	*Complexity*
G.711 PCM	64 kbit/s	0 / 0 ms	0 MIPS
G.726, G.727	16, 24, 32, 40 kbit/s	0.125 / 0 ms	2 MIPS
G.722	48, 56, 64 kbit/s	0.125 / 1.5 ms	5 MIPS
G.728	16 kbit/s	0.625 / 0 ms	30 MIPS
G.729	8 kbit/s	10 / 5 ms	20 MIPS
G.723	5.3 & 6.4 kbit/s	30 / 7.5 ms	16 MIPS

Tabelle 1 Vergleich der Audiocodecs der G.700 Reihe (aus [MANDUCHI_2002])

Für den „High-End“ Audiobereich sind diese Codecs eher ungeeignet, da sie auf bestimmte Eigenschaften der menschlichen Sprache ausgerichtet sind und zum Beispiel Nebengeräusche und Störeffekte oder Verzerrungen absichtlich heraus filtern. Im Entertainmentbereich haben sich in den letzten Jahren besonders das MP3 (MPEG-1 Layer 3) [MP3LN_2003] und das MPEG 1 Layer 2 (auch MP2 genannt) Format durchgesetzt, wobei das MP3 Format eher im reinen Musikbereich und das MP2 Format in Filmen zum Einsatz kommt. Im Zusammenhang mit den Videostandards VCD (Video Compact Disc) und SVCD (Super Video Compact Disc) welche auf den Videocodecs MPEG 1 und MPEG 2 beruhen, kam auch das Audioformat MP2 zum Einsatz. Im Bereich von MPEG-4 Videoformaten (oft auch fälschlicher Weise als DIVX bezeichnet) hat sich in letzter Zeit das AC-3 [AC3_1995] Audioformat und das MPEG 2 AAC (Advanced Audio Coding) [MP2LN_2003] durchgesetzt. Der MP3 Codec wird dabei meist in Bandbreiten um die 128 kbit/s eingesetzt und bietet eine Abtastrate von 44100 kHz also CD-Qualität.

In diesem Bereich der Audiokomprimierung gibt es viel Bewegung, unter anderem warten Neuentwicklungen des MPEG-4 Standards auf ihre Einführung. Die Bandbreiten der Codecs reichen von mehreren Mbit/s bis zu 32 kbit/s, aber verlustbehaftet sind sie, (wie MP3) alle. Eigentlich sind diese Codecs nicht fähig Rohdaten in Echtzeit zu kodieren, allerdings sind die Rechner heute so schnell, dass die Kodierungssoftware die Kodierung ohne merklich spürbare Zeitdifferenz erledigt.

Bei der Kodierung von Filmen im VCD Standard wird ein MP2 Strom mit einer Bandbreite von 224 kbit/s kodiert. Das dazugehörige MPEG 1 Video [SIKORA_2003] wird in einer Auflösung von 352x288 Bildpunkten mit 25 Bildern pro Sekunde bei PAL und einer Auflösung 352x240 Bildpunkten mit 29,97 Bilder pro Sekunde bei NTSC

mit einer Bandbreite von je 1150 kbit/s kodiert. Diese Kodierung führt zu einer Qualität ähnlich der von VHS. Das MPEG 1 Videokomprimierungsverfahren basiert dabei im Wesentlichen auf der diskreten Cosinus Transformation, die ein Bild, aufgeteilt in Blöcke, als Funktion ansieht, die dann als Polynome dargestellt werden können. Durch diese Darstellung können die Daten des Polynoms, welches in der Algebra nur eine Ansammlung von Koeffizienten ist, reduziert werden. Dabei gehen allerdings Informationen über den Verlauf des Polynoms, also des Bildes, verloren. Nun werden dabei jedoch noch Informationen über die Bewegung der Blöcke gesammelt, um das Video noch mehr zu komprimieren. Dieses Verfahren wird Motion Compensated Prediction genannt, Details darüber sollen im Rahmen dieser Arbeit nicht weiter erläutert werden. MPEG 1 bietet an sich schon eine subjektiv gute Qualität, ist allerdings nur für Bandbreiten bis zu 1500 kbit/s und Auflösungen bis zu 352x288 ausgelegt.

Um auch höhere Auflösungen und höhere Bandbreiten (für bessere Qualität) ermöglichen zu können, wurde das Videokomprimierungsverfahren MPEG 2 entwickelt, das unter Anderem beim Super Video CD Standard [SVCD_2003] und beim DVD Video eingesetzt wird. Beim SVCD Standard werden Auflösungen von 480x576 bei PAL und 480x480 bei NTSC erlaubt. Die Bandbreiten können dabei bis zu 2,3 Mbit/s betragen. MPEG 2 unterstützt allerdings auch wesentlich höhere Bandbreiten wie bei DVD Video mit einer Bandbreite von 9.8 Mbit/s. Das von SVCD unterstützte Audioformat ist MP2 also MPEG 1 Layer 2 und MPEG 2 mit Surround Erweiterung bei Bandbreiten bis zu 384 kbit/s. Das MPEG 2 Videokomprimierungsverfahren ist dabei im Grunde genommen dem MPEG 1 Verfahren gleich, kommt aber aufgrund verbesserter Dekodierungseffiziens und Qualität in vielen Bereichen wie DVD oder dem digitalen Fernsehen vor.

Ein weiteres Verfahren zur Videokodierung ist das H.263 Verfahren, das ein Nachfolger des H.261 der CCITT ist. Entwickelt wurde dieses Verfahren für Videotelefonie in IDSN basierten Netzen, kommt also mit Bandbreiten von n*64 kbit/s aus. Abbildung 1 zeigt den Unterschied zwischen MPEG (reines JPEG ohne Motion Prediction) und H.263 bei niedrigen Frameraten und verschiedenen Auflösungen. Es ist ersichtlich, dass H.263 auch bei Bandbreiten von weniger als 64kbit/s arbeiten kann, wenn nur die Frameraten niedrig (weniger als 5 Bilder pro Sekunde) gehalten werden. H.263 benutzt wie MPEG die diskrete Cosinustransformation, besitzt aber die Möglichkeit, die Qualität der geforderten Bandbreite anzupassen, das heißt also,

dass die Qualität zu vorgegebener Bandbreite gesteuert werden kann. Diese Eigenschaft und die gute Kodierungseffizienz machen das H.263 Verfahren zu einem beliebten Verfahren für Echtzeitstreaming. Tabelle 2 zeigt einen Performancevergleich bei den unterschiedlichen Methoden zur Bewegungsabschätzung (Motion Estimate) bei H.261, dem Vorgänger von H.263, und lässt dabei in etwa die Kodierungseffizienzen der Verfahren erkennen. Der Parameter „p“ charakterisiert dabei die Größe des Suchfeldes, in dem nach Bewegung gesucht wird.

Suchmethode	***Operationen bei 720x480 bei 30 fps und p=15***	***Operationen bei 720x480 30 fps und p=7***
Full Search	29.89 GOPS	6.99 GOPS
Logarithmic	1.02 GOPS	777.60 MOPS
Hierarchichal	507.38 MOPS	398.53 MOPS

Tabelle 2 Performancevergleich von Motion Estimate Methoden bei H.261 (aus [NIANLI_1996])

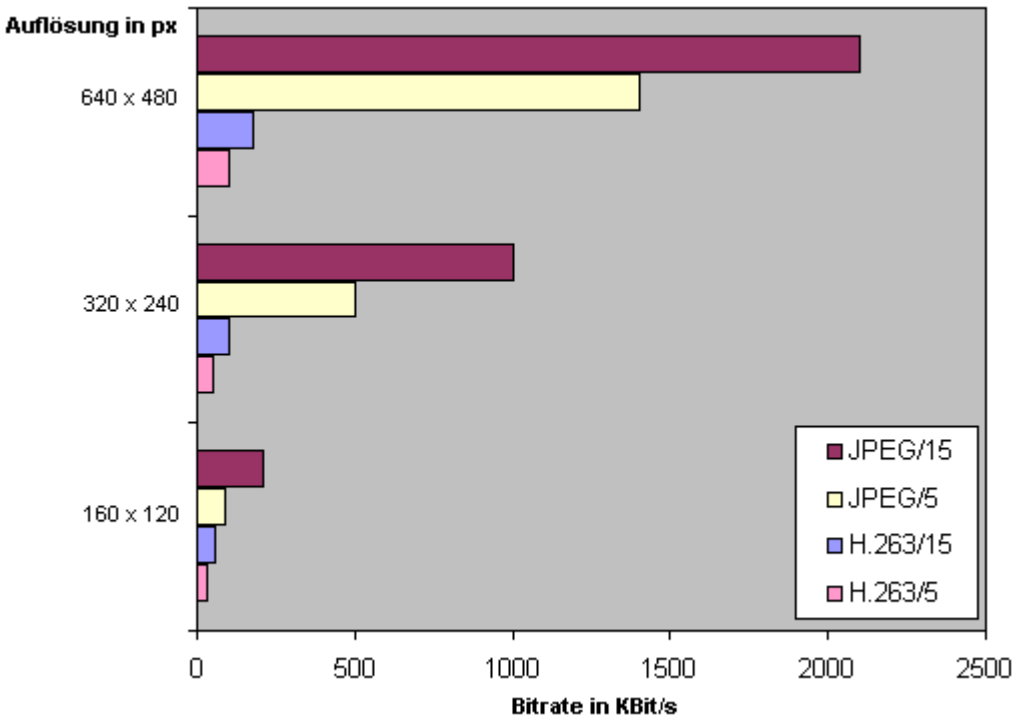

Abbildung 1 Bandbreitenvergleich bei Videokompression (aus [BEHA_2001])

Ein relativ neues Verfahren zur Videokomprimierung ist das MPEG-4 [KOENEN_2002] Verfahren, welches unter anderem von Microsoft benutzt wird. Es

wurde ursprünglich entwickelt um multimediale Inhalte über das Internet Bandbreiten schonend zu übertragen. Anfänglich wurde dazu eine Auflösung von 176x144 Pixeln vorgeschlagen und von Microsoft im WMV-Codec auch so implementiert. Die Bildqualität von MPEG-4 gegenüber MPEG-2 ist bei gleicher Bandbreite wesentlich höher, produziert also weniger sichtbare Kodierungsartefakte, und ist dadurch gerade bei Anwendungen im Internet gut einsetzbar.

Das MPEG-4 Verfahren ist dabei objektorientiert, das heißt, dass ein Szene aus einer Menge von Videoobjekten besteht, so genannten VOs. Jedes dieser VOs ist in Makroblöcke unterteilt. Die Oberfläche dieser Blöcke, also die Textur, wird dann wie bei den meisten Videocodecs mit der Diskreten Cosinustransformation komprimiert. Für die Bewegungsvorhersage der Videoobjekte wird ein Motion-Vector Verfahren benutzt, das sowohl in Vorwärtsrichtung als auch in bidirektionaler Richtung funktioniert und zur Vorhersage somit auch das jeweils folgende Bild benutzen kann, wobei die Vorhersage nicht nur das jeweils nächste Bild betrifft. Für jedes Videoobjekt werden die Informationen der Form, der Bewegung und der Oberfläche jedes einzelnen Makroblocks getrennt gespeichert und übertragen.

Abbildung 2 H263+ kodiert bei 64 kbit/s (Anlage A - [H263p_64])

Abbildung 3 MPEG-4 kodiert bei 64 kbit/s (Anlage A - [MP4_64])

Quality	*Trace*	*Compr.ratio YUV:MP4*	*Framesize Mean X in kbyte*	*Framesize CoV Sx/X*	*Framesize Peak/Mean Xmax/X*	*Bitrate Mean X/t in Mbit/s*	*Bitrate Peak Xmax/t in Mbit/s*
High	Jurassic Park	9,92	3,80	0,59	4,37	0,77	3,30
High	Silence of the Lambs	13,22	2,90	0,80	7,73	0,58	4,40
High	Star Wars IV	27,62	1,40	0,66	6,81	0,28	1,90
High	Mr.Bean	13,06	2,90	0,62	5,24	0,58	3,10
High	First Contact	23,11	1,60	0,73	7,59	0,33	2,50
Medium	Jurassic Park	28,40	1,30	0,84	6,36	0,27	1,70
Medium	Silence of the Lambs	43,83	0,88	1,21	13,60	0,18	2,40
Medium	Star Wars IV	97,83	0,39	1,17	12,10	0,08	0,94
Low	Jurassic Park	49,46	0,77	1,39	10,61	0,15	1,60
Low	Silence of the Lambs	72,01	0,53	1,66	21,39	0,11	2,30
Low	Star Wars IV	142,52	0,27	1,68	17,57	0,05	0,94

Tabelle 3 MPEG-4 Kodierstatistiken aus [FIREI_2000]

Rate	*Trace*	*Mean F in byte*	*CoV Sf/F*	*Peak/Mean Fmax/F*	*Mean R in kbit/s*	*CoV Sr/R*	*Peak/Mean Rmax/R*
16 kbit/s	Jurassic Park	79,81	2,48	111,96	16	0,35	5,8
16 kbit/s	Silence of the Lambs	79,8	2,38	157,14	16	0,37	6,3
16 kbit/s	Star Wars IV	79,81	2,15	46,24	16	0,42	6,1
64 kbit/s	Jurassic Park	319,59	1,73	27,96	64	0,42	5,7
64 kbit/s	Silence of the Lambs	319,6	1,77	39,23	64	0,42	5,7
64 kbit/s	Star Wars IV	319,62	1,81	25,66	64	0,45	5,2
256 kbit/s	Jurassic Park	1278,72	1,72	9,24	256	0,42	5,5
256 kbit/s	Silence of the Lambs	1278,61	1,73	17,4	256	0,42	5,9
256 kbit/s	Star Wars IV	1278,8	1,72	12,76	256	0,47	5,3
VBR	Jurassic Park	2225,26	1,23	8,16	450	0,69	7,7
VBR	Silence of the Lambs	1509	1,6	18,41	300	1,1	17
VBR	Star Wars IV	589,63	1,24	15,25	120	0,76	11

Tabelle 4 H.263 Kodierstatistiken aus [FIREI_2000]

In den Tabellen Tabelle 3 und Tabelle 4 sind die durchschnittlichen Datenraten bei MPEG-4 und H.263 Kodierungen von einigen Kinofilmen festgehalten. Es ist auf den ersten Blick erkennbar, dass bei der MPEG-4 Kodierung das erste Kriterium die Qualität der Kodierung ist, also in High, Medium und Low untergliedert ist, wogegen bei H.263 nach gewünschter Bitrate unterschieden wird. Die Bitraten bei MPEG-4 liegen meist weit über denen von H.263, so dass nur noch zu klären bleibt, bei welchen Bitraten der H.263 Strom dem MPEG-4 Strom subjektiv qualitativ gleichwertig ist. Aus rechtlichen Gründen können die Beispieldateien nicht beigelegt werden. Aus diesem Grund wurden weitere Qualitäts- und Kodierungstests vorgenommen. Die Dateien dazu befinden sich im Anhang. Grundlage der Tests ist die Neukodierung eines HDTV-Trailers (18MBit/s). Dabei wurde das Original in ein Video mit der Auflösung von 720x576 (DVD-Auflösung) und einer Bandbreite von 9600 kbit/s (DVD-Bandbreite) mit MPEG-1 kodiert. Danach wurde jeweils mit dem Kodierungstool FFMPEG [FFMPEGLN_2003] unter Linux auf einem Pentium 4 mit 1.7 Ghz Taktfrequenz und 256 MB PC-133 SDRAM das Video mit verschiedenen

Videocodecs komprimiert und die Zeit für die Komprimierung gemessen. In Abbildung 2 und Abbildung 3 sind die Unterschiede eines mit H.263 kodierten Videos und eines mit MPEG-4 kodierten Videos erkennbar. Die Qualität des Bildes ist bei MPEG-4 höher (es sind also weniger Blöcke zu sehen), da hier absichtlich eine Szene mit Bewegung gewählt wurde. In der Testreihe wurde mit MPEG-1, MPEG-4, WMV (Microsoft MPEG-4), H.263 und H.263+ in mehreren Bitraten kodiert. Die Bitraten reichen von 64 kbit/s bis 4000 kbit/s. Die Daten (siehe Anlagen) lassen erkennen, dass MPEG-4 eindeutig die beste Qualität (es sind die wenigsten Makroblöcke zu erkennen) bietet. In Tabelle 5 sind die Messdaten dargestellt, wobei der Versuch nur genau einmal ausgeführt wurde, da sich beim ersten Versuch keine signifikanten Unterschiede herausgestellt haben. Die Genauigkeit ist mit 1/10 Sekunde zu betrachten. So ist ersichtlich, dass alle Codecs die Kodierung des 16 Sekunden langen Videos in etwa 10 Sekunden erledigen, also auf dem benutzten System echtzeitfähig sind. Interessant ist dabei, dass es kaum Unterschiede bei der Kodierung in unterschiedliche Bitraten gibt.

	64 kbit/s	***256 kbit/s***	***512 kbit/s***	***1000 kbit/s***	***2000kbit/s***	***4000 kbit/s***
H.263	9.120 s	9.33 s	9.55 s	9.97 s	10.13 s	10.01 s
H.263+	9.15 s	9.49 s	9.65 s	9.92 s	9.99 s	10.2 s
MPEG-4	9.34 s	9.53 s	9.70 s	9.96 s	10.02 s	9.95 s
WMV	9.53 s	9.71 s	10.00 s	10.32 s	10.39 s	10.49 s
MPEG-1	9.05 s	9.24 s	9.37 s	9.79 s	10.09 s	10.02 s

Tabelle 5 Vergleich der Kodierungsgeschwindigkeiten bei Videocodecs

Die gesammelten Daten lassen erkennen, dass bei der Auswahl der Hardwareparameter auch Rücksicht auf die Kodierungszeiten genommen werden muss, da es möglich sein muss nicht nur ein Videobild zu kodieren, sondern im Falle von Videokonferenzen auch mehrere zu dekodieren.

3.3. Untersuchung von Hardwareparametern

Zu den Hardwareparametern gehören leistunsbedingte Einschränkungen und Vorgaben durch bestimmte Benutzergruppen. Leistungsbedingte Einschränkungen sind dabei durch Rechnerarchitekturen gegeben, die geforderte Leistungen nicht

erfüllen können, hier speziell zum Beispiel ein älterer Rechner, der es nicht schafft ein flüssiges Video in Echtzeit zu kodieren. Vorgaben durch Benutzergruppen äußern sich in der Verfügbarkeit von Rechnerarchitekturen in der Benutzerzielgruppe. So muss zum Beispiel eine Software, die für den Massenmarkt entwickelt werden soll, eine Rechnerarchitektur unterstützen, die von dieser Benutzergruppe verwendet wird. Die hier zu entwickelnde Software besteht aus grundsätzlich zwei Komponenten, einem Server und der Clientsoftware. Wie Server- und Clientsoftware letztendlich programmiert werden, wird hier noch keine Rolle spielen, es gilt vielmehr zu untersuchen, mit welchen Parametern Server und Client in Bezug auf die eingesetzte Hardware ausgestattet werden. Dabei sind Betriebssystem, Hardware und die zu entwickelnde Software eng miteinander verknüpft, da die Rechnerarchitektur das Betriebssystem und dadurch auch Anforderungen an die zu entwickelnde Software bestimmt.

Zunächst wird die Serversoftware betrachtet werden, die das Dateisystem repräsentiert, also mehrere hundert Benutzer gleichzeitig bedienen können muss. Da das Dateisystem den Zugriff über das Internet ermöglichen muss, muss der Server auf einem System laufen, das einen öffentlichen Zugang zum Internet hat und zu dem eine TCP/IP-Unterstützung.

Mit den Anforderungen an die Plattformunabhängigkeit des Systems stellt sich nun die Frage, wie dieses erreicht werden kann. Die meisten Betriebssysteme haben heute eine TCP/IP-Unterstützung, so dass diese Frage wohl keine Bedeutung hat. Werden nur die gängigsten Betriebssysteme wie Microsoft Windows, Sun Solaris, Linux, Mac OS X und Unix betrachtet, so kann eine Software, die alle diese Betriebssysteme unterstützt, durchaus plattformunabhängig genannt werden. Nach [SCHULTE_2001] haben bei den Serverbetriebssystemen Unix und Linux zusammen 71% Marktanteil und Microsoft Systeme nur 25%.

Ein weiterer Punkt ist die Hochverfügbarkeit des Serverbetriebssystems und die Auslegung der Hardwarearchitektur auf Skalierbarkeit. So empfiehlt sich auf jeden Fall ein Unixsystem, das zur Skalierbarkeit im Cluster eingesetzt werden kann. Da Unixsysteme auch für die PC-Rechnerarchitektur verfügbar sind, bestimmt diese Rechnerarchitektur alle folgenden Hardwareparameter wie Festplattenkapazität, Netzwerkanbindung und Arbeitsspeicher.

Aber nicht nur das Betriebssystem muss hoch verfügbar sein, sondern auch die Software die entwickelt werden soll. In letzter Zeit ist immer wieder von „Löchern“ in

Serversoftware zu hören, also Sicherheitslecks die zumeist durch Bufferoverflows oder Bufferunderruns entstehen. Da bei diesem hier zu entwickelnden System solche Probleme von Anfang verhindert werden sollen, stellt sich die Frage nach der Programmiersprache, in der das System programmiert werden soll. In der E-Business-Welt hat sich Java als optimale Programmiersprache durchgesetzt. Java hat durch ein ausgereiftes Speicher- und Cachemanagement gerade beim Verwalten von Datenmengen, in der Größenordnung die Datenbanken verwalten, gegenüber anderen Laufzeitsystemen erhebliche Vorteile. Da Java zusätzlich mit einem ausgereiften Sicherheitssystem aufwarten kann und durch seine Virtual-Machine-Strategie auf allen relevanten Betriebssystemen ohne Neuübersetzung ([JRELN_2003]) des Quelltextes läuft, fällt die Entscheidung bei der Programmierung des Servers auf Java. Da der Streamingserver formatunabhängig laufen soll, werden serverseitig keine Kodierungsvorgänge gestartet, sondern lediglich ein Relaysystem etabliert. Durch die Auslagerung der Großzahl von Kodierungsprozessen auf die Clientseite, wird der Server schon einmal stark entlastet, nur für die Archivierung muss eine Art Dekodierungsprozess (im Sinne der Multimediaschnittstelle ein einfacher Demultiplex- und erneuter Multiplexprozess) gestartet werden, aber diese Einzelheiten werden in Kapitel 4.3 noch näher erklärt werden. Was muss der Server nun alles leisten können? Er muss skalierbar sein, er muss stabil sein, er muss sicher sein und er muss in der Lage sein, mehrere hundert Verbindungen zur gleichen Zeit bedienen zu können. Durch den hohen zu erwartenden Datentransfer im Bereich von mehreren Megabit pro Sekunde (abhängig vom Einsatzgebiet) muss er ein effizientes Datenhaltungskonzept haben, somit sollte die Plattform auch über schnelle und leistungsfähige Festplatten und viel RAM im Bereich von mehreren hundert Megabyte verfügen. Diese Daten sind in hohem Maße vom Einsatzgebiet und der Anzahl der Benutzer abhängig. Da das System erweiterbar und flexibel einsetzbar sein soll, ist es schwierig zu erwartende Maximalwerte in Bezug auf Verkehrsaufkommen, Rechenleistung und benötigtem Arbeitsspeicher zu ermitteln. Ein Ziel bei der Implementierung ist, das System mit Standardhardware einsatzfähig zu machen, also heute zum Beispiel mit einem PC mit etwa 1GHz Taktfrequenz, einer 100 Mbit/s Netzwerkanbindung und etwa 500 Mbyte PC-133 RAM.

Für die Clientseite ergibt sich als Anforderung Nummer eins fast automatisch die Einsetzbarkeit auf Microsofts Windows Betriebssystem, da diese laut [WEBSTATLN_2003] einen Marktanteil von fast 95% haben. Die Statistik bezieht

sich allerdings nur auf gezählte Benutzerbetriebssysteme der Nutzer die auf die eigenen Seiten zugreifen, das trifft aber andererseits auch genau den Endnutzerstamm des zu entwickelnden Systems, da die zu erwartenden Nutzer des Systems zu einem hohen Anteil (zu erwarten sind hier mehr als 80%) Microsoft Betriebssysteme benutzen werden.

Nun zur Leistungsfähigkeit die die Hardware bieten muss. Da die Hauptkodierungsprozesse für das Streaming nicht auf dem Server laufen, sondern auf dem Client, da sonst auch die Übertragung von zum Beispiel großen unkomprimierten Videos von mehreren Gigabyte Länge nur mit einem Gigabit Netzwerk in akzeptabler Zeit möglich wäre, muss der Client in der Lage sein, diese Kodierungs- und Dekodierungsprozesse erledigen zu können. Für das Anschauen eines Videos ohne Audio wäre das also genau ein Dekodierungsprozess. Für eine Videokonferenz mit fünf Teilnehmern wären es dann schon zehn Dekodierungsprozesse und zwei Kodierungsprozesse (für jeden Teilnehmer ein Audio- und ein Videodatenstrom) . Kapitel 3.2 hat gezeigt, dass auf einem 1.7 Ghz Pentium 4 und 256MB RAM unter Volllast ein 16 Sekunden langer hochqualitativer Videodatenstrom in etwa zehn Sekunden kodiert werden kann. Damit liegen die Anforderungen für die Kodierung im Bereich des heute Möglichen. Zu untersuchen bleibt die Dekodierung.

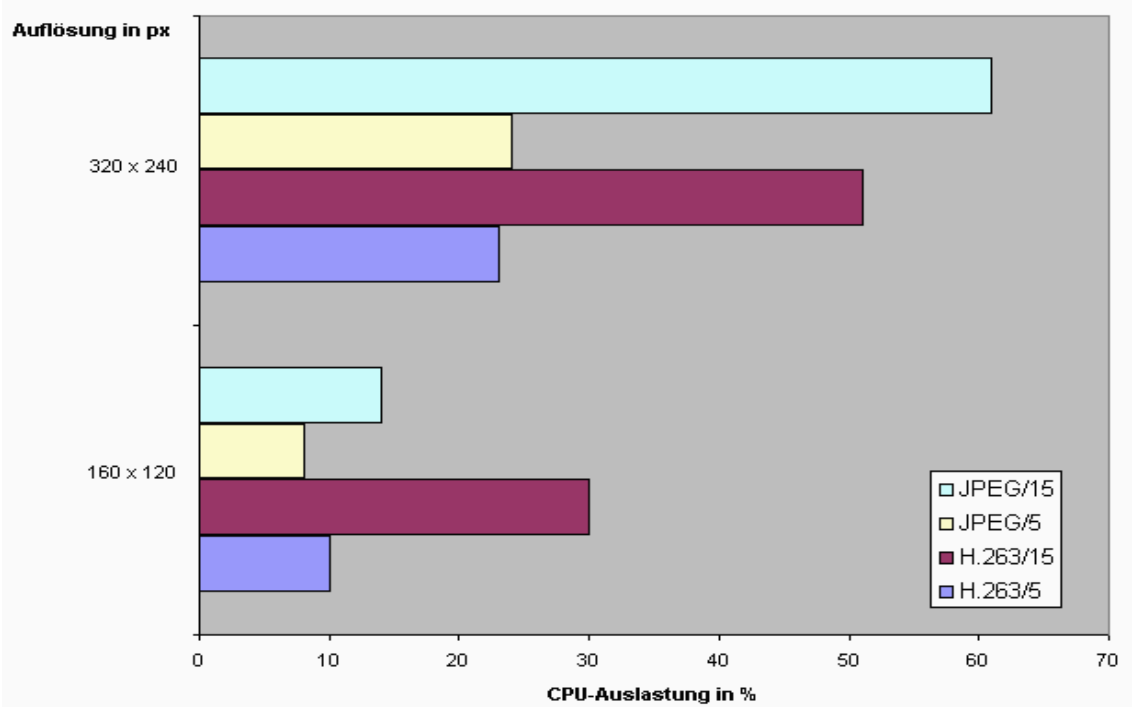

Abbildung 4 Vergleich der CPU Auslastung bei verschiedenen Videocodecs (aus [BEHA_2001])

In Abbildung 4 ist ein Vergleich der CPU-Auslastung aus [BEHA_2001] zu sehen. Dabei wurde ein Pentium 3 mit 700 MHz und 256MB PC-100 SDRAM bei gleichzeitiger Kodierung und Dekodierung verwendet. Dieses Rechnersystem wird somit als Mindestanforderung für den Client definiert. Da die NVACS-Software für die Videokommunikation entwickelt wurde, sind hier Videos (ohne Audiospur) auf diesem Anforderungsniveau erzeugt worden, also mit relativ geringer Auflösung und geringen Frameraten. Für eine angemessene Kommunikation hat sich gezeigt, dass eine Auflösung von 160x120 Bildpunkten und eine Framerate von 5 Bilder/s hinreichend ist. Unter Rücksichtnahme dieser Werte kann also auch davon ausgegangen werden, dass die Anforderungen für die Dekodierung von 5 ankommenden Videoströmen im Bereich des heute Machbaren liegen. Die Audiodatenströme wurden bei allen Test nicht betrachtet, da diese den Rechner bei der Kodierung und Dekodierung sowie die Leistung des Netzwerkes kaum beanspruchen.

In Hinblick auf eine Videokonferenz mit fünf Teilnehmern stellt sich die Frage der Leistungsfähigkeit des Netzwerkes, denn sowohl Server also auch Client müssen in der Lage sein, die Datenströme zu übertragen. Gerade bei Videokommunikation gibt es eine Regel die besagt, dass die Datenbandbreite etwa zwei Drittel der Leistungsbandbreite nicht übersteigen sollte, um eventuelle Jitter- und Peakeffekte in der Leitung zu minimieren.

Die heutigen Internetzugänge bei Privatpersonen lassen sich in DSL-Zugänge, ISDN- und Modemzugänge untergliedern. Nach [WEBSTATLN_2003] fallen auf DSL-Zugänge etwa 53% und auf IDSN- und Modemzugänge etwa 47% der Benutzer (in Deutschland). Ein Audiodatenstrom (zum Beispiel für Internetradio) mit 32kbit/s kann demzufolge von allen Internetnutzern mit einer verfügbaren Bandbreite von mehr als 32 kbit/s empfangen und in Echtzeit abgespielt werden, für Videokonferenzen sollten allerdings Bandbreiten im Bereich von DSL verwendet werden.

Für den Server liegen die Bandbreitenanforderungen höher. Als Berechnungsgrundlage werden folgende zu erwartende Daten abgeschätzt. Angenommen, ein Video-On-Demand Anbieter will über das System eine Online-Videothek aufbauen, dann muss er damit rechnen, dass pro Tag etwa 100 bis 500 Nutzer einen Film herunterladen. Im Abschnitt 3.2, wurde gezeigt, dass mit MPEG-4 Kodierung unterhalb von 1 Mbit/s eine Qualität nahe der von VHS erreichbar ist. Bei

500 Nutzern die sich Videofilme von etwa 1,5 Stunden Länge mit einer Bandbreite von 500 kbit/s über den Tag verteilt herunterladen, erzeugt das ein Volumen von 160 GB pro Tag und eine durchschnittlichen Bandbreite von knapp 16 Mbit/s. Da davon ausgegangen werden muss, dass die Downloads nicht gut über den Tag verteilt liegen, sondern etwa bei 50% zwischen 20 Uhr und 24 Uhr, errechnet sich eine Bandbreite von 45 Mbit/s während dieser Zeit. Werden hier zur Sicherheit noch eventuelle Spitzen mit eingerechnet, sollte ein solches System mit mindestens 100MBit/s Netzanbindung auslegt werden.

3.4. Untersuchung von Benutzerparametern

Zu den benutzerspezifischen Parametern gehören unter anderem Aspekte der Bedienbarkeit, der Installation, der Qualität der Video- und Audioströme, also insgesamt Anwendbarkeit und Qualität. Auch müssen kommerzielle Aspekte berücksichtigt werden, also zum Beispiel die Unterhaltungskosten des Systems.
Die Unterhaltungskosten des Systems beziehen sich fast ausschließlich auf den Server. Wie in Kapitel 3.3. schon angedeutet, wird ein Rechner mit hohen Hardwareanforderungen (mehr als 500 MByte RAM) und ein skalierbares Netzwerk (n*100 Mbit/s) benötigt. Wer allerdings ein solches System, wie in Kapitel 3.3. beschrieben, betreiben will, muss sich mit diesen Kosten auf jeden Fall auseinandersetzen. Vielmehr spielen jedoch Lizenzkosten für eingesetzte Software eine Rolle. Da das System jedoch in Java implementiert wird und die Benutzung eines Unixsystems wie etwa Linux empfohlen wird, sind die Lizenzkosten gleich Null. In gängigen Untersuchen zur Total Cost of Ownership bei OpenSource Software [CTOS_2003] wird immer wieder gesagt, dass OpenSource erst dann wirklich günstig wird, wenn auch das Administration- und Wartungsknowhow gesichert ist, also wenn zum Beispiel eine Firma auch einen Administrator hat, der damit umgehen kann. Um diesen Effekt zu kompensieren muss das System einfach zu warten und zu administrieren sein. Das sollte in etwa so aussehen, dass die Software (also primär der Server) heruntergeladen werden und die Software auf einem beliebigen Rechner mit einem einzigen Befehl gestartet werden kann, möglichst ohne Anpassung und Konfiguration des Systems.
Für die Clientseite bedeutet das, dass über eine einfache Schnittstelle ohne

Installation von Zusatzsoftware das System bedient werden kann, und das möglichst von jedem beliebigen die Mindestanforderungen erfüllenden Rechner aus. Diese Anforderung legt für die Clientseite eine Webschnittstelle fest, also eine Benutzerschnittstelle auf Browserbasis. Da ein Browser im Allgemeinen keine Videostreams wiedergeben kann, oder zumindest nicht auf standardisierte Art und Weise muss eine Art Plugin und Schnittstelle für die Streamingdaten geschaffen werden. Diese Schnittstelle sollte möglichst ohne Installation von Zusatzsoftware zugänglich sein. Da die Clientsoftware auch plattformunabhängig sein soll, empfiehlt sich hier wiederum die Javatechnologie, die es ermöglicht in jedem Browser kleine Java-Anwendungen (Applets) laufen zu lassen. Allerdings hat Java in seiner Grundform, wie sie von SUN ausgeliefert wird, keine nennenswerten Multimediafähigkeiten. Eine Lösung des Problems bietet das Java Media Framework von SUN [JMFLN_2003], das auch in [BEHA_2001] zum Einsatz kommt. Dieses Framework beinhaltet Möglichkeiten zum Aufzeichnen von Audio- und Videodatenströmen und umfasst eine Menge von Audio- und Videocodecs (unter anderem H.363, MJPEG, MPEG, GSM, G.723, MP3), allerdings kein MPEG-4. Für das rechenintensive Kodieren von Videos, gibt es das Java Media Framework in speziellen Versionen für die Betriebssysteme Microsoft Windows, Sun Solaris und Linux. Bei diesen Versionen werden Aufnahmegeräte unterstützt und die Kodierungsarbeiten (nicht bei allen Codecs) mit nativen Codecs erledigt. In der Version für alle Plattformen bietet das Framework Unterstützung für unter Anderem H.263, MJPEG, JPEG, GSM und G.723. Das bedeutet, dass nicht nur bei der Konzeption sondern auch beim Einsatz des Systems gesteigerte Aufmerksamkeit auf die Plattformunabhängigkeit der verwendeten Codecs gelegt werden muss.

Eine weitere Rolle für den Benutzer spielt die Geschwindigkeit und Gestaltung des Systems. Für den Server bedeutet das, dass die Zugriffe gecached werden müssen und die Logik schnell reagieren (im Millisekunden-Bereich) muss. Die Gestaltung der Benutzerschnittstelle spielt nicht nur für den Endbenutzer des Systems eine Rolle, sondern auch für den Anbieter des Systems, der sich dem Benutzer in seinem Corporate Identity Look präsentieren will.

Es muss also eine Schnittstelle geschaffen werden, die es erlaubt, auf einfachste Weise das Look and Feel der Benutzerschnittstelle anzupassen. Dieses wird mit der schon erwähnten XML/XSL-Technologie erreicht, in dem für die Benutzerschnittstelle eine einfache, klar strukturierte und eindeutige DTD entworfen wird, die mit wenigen

Änderungen am Stylesheet angepasst werden kann. Die Trägertechnologie, die dieses ermöglicht, wird im Kapitel 5 behandelt.
Ein Dateisystem hat die Eigenschaft, dass Nutzer Dateien darin ablegen können. Nun muss auch geklärt werden, wie viele Dateien ein Nutzer darin ablegen können darf. Deswegen muss es eindeutige Speicherplatzprofile geben, die der Administrator jedem einzelnen Nutzer zuweisen kann. Dieses Thema wird allerdings im Kapitel 4.3. noch näher behandelt.
Der nächste Punkt wäre dann die Qualität der Audio- und Videocodecs die das System bieten muss. Das Java Media Framework bietet H.263 und MPEG-1 als Codecs. Für den Einsatz in Videokonferenzen ist der H.263 für Video und GSM für Audio geeignet, für den Einsatz im Entertainmentbereich stehen MJPEG Video und MPEG-1 Audio zur Verfügung.
Bei den Codecs gibt es in der speziellen Versionen für Windows die Möglichkeit, auch nativ installierte Codecs wie MP3 oder MPEG-4 zu benutzen, in den Versionen für Linux und Solaris fehlt diese Möglichkeit jedoch. Da das Framework jedoch auch die Möglichkeiten der Erweiterung bietet, schließt sich die Möglichkeit der Entwicklung von Java basierten Audio- und Videocodecs nicht aus.

3.5. zukunftsorientierte Designparameter und Techniken

Wie schon in Kapitel 2.3 kurz erwähnt, wird für die Protokollschnittstelle XML benutzt. Als Transportmedium dient das Internet, als Transportprotokoll somit HTTP und TCP. Diese Techniken sind flexibel und gelten als erweiterbar und zukunftsträchtig. Besonders XML hat sich in den letzten Jahren zu einem Standard für internetbasierten Datenaustausch entwickelt. Im Umfang dieser Arbeit wird eine Referenzimplementation des Systems erstellt. Dazu kommt als Programmiersprache Java zum Einsatz, um zum Einen die Plattformunabhängigkeit zu gewährleisten und zum Anderen, weil es für Java Bibliotheken gibt, die alle bisher genannten Technologien (XML, Http, Multimedia) unterstützen.
Die Applikation wird eine Webapplikation sein, die zum einen eine anpassbare Benutzerschnittstelle und zum Anderen einen XML-basierten Datenaustausch ermöglicht. Daher liegt es nahe, als Strukturspräche für Daten XML und für die Generierung der einzelnen Schnittstellen XSL/XSLT zu benutzen. Welche

Werkzeuge und Methoden im Einzelnen dafür benutzt werden, wird im Kapitel 5 noch näher behandelt.

Da das System eine Benutzerverwaltung und Datenverwaltung implementieren muss, ist eine Datenbank unumgänglich. Damit die Applikation skalierbar und hoch verfügbar bleibt, empfiehlt es sich, bei der Implementation Techniken zu nutzen, die es ermöglichen die Last der Applikation zu verteilen. Im Java Umfeld haben sich so genannte Applicationserver und EJB-Container durchgesetzt, die diese Anforderungen erfüllen. Die Datenhaltung kann durch EJB (Enterprise Java Beans) [DEMICHIEL_2003] gekapselt werden und bietet zugleich die Möglichkeit zur Verteilung der Daten und der Last. Der Applikationserver übernimmt dabei die Verteilung der Last über die verfügbaren ServletEngines und EJB-Container. Somit wird ein dreistufiges Design im Sinne des MVC-Konzeptes (Model-View-Control) aufgebaut. Die Model-Ebene wird durch EJB CMP (Container Managed Persistance), die Control-Ebene durch Java-Servlets und die View-Ebene durch XML/XSL realisiert. Alle Komponenten werden bei Bedarf parallelisiert und auf unterschiedliche Rechner verteilt werden. Die Schnittstelle zwischen den Techniken kann wahlweise über CORBA oder Java RMI realisiert werden.

In den letzten zwei Jahren hat sich noch eine weitere Technik im Zusammenhang mit XML und Internet entwickelt. Die Rede ist von den so genannten Webservices, einer Technik, die das Aufrufen von Funktionen über das Internet ermöglicht. Die Technik funktioniert dabei so ähnlich wie das herkömmliche RPC (Remote Procedure Call), nur das bei den Webservices die Aufrufschnittstelle XML und das Transportprotokoll meist HTTP ist. Die Spezifikation lässt dabei auch andere Transportprotokolle zu.

Da die Technik der Webservices der konzipierten Technik der zu entwickelnden Applikation ähnlich ist, liegt es nahe, die Schnittstellen mit Webservices zu implementieren. In dieser Arbeit wird zunächst noch ein eigenes XML-basiertes Schnittstellenprotokoll spezifiziert, da durch Streitigkeiten unter den Webservices-Standardisierungsparteien im Moment noch keine Entwicklungsrichtung abzusehen ist. Eine spätere Erweiterung des Protokolls auf Webservices stellt kein Problem dar, da lediglich die Schnittstellenbeschreibung (WSDL – Web Services Description Language) generiert werden muss.

4. Spezifikation

4.1. Strukturierung des Systems und Festlegung der Schnittstellen

In diesem Abschnitt wird zunächst die Struktur des Systems und die Kommunikation der Teilkomponenten festgelegt. Das System ist ein Client-Server-System, besteht daher aus einem oder mehreren Servern und mehreren Clients. Die Clients sind dabei die Nutzer des Dateisystems und der Server das Dateisystem selbst. In Abbildung 5 wird die Grobstruktur des Systems beschrieben. Auf dem Clientrechner können eine oder mehrere Anwendungen (also Clients) laufen, die Daten vom Server empfangen und zum Server schicken können. Die Standardschnittstelle ist eine HTML-Schnittstelle über HTTP, über die alle Verwaltungsfunktionen mit Hilfe eines Browsers genutzt werden können. Für Fremdapplikationen (Clients) dient eine XML-Schnittstelle für das Empfangen von Daten. Die Befehle werden dabei alle über das URL-Schema abgesetzt. Eine Auflistung aller zu implementierenden Befehle folgt noch (Kapitel 4.4).

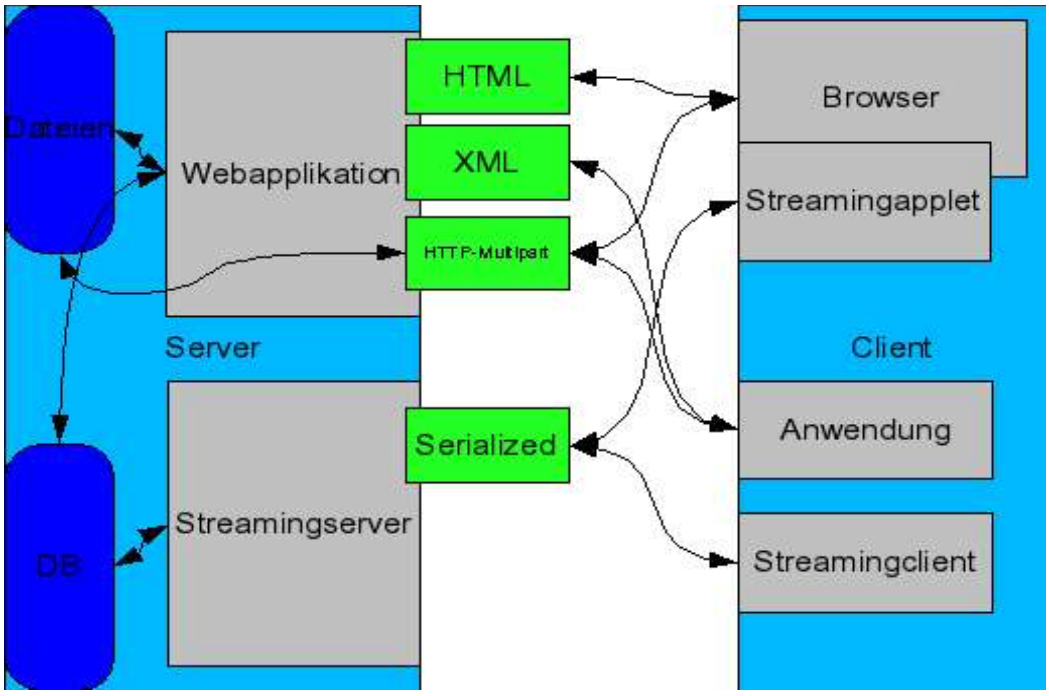

Abbildung 5 Struktur des Client Server Systems

Das Transportprotokoll ist HTTP. Um Dateien zum System hochzuladen, wird das HTTP-Multipartprotokoll benutzt. Eine Methode eine Datei hochzuladen ist somit ein

HTML-Formular der Form:

```
<form method="POST" ENCTYPE="multipart/form-data" action="upload">
        <input type=“file“ name=“file“>
</form>
```

Die genaue Spezifikation aller Parameter folgt in Kapitel 4.4. Um Dateien hochzuladen, muss allerdings kein HTML verwendet werden, es existieren Bibliotheken, die in der Lage sind, HTTP-Multipartrequests zu erstellen und damit Dateien hochzuladen. Pro Request kann jedoch nur eine Datei hochgeladen werden, da für diese noch ein Hash berechnet werden muss und danach ein Berechtigungseintrag in der Datenbank angelegt werden muss. Für den Streamingserver gibt es genau eine Schnittstelle. Die Schnittstelle Serialized in Abbildung 5 bedeutet, dass der Streamingserver nur Serialisierte Objekte annimmt. Das Streamingprotokoll ist damit Objektorientiert. Serialisierte Objekte sind dabei Objekte, die mit Hilfe eines Serialisierungsprotokolls in ein textbasiertes Format umgewandelt worden sind, um diese dann an anderen Stellen wieder zu deserialisieren und weiter zu verwenden. Da die Implementierung in Java erfolgt, wird das Serialisierungsprotokoll von Java benutzt. Das Protokoll kann auch in anderen Programmiersprachen verwendet werden, da es textbasiert ist und alle nötigen Schnittstellendaten der Objekte enthält. Ein großer Vorteil des Protokolls bei der Verwendung im Streamingserver ist die Unabhängigkeit von den verschiedenen Medienformaten, mehr dazu in Kapitel 4.3.
Der Ablauf eines Verwaltungsaufrufs sieht immer so aus:

Request (Befehl,Parameter,Daten) --> Server --> Response (XML|HTML|...)

Alle Funktionen werden nach dem Request-Response Prinzip abgehandelt. Dabei muss allerdings immer sicher gestellt werden, dass der Nutzer auch authentifiziert ist. Will ein Nutzer eine Datei hochstellen, so muss folgender Ablauf sicher gestellt werden:

Client:

Request mit Datei, Dateiname, Benutzername, Passwort

Server:

nimmt Request entgegen

Server:

überprüft Authentizität des Nutzers

bei Authentizität

Server:

speichert Datei im eigenen Dateisystem mit global eindeutiger Vorgangsnummer

Server:

berechnet Hashwert für die Dateien

Server:

speichert Root-Berechtigung der Datei für den Benutzer in der Datenbank

Datenbank:

falls Hash schon vorhanden -->Fehler

Server:

falls alles OK, schicke Response mit Nachricht OK

Soll jedoch ein Stream hochgestellt werden, so wird der Request über den Streamingserver ausgelöst:

Streamingclient:

Request mit Benutzername, Passwort, Streamname (Channel), Archivierungsbefehl

Streamingserver:

nimmt Request entgegen

Streamingserver:

überprüft Authentizität

bei Authentizität

Streamingserver:

legt Berechtigung für einen Stream unter angegebenem Streamnamen an, da kein Hash berechnet werden kann, erzeugt er global eindeutige Vorgangsnummer

Streamingserver:

beginnt Daten (Objekte) entgegen zunehmen und speichert bekannte Medienformate bei gesetztem Archivierungsflag im Request in eine Datei

Streamingserver:
bei Ende des Streams wird Berechtigung gelöscht und für eventuell aufgezeichnete Datei der Hash berechnet und eine Berechtigung angelegt

Bei Nichtauthentizität des Nutzers wird in allen Fällen sofort die Verbindung abgebrochen. Wie diese Aktionen in anderen Implementationen gelöst werden, wird hier nicht festgelegt, einzig der Charakter der Funktionen und Befehle sind festgelegt.
Um eine Datei herunterzuladen, wird der download-Befehl des Systems benutzt, danach wird die Datei bei Berechtigung per HTTP ausgeliefert. Soll ein Stream empfangen werden, so wird der stream-Befehl des Systems benutzt, die Einzelheiten dazu werden in den Kapiteln 4.3. und 4.4. noch näher beschrieben.
Die zentrale Verwaltungseinheit des Systems ist die Berechtigung. Eine Berechtigung besteht aus folgenden Elementen:

<file>
<name>
Der Name der Datei oder der Channelname
</name>
<hash>
Der Hash oder die eindeutige Vorgangsnummer
</hash>
<downloadurl>
Die URL für den Download der Datei.
- bei einem Stream die URL des Streamingservers
corona://adresse:port
</downloadurl>
<parent>
Der Ursprung der Datei für Versionisierung.
- hier wird der Hash der Ursprungsdatei eingetragen
</parent>
<size>
Die Größe der Datei in Bytes.

- bei einem Stream wird '-1' eingetragen

</size>

<format>

Das Format der Datei oder des Streams.

- vorgeschlagen ist der Mimetype der Datei

oder des Formats des Streams

</format>

<type>

Der Typ (Stream oder Datei)

0 - Datei

1 - Stream

</type>

<owner>

Systeminternes Kürzel des Besitzers.

- bei systemübergreifendem Besitzer muss die

Adresse des Systems nach Email Addressing Specification

angehängt werden: user@host

</owner>

<upload-date>

Datum des Uploads in UTC

</upload-date>

<permission>

<type>

Berechtigungstyp

0 - Besitzer

1 - Administrator

2 - Normaler Benutzer

3 - Gast

weitere Typen sind offen und abhängig von der Implementation

</type>

<user>

Systeminterner Nutzername

</user>

<timeout>

Ablaufdatum für die Benutzung nach UTC

</timeout>

<creation-date>

Datum der Erstellung der Berechtigung in UTC

</creation-date>

<usecount>

Verbleibende Anzahl der Benutzungen

wird bei jeder Benutzung bis '0' um eins verringert.

Bei unendlicher Benutzung muss '-1' eingetragen werden,es wird

auch hier bei jeder Benutzung um eins verringert.

</usecount>

</permission>

</file>

Die Berechtigung wird in einer Datenbank unter dem primären Key 'Hash' verwaltet. Das Format der Berechtigung ist XML, eine DTD wird jedoch nicht benötigt, da sie nur vom System erstellt wird und somit nicht validiert werden muss.
Alle Verwaltungsbefehle nutzen für die Ausgabe von Berechtigungen dieses XML-Format in der Form:

<file/>

einzeln oder als Liste von Berechtigungen

<filelist>

<file/>

<file/>

<file/>

...

</filelist>

Eine Strukturierung im Sinne von Ordnern oder Kategorien ist nicht vorgesehen und muss von darüber liegenden Diensten wie Client oder Betriebssystem erledigt werden.

4.2. Eindeutige Identifikation der Daten durch einen Hashalgorithmus

In diesem Abschnitt wird ein geeigneter Hashalgorithmus entwickelt. Der Hashalgorithmus ist austauschbar und deshalb hier in einer Referenzimplementation spezifiert.
Um die Dateien eindeutig zu identifizieren muss ein Verfahren entwickelt werden, bei dem sichergestellt werden kann, dass der Hash einer Datei die wesentlichen Charakteristika jeder beliebigen Datei widerspiegelt.
Ein solches Verfahren muss jedoch auch gewährleisten, dass kein Hash doppelt mit zwei unterschiedlichen Dateien berechnet werden kann, es muss also die Schlüsselmenge (Hashmenge) mindestens so groß gestaltet werden, wie sie Dateien identifizieren können soll. Da es jedoch unverhältnismäßig ist, für eine 500 byte große Datei einen 1024 byte Schlüssel zu verwalten, muss der Schlüssel eine von der Größe der Datei abhängige Länge haben.
Die Datei wird in eine feste Anzahl von Blöcken variabler Länge unterteilt und die Hashes auf der Grundlage der einzelnen Blöcke berechnet. Welche Länge die Blöcke dabei haben, hängt dann auch vom Einsatzgebiet des Systems ab. Soll eine Online-Videothek mit großen Dateien (mehrere 100 MB) erstellt werden, so wird bei geringer Blockanzahl eine große Blocklänge entstehen, soll jedoch ein Verwaltungssystem für E-Books erstellt werden, wird bei großer Blockanzahl eine zu kleine Blocklänge entstehen. Das Blockverfahren sichert auch eine gewisse Stabilität gegenüber der Verfälschung kleiner Stellen, da sich der Hash nur im jeweiligen veränderten Block ändert.
Der nächste Punkt ist, wie gerade schon erwähnt, die Gewährleistung der Identität bei geringfügiger Veränderung einer Datei. Dazu muss zunächst untersucht werden, was alles verändert werden kann. Unterschieden wird hier zwischen binären Daten wie Programmen, multimedialen Daten wie Audio und Video und textuellem Inhalt.

Änderung	*Text*	*Audio/Video*	*Binärdaten*
Ändern der Länge	-Entfernen von Leerzeichen	-Abschneiden des Abspanns bei Videos -ändern der Bitrate des Codecs	Im Allgemeinen nicht möglich
Herausschneiden von Fragmenten	-Textfragmente weglassen (meist eher unerwünscht)	-Abschneiden des Abspanns bei Videos -Weglassen der letzten Sekunden bei Audio	Im Allgemeinen nicht möglich
Verfälschen durch Einfügen von Rauschen	-umwandeln von Tabulatoren in Leerzeichen	-Rauschen generell möglich	Im Allgemeinen nicht möglich
Gezieltes Editieren	-Weglassen von Copyrightinformationen, Autor, oder unwichtigen Informationen	-Herausschneiden oder gezieltes verändern jedes 25ten Frames bei Video oder Audio	Im Allgemeinen nicht möglich
Komprimieren	-Zippen eines Textes, hat den Nachteil, dass Text immer erst dekomprimiert werden muss	-verringert die Größe zwar nicht, aber die Datenorganisation, Dekomprimierung auch hier ein Nachteil	-Zippen eines Programmes, Dekomprimierung auch hier ein Nachteil
Verschlüsseln	-Verschlüsseln eines Textes, muss aber vor Lesen erst wieder entschlüsselt werden	-Verschlüsseln von Audio und Video, muss aber vor Wiedergabe erst wieder entschlüsselt werden	-Verschlüsseln von Programmen, muss aber vor Ausführung erst wieder entschlüsselt werden

Tabelle 6 Verfälschungsmethoden bei Daten

Tabelle 6 zeigt verschiedene Methoden der Verfälschung. Die effizientesten Methoden sind die Komprimierung der Daten, die zwar nicht immer die Größe reduzieren kann, jedoch bei den meisten Komprimierungsalgorithmen die Organisation der Daten ändert, und die Verschlüsselung, die je nach Verfahren ebenfalls die Datenorganisation entropiert.

Das Verfahren muss also damit rechnen, dass die Daten durcheinander „gewürfelt“ werden, darf sich also nicht an bestimmten Daten orientieren. Als Lösung wird die Interpretation der Daten als Funktion, ähnlich wie bei Bildern und Videos, eingesetzt. Somit ist eine Datei eine Menge von Polynomen mit endlich vielen Stützpunkten. Wird jetzt von einer Restrukturierung der Daten bei einer Verfälschung ausgegangen,

so würde dies die Veränderung im schlechtesten Falle aller Polynome bedeuten, jedoch kann auch bei der Restrukturierung noch von linearen Zusammenhängen zwischen den Daten ausgegangen werden. Da nicht vorhersehbar ist, wie das System später umgangen werden könnte, soll hier nur der Datenverlauf, der charakteristische Verlauf der Daten in einer bestimmten Umgebung betrachtet werden. Da Daten verrauscht werden können, wird die Funktion geglättet. So werden nur die charakteristischen Verläufe im Hash gespeichert.
Der Algorithmus für die Hashberechnung sieht dann so aus:

```
Blocklänge =  Dateilänge / Anzahl der Blöcke
while (true) {
        Lese nächsten Block aus Datei
        Wenn kein Block mehr --> Ende
        BlockWert = 0;
        for (g = 0; g < Blocklänge; g++) {
                BlockWert = BlockWert + Block.Zeichen[g];
        }
        Glättungswert=Blockwert;
        AlterGlättungswert=0;
        Urwert=Glättungswert;
        for (l = 0; l < Glättungsfaktor; l++) {
                Glättungswert =
                        ((AlterGlättungswert + Urwert) / 2)
                                modulo Blocklänge;
                AlterGlättungswert = Glättungswert;
                BlockWert += (Glättungswert + BlockWert) / 2;
        }
        hash += "" + BlockWert;
}
```

Der Algorithmus behandelt die Stützpunkte der Funktion als addierte Zahl, so kann der Verlauf der Funktion verändert werden, der Hash bleibt aber stabil, denn 10+100+4 oder 4+10+100 liefern die gleichen Ergebnisse. Die Glättung der Funktion geschieht in der Glättungsschleife. Hier wird jeweils vom voran gegangenen Durchlauf der Mittelwert gebildet. Bei dem konkreten Beispiel 10+100+4 wird von

einer Blocklänge von 3 ausgegangen:

1. Glättungswert=114, Alterglättungswert=0
   Glättungswert=(0+114)/2 mod 3=57 mod 3=0;Blockwert=0+114/2=57;
2. Glättungswert=(0+114)/2 mod 3=57 mod 3=0;Blockwert=(0+57)/2=29;
3. ...

An diesem Beispiel wird klar, dass nach Möglichkeit größere Blocklängen als 3 verwendet werden sollten, da durch die modulo-Rechnung sonst schnell Null-Glättungswerte entstehen, die über alle Glättungsphasen erhalten bleiben und bei genügend hohem Glättungsfaktor den Blockwert auf Null glätten. Eine mittlere Blocklänge ergibt sich bei 8 Blöcken pro Datei. Bei einer 1000 byte großen Datei, wäre das eine Blocklänge von 125. Im Einsatz mit großen (mehrere hundert MB) Dateien sollte die Blockanzahl dennoch nur im Notfall verändert werden. Zum Beispiel kann, wenn das erste mal ein Hash doppelt berechnet wurde, immer noch auf eine höhere Blockanzahl umgestiegen werden, da die Hashes der bereits bestehenden Dateien davon unberührt bleiben.
Die Blockanzahl sollte nicht nur wegen der nötigen Stringvergleiche so niedrig wie nötig gehalten werden, sondern auch weil der Algorithmus den rechenaufwendigen Glättungsalgorithmus pro Block anwendet. Umso mehr Blöcke also verwendet werden, umso rechenaufwendiger wird die Hashberechnung, bei großen Dateien im Bereich von mehreren hundert Megabyte wirkt sich das merkbar (exponentiell) auf die Laufzeit des Algorithmus aus. Der Algorithmus kann jederzeit modifiziert werden, es sollte jedoch darauf geachtet werden, dass eventuelle Änderungen keine schon vorhanden Hashes doppelt berechnen.

4.3. Spezifikation des Streamingservers

Der Streamingserver ist die Besonderheit des Dateisystems, da durch ihn ermöglicht wird, auch Streams wie Dateien zu behandeln. Herkömmliche Streamingserver laufen meist auf dem Rechner, der die Daten streamen will. Der Streamingserver dieses Systems funktioniert nach dem Publish/Subscribe-Prinzip, das heißt, dass ein Nutzer erst Daten zum Server streamt und diese Daten dann von anderen Nutzern herunter gestreamt werden können. Das erspart dem Anbieter des Streams in aller erster Linie Bandbreite, denn er muss weder allen Nutzer den Stream anbieten, noch

muss er Multicastfähigkeit besitzen, um Bandbreite zu sparen. Da Multicast zwar eine fortschrittliche Entwicklung ist, aber über das Internet für Privatpersonen kaum einsetzbar ist, wird hier komplett auf Multicast verzichtet. Das liegt unter anderem auch daran, dass der Streamingserver einzig über TCP streamt.

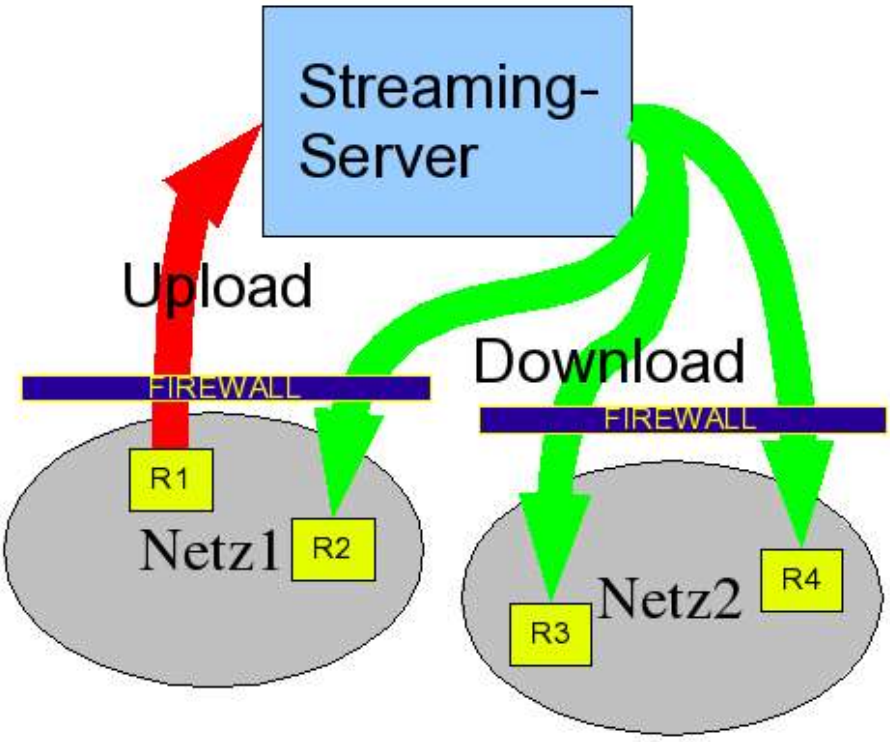

Abbildung 6 Prinzip des Streamingservers

In Abbildung 6 ist das Prinzip des virtuellen Multicastings erkennbar. Durch dieses Prinzip ist es möglich von jedem beliebigen Punkt in jedem beliebigen Netzwerk, das eine Route zum Streamingserver hat, einen Stream hoch zustellen und diesen auch abzurufen. Einzige Voraussetzung dafür ist die Erreichbarkeit des Servers über das Internet. Um dieses zu ermöglichen, musste auf UDP und Multicast verzichtet werden und stattdessen TCP verwendet werden. Da die Netze aber immer besser und leistungsfähiger werden, stellt das keinen Performanzverlust dar, sondern bietet stattdessen noch weitere Vorteile wie den Einsatz von Verschlüsselung via SSL.
Der Streamingserver ist nicht auf bestimmte Medienformate fixiert, sondern realisiert in lediglich eine Weiterleitung der Streams (Relay). Dabei werden ankommende Daten entgegen genommen und an alle Empfänger weitergeleitet. Dadurch muss der Rechner, auf dem der Streamingserver läuft eine breitbandige Netzanbindung

(abhängig vom Einsatzgebiet im Bereich von 100 Mbit/s) besitzen, da er ähnlich einem Multicastrouter die Daten kopiert und an verschiedene Knoten weiterleitet. Der Server wird in Java und verwendet zur Übertragung der Daten das Objekt-Serialisierungsprotokoll von Java. Hier stellt sich eine große Herausforderung in Bezug auf Speichermanagement an den Server, da er in der Lage sein muss, mehrere hundert Objekte gleichzeitig zu empfangen, zu kopieren und weiter zu versenden. Rechenaufwand produziert der Server nur beim Parsing des Objektserialisierungprotokolls. Die Objektserialisierung hat den Vorteil, das ein effizienter Parsingalgorithmus schon existiert und somit die Unabhängigkeit von bestimmten Formaten und die Erweiterbarkeit des Protokolls gewährleistet ist.

Neben einem Standardprotokoll für die Auslieferung streamingspezifischer Parameter, welches im Folgenden noch erklärt wird, können noch diverse Erweiterungen implementiert werden, ohne dass das Standardprotokoll seine Gültigkeit verliert.

Die Schnittstelle des Servers ist einfach und effizient. Ein Client verbindet sich zum Server und schickt ein Protokollobjekt, das eine Java-Hashtable enthält. Die Hashtable enthält dabei folgende Einträge. Die in Anführungszeichen geschriebenen Namen sind dabei die Zeichenketten, die vom System erwartet werden.

TYPE_ID - „type“

der Typ des Request:

„upstream“	für den Upload eines Streams
„downstream“	für den Download eines Streams

CHANNEL - „channel“

der Name des Channels (des Streams)

HASH - „hash“

der Hash oder die Vorgangsnummer des Streams für einen Download

FORMAT - „format“

das Format des Streams, sollte nach Möglichkeit der Mime-Type sein

PASSWORD - „password“

das Passwort des Benutzers

SAVE - „save“

ein Flag zum Speichern des Streams

USER - „user“

der Benutzername

Nach dem Request kann der Client dann die Datenobjekte schicken. Inzwischen überprüft der Server die Authentizität des Nutzers und legt bei Erfolg eine Berechtigung für den Stream an, die bis zum Ende des Streams erhalten bleibt. Ist der Benutzer nicht erfolgreich authentifiziert worden, so wird die Verbindung sofort unterbrochen, um den Server nicht unnötig zu belasten. Der Server schickt nach dem erfolgreichen Anlegen der Berechtigung eine Erfolgsmeldung und eine Fehlermeldung anderenfalls im Protokollobjekt zurück. Der Rückgabewert befindet sich dann im Feld RET_CODE - „ret_code“ mit der jeweiligen Fehlermeldung. Die ausführliche Meldung steht im Feld RET_TEXT - „ret_text“.
Ist das SAVE-Flag gesetzt (der Wert wird ignoriert, nur der Schlüssel muss vorhanden sein), so versucht der Streamingserver das Format des Streams zu erkennen und in einer Datei <CHANNEL>.avi als Video zu speichern. Da dieses nicht immer gelingt (durch eventuell auf dem Server nicht vorhandene Codecs), empfiehlt es sich eher, den Stream hoch zustellen, gleichzeitig vom Server mit einem Client zu empfangen und lokal zu speichern, da hier alle eventuell notwendigen Codecs vorhanden sind. Das erfordert zwar die doppelte Bandbreite, entlastet aber den Server.
Das Archivieren auf dem Server verlangt alle Codecs, die auf dem Client zur Komprimierung verwendet werden. Nach Ende des Streams wird für die gespeicherte Datei ein Hash berechnet und eine Berechtigung für den Nutzer angelegt.
Da der Streamingserver nur Objekte weiterleitet, muss ihm auch die Objektschnittstelle bekannt sein, wenn also weitere Protokolle entwickelt werden sollen, so müssen diese auch auf dem Server hinterlegt werden. Ein Standardprotokoll für die Übertragung des JMF-internen Media-Buffers wurde bereits implementiert. So ist es möglich über das JMF Audio und Video zu übertragen. Im Kapitel 5 wird noch auf Einzelheiten wie zum Beispiel Klassennamen eingegangen werden.

4.4. Spezifikation der Verwaltungsfunktionen

Die Hauptfunktionalität für den Benutzer wird durch die Verwaltungsfunktionen

definiert. Diese Funktionen stellen die direkte Interaktion zwischen Nutzer und Dateisystem sowie zwischen Client und Server dar. Die Funktionen werden alle über ein HTTP-Request aufgerufen, möglich sind auch andere Protokolle, da die Kapselung der Befehle über URLs geschieht. Dabei werden alle wichtigen Parameter in die URL kodiert. Ob nun HTTP-POST oder HTTP-GET verwendet wird, ist dem System egal, es muss auf jeden Fall möglich sein, über einen URL-Encoded Befehl alle wichtigen Verwaltungsfunktionen zu erreichen.
Einzig der Upload einer Datei wird über einen HTTP-Multipartrequest realisiert. Im Folgenden werden alle Funktionen aufgelistet, Erweiterungen sind möglich:

4.4.1. Benutzerverwaltung

Anlegen eines Benutzers:

Hier wird ein neuer Benutzer angelegt. Je nach Systemkonfiguration darf das entweder jeder Nutzer oder nur ein Administrator tun.
Parameter:

user=Nutzername des anzulegenden Nutzers
password=Passwort des anzulegenden Nutzers
password2=Wiederholung des Passwortes
firstname=Vorname des anzulegenden Nutzers
lastname=Nachname des anzulegenden Nutzers
group=Gruppenzugehörigkeit des anzulegenden Nutzers
auth_user=Nutzer der den Nutzer anlegt
auth_password=Passwort des anlegenden Nutzers

Befehl:

create_user

Löschen eines Benutzers:

Hier wird ein Nutzer gelöscht. Die Mindestanforderung ist, dass der Nutzer sich selbst authentifiziert. Sonst kann nur ein Administrator einen Nutzer löschen. Die Dateien des Nutzers werden nicht gelöscht, da bereits vergebene Berechtigungen davon abhängen könnten. Die Dateien können später manuell gelöscht werden, da der Benutzername im Dateinamen steht. Sollen Dateien dennoch gelöscht werden, muss eine kaskadierende Löschfunktion implementiert werden, Teil der Spezifikation

ist diese jedoch nicht.

Parameter:

user=zu löschender Nutzern

auth_user=löschender Nutzern

auth_password=Passwort des löschenden Nutzers

Befehl:

delete_user

Ändern eines Benutzers:

Hier können die Daten eines Nutzers, wie Name und Passwort geändert werden. Ein Administrator darf alle Daten ändern. Der Nutzer selber kann alle Daten bis auf seine Gruppenzugehörigkeit ändern.

Parameter:

user=Nutzername des zu ändernden Nutzers

auth_user=ändernder Nutzern

auth_password=Passwort des ändernden Nutzers

firstname=Vorname des zu ändernden Nutzers

lastname=Nachname des zu ändernden Nutzers

group=Gruppe des zu ändernden Nutzers

password=Passwort des zu ändernden Nutzers

password2=Passwortwiederholung

Befehl:

edit_user

Anlegen einer Gruppe:

Nur ein Administrator darf eine Gruppe anlegen.

Parameter:

auth_user=anlegender Nutzer

auth_password=Passwort des anlegenden Nutzers

group=Name der Gruppe

base_group=Basisgruppe, von der die Rechte übernommen werden

can_create_groups=Kann Gruppen anlegen

can_create_users=Kann Nutzer Anlegen

can_delete_users=Kann Nutzer Löschen

can_view_all_users=Kann die Daten und Dateien aller Nutzer sehen

can_change_all_users=Kann die Daten und Dateien aller Nutzer ändern
can_view_all_permissions=Kann alle Berechtigungen sehen
can_edit_all_permissions=Kann alle Berechtigungen ändern
can_create_permissions=Kann Berechtigungen erstellen
can_edit_permissions=Kann Berechtigungen editieren
allowed_upload_bandwith=maximale Bandbreite zum Hochladen
allowed_download_bandwith=maximale Bandbreite zum Herunterladen
allowed_upload_number=maximale Anzahl der Uploads
allowed_download_number=maximale Anzahl der Downloads
allowed_number_upload_connections=maximale Verbindungen
allowed_number_download_connections=maximale Verbindungen
allowed_diskspace=maximaler Speicherplatz der genutzt werden kann
reset_time=Abrechnungszeitraum

Befehl:

create_group

Löschen einer Gruppe:

Mit diesem Befehl kann ein Administrator Gruppen löschen.

Parameter:

group=Name der zu löschenden Gruppe
auth_user=Nutzername des löschenden Nutzers
auth_password=Passwort des löschenden Nutzer

Befehl:

delete_group:

Ändern einer Gruppe:

Hiermit können alle Daten einer Gruppe geändert werden. Das kann allerdings nur ein Administrator tun.

Parameter:

auth_user=anlegender Nutzer
auth_password=Passwort des anlegenden Nutzers
group=Name der Gruppe
base_group=Basisgruppe, von der die Rechte übernommen werden
can_create_groups=Kann Gruppen anlegen
can_create_users=Kann Nutzer Anlegen

can_delete_users=Kann Nutzer Löschen
can_view_all_users=Kann die Daten und Dateien aller Nutzer sehen
can_change_all_users=Kann die Daten und Dateien aller Nutzer ändern
can_view_all_permissions=Kann alle Berechtigungen sehen
can_edit_all_permissions=Kann alle Berechtigungen ändern
can_create_permissions=Kann Berechtigungen erstellen
can_edit_permissions=Kann Berechtigungen editieren
allowed_upload_bandwith=maximale Bandbreite zum Hochladen
allowed_download_bandwith=maximale Bandbreite zum Herunterladen
allowed_upload_number=maximale Anzahl der Uploads
allowed_download_number=maximale Anzahl der Downloads
allowed_number_upload_connections=maximale Verbindungen
allowed_number_download_connections=maximale Verbindungen
allowed_diskspace=maximaler Speicherplatz der genutzt werden kann
reset_time=Abrechnungszeitraum

Befehl:

edit_group

4.4.2. Dateiverwaltung

Hochladen einer Datei:

Mit diesem Befehl wird eine Datei hochgeladen. Dies ist der einzige Befehl, der nicht über einen GET-Request, sondern über einen Multipartrequest abgesendet werden muss. Es kann nur eine Datei hochgeladen werden. Nach dem Hochladen und der Verifikation des Nutzers, wird der Hash berechnet und überprüft. Soll eine veränderte Datei hochgeladen werden, so muss der Hash der ursprünglichen Datei angegeben werden. Die ursprüngliche Datei bleibt immer bestehen. Clients auf Basis des Parent-Flags eine Art Baumansicht aller Versionen einer Datei generieren.

Parameter:

auth_user=Nutzername des hochladenden Nutzers
auth_password=Passwort des hochladenden Nutzers
file=Datei
filename=Name der Berechtigung

parent=Basis (hash) der Datei (für das Verändern von Dateien wichtig)

Befehl:

upload

Herunterladen einer Datei:

Mit diesem Befehl kann eine Datei heruntergeladen werden. Es wird zunächst überprüft, ob der Nutzer die Datei gemäß seiner Berechtigung herunterladen darf.

Parameter:

auth_user=Nutzername des hochladenden Nutzers

auth_password=Passwort des hochladenden Nutzers

hash=Hash der Datei

Rückgabe:

die Datei im Content-Stream des HTTP-Response

Befehl:

download

Ändern der Berechtigung einer Datei:

Hiermit können Daten der Berechtigung einer Datei geändert werden. Dabei muss der Nutzer aber das Recht haben. Es können auch vergebene Berechtigungen editiert werden. Dafür muss allerdings auch die Berechtigung bestehen.

Parameter:

auth_user=Nutzername des hochladenden Nutzers

auth_password=Passwort des hochladenden Nutzers

hash=Hash der Datei (kann nicht geändert werden)

name=Neuer Name der Datei

permission_user=Nutzer dem die Berechtigung erstellt werden soll

permission_type=Typ der Berechtigung

permission_timeout=Ablaufdatum der Berechtigung (UTC)

permission_usecount=Anzahl der noch verbleibenden Benutzungen

Befehl:

edit_permission

Löschen einer Berechtigung:

Löscht eine Berechtigung, sofern der Typ der Berechtigung das erlaubt.

Parameter:

auth_user=Nutzername des hochladenden Nutzers

auth_password=Passwort des hochladenden Nutzers

hash=Hash der Datei

Befehl:

delete_permission

Herunterladen der Berechtigung einer Datei:

Löscht eine Berechtigung, sofern der Typ der Berechtigung das erlaubt.

Parameter:

auth_user=Nutzername des hochladenden Nutzers

auth_password=Passwort des hochladenden Nutzers

hash=Hash der Datei

Befehl:

get_permission

Erstellen einer Berechtigung einer Datei:

Erstellt eine Berechtigung, sofern der Typ der eigenen Berechtigung das erlaubt.

Parameter:

auth_user=Nutzername des hochladenden Nutzers

auth_password=Passwort des hochladenden Nutzers

hash=Hash der Datei

permission_user=Nutzer dem die Berechtigung erstellt werden soll

permission_type=Typ der Berechtigung

permission_timeout=Ablaufdatum der Berechtigung (UTC)

permission_usecount=Anzahl der noch verbleibenden Benutzungen

Befehl:

create_permission

Liste aller Berechtigungen:

Liefert eine Liste aller verfügbaren Berechtigungen, inklusive der nicht mehr Gültigen. Zusatzparameter ermöglichen durch reguläre Ausdrücke die Eingrenzung der Ergebnisse.

Parameter:

auth_user=Nutzername des hochladenden Nutzers

auth_password=Passwort des hochladenden Nutzers

Optionale Parameter:

Es sind bei allen Parameter reguläre Ausdrücke erlaubt. Rückgabe ist die Schnittmenge aller gefundenen Berechtigungen pro Einschränkung.

file_name=Liefert alle Berechtungen, die auf diesen Namen passen
file_parent=Liefert alle Berechtigungen mit diesem Parent
file_size=Liefert alle Berechtigungen mit dieser Größe (Alpha-Numerisch)
file_type=Liefert alle Berechtigungen mit diesem Typ (Stream/Datei)
file_usecount=Liefert alle Berechtigungen mit diesem Usecount
file_hash=Liefert alle Berechtigungen mit diesem Hash
file_permission_type=Liefert alle Berechtigungen mit diesem type
file_format=Liefert alle Berechtigungen mit diesem Format
file_timeout=Liefert alle Berechtigungen mit diesem Timeout
file_date=Liefert alle Berechtigungen mit diesem Erstellungsdatum

Rückgabe:

eine XML-basierte Liste aller Berechtigungen in der Form:

```
<files>
        <file>...</file>
        <file>...</file>
        <file>...</file>
        ...
</file>
```

Befehl:

get_permissions

4.4.3. Abrechnungsfunktionen

Liste aller vergebenen Berechtigungen:

Um die vergebenen Berechtigungen abrechnen zu können, kann hier eine Liste aller vergebenen Berechtigungen abgefragt werden, so kann auch eingesehen werden, wie oft ein Nutzer eine Berechtigung schon genutzt hat.

Parameter:

auth_user=Nutzername des hochladenden Nutzers
auth_password=Passwort des hochladenden Nutzers

Rückgabe:

eine XML-basierte Liste aller Berechtigungen in der Form:

```
<files>
        <file>...</file>
        <file>...</file>
        <file>...</file>
        ...
</file>
```

Befehl:

get_all_permissions

Alle Befehle und Funktionen liefern als Rückgabewerte einen Fehlercode und einen lokalisierten Fehlertext. Je nachdem wie die Rückgabe geschieht (XML, HTML, ...) steht der Rückgabecode in dem Feld „ret_code“ und der Rückgabetext im Feld „ret_text“. Die möglichen vordefinierten Fehlercodes sind im Folgenden beschrieben:

- RET_OK = "[OK]";
- RET_ERROR = "[ERROR]";
- RET_CREATED_USER = RET_OK + "USER CREATED";
- RET_USER_ALREADY_EXIST =
  RET_ERROR + "USER ALREADY EXISTS";
- RET_COULD_NOT_CREATE_USER =
  RET_ERROR + "COULD NOT CREATE USER";
- RET_CREATED_GROUP = RET_OK + "GROUP CREATED";
- RET_GROUP_ALREADY_EXIST =
  RET_ERROR + "GROUP ALREADY EXISTS";
- RET_COULD_NOT_CREATE_GROUP =
  RET_ERROR + "COULD NOT CREATE GROUP";
- RET_USER_AUTHENTICATED =
  RET_OK + "USER AUTHENTICATED";
- RET_COULD_NOT_AUTHENTICATE_USER =
  RET_ERROR + "COULD NOT AUTHENTICATE USER";
- RET_WRONG_PASSWORD =
  RET_ERROR + "WRONG PASSWORD";
- RET_NO_SUCH_USER = RET_ERROR + "NO SUCH_USER";
- RET_COULD_NOT_DELETE_USER =

RET_ERROR + "COULD NOT DELETE USER";

- RET_USER_DELETED = RET_OK + "USER DELETED";
- RET_COULD_NOT_CHANGE_USER_DATA =
RET_ERROR + "COULD NOT CHANGE USER DATA";
- RET_USER_DATA_CHANGED =
RET_OK + "USER DATA CHANGED";
- RET_NO_SUCH_GROUP = RET_ERROR + "NO SUCH GROUP";
- RET_COULD_NOT_DELETE_GROUP =
RET_ERROR + "COULD NOT DELETE GROUP";
- RET_GROUP_DELETED = RET_OK + "GROUP DELETED";
- RET_COULD_NOT_CHANGE_GROUP_DATA =
RET_ERROR + "COULD NOT CHANGE GROUP DATA";
- RET_GROUP_DATA_CHANGED =
RET_OK + "GROUP DATA CHANGED";
- RET_COULD_NOT_CREATE_FILE =
RET_ERROR + "COULD NOT CREATE FILE";
- RET_COULD_NOT_CREATE_PERMISSION =
RET_ERROR + "COULD NOT CREATE PERMISSION";
- RET_FILE_CREATED = RET_OK + "FILE CREATED";
- RET_PERMISSION_CREATED =
RET_OK + "PERMISSION CREATED";
- RET_FATAL_TRANSACTION_ERROR =
RET_ERROR + "FATAL TRANSACTION ERROR";
- RET_COULD_NOT_DELETE_FILE =
RET_ERROR + "COULD NOT DELETE FILE";
- RET_DELETED_FILE = RET_OK + "FILE DELETED";
- RET_NO_PERMISSION = RET_ERROR + "NO PERMISSION";
- RET_NO_SUCH_FILE = RET_ERROR + "NO SUCH FILE";
- RET_ROOT_PERMISSION_ALREADY_EXISTING =
RET_ERROR + "ROOT PERMISSION ALREADY
EXISTING";
- RET_HIGHER_PERMISSION_ALREADY_EXISTING =
RET_ERROR + "HIGHER PERMISSION ALREADY
EXISTING";

- RET_UPDATED_PERMISSION =
  RET_OK + "UPDATED PERMISSION";
- RET_DELETED_PERMISSION =
  RET_OK + "DELETED PERMISSION";
- RET_CAN_NOT_DELETE_ROOT_PERMISSION =
  RET_ERROR + "CAN NOT DELETE ROOT PERMISSION";
- RET_COULD_NOT_DELETE_PERMISSION =
  RET_ERROR + "COULD NOT DELETE PERMISSION";
- RET_DOWNLOAD_PERMISSION_GRANTED =
  RET_OK + "DOWNLOAD PERMISSION GRANTED";
- RET_COULD_NOT_PRODUCE_TICKET =
  RET_ERROR + "COULD NOT PRODUCE TICKET";

Alle Rückgabewerte richten sich dabei nach dem Schema <[ERFOLG]Meldung>, so dass bei einer dem System unbekannten Meldung erkannt werden kann, ob die Aktion erfolgreich oder mit Fehler ausgeführt wurde. Die Fehlermeldungen sind alle selbsterklärend und werden deshalb nicht weiter erläutert.

Prinzipiell kann und darf jede Fehlermeldung beim Aufruf eines jeden Befehls zurückgeliefert werden, soweit das Sinn macht. Zum Beispiel macht es keinen Sinn bei der Erstellung einer Gruppe den Fehlercode RET_DOWNLOAD_PERMISSION_GRANTED abzufragen, sondern eher RET_COULD_NOT_CREATE_GROUP und RET_CREATED_GROUP.

Die Liste der Befehle und Verwaltungsfunktionen erhebt keinerlei Anspruch auf Vollständigkeit, jedoch können mit diesen elementaren Befehlen alle Funktionen eines Dateisystems und eines Versionisierungssystems realisiert werden. Redaktionelle Funktionen können durch verzögertes Erstellen von Berechtigungen realisert werden. Wie das Kapitel 5 zeigen wird, hat die Referenzimplementation nicht alle Befehle implementiert, sondern nur die Grundfunktionen wie Upload, Download, Anlegen einer Berechtigung, Upstream, Downstream und Löschen einer Berechtigung, zur Demonstration der Funktionalität das Systems.

5. Implementation

5.1. Werkzeuge

Die Referenzimplementation des Dateisystems realisiert nicht immer die genaue Schnittstellenbeschreibung und implementiert auch nicht alle Befehle und Funktionen, da durch sie lediglich die Machbarkeit und Funktionalität des Systems demonstriert werden soll. Dabei ist die Implementation auf den vollen Umfang der Spezifikation erweiterbar. Wie in den Kapiteln 3 und 4 schon beschrieben, dient als Programmiersprache Java, einer von SUN Microsystems entwickelten objektorientierten Programmiersprache, die die Programme nicht in nativen Maschinencode übersetzt, sondern in maschinenunabhängigen Zwischencode (Bytecode). Dieser Zwischencode läuft dann in einer eigens für jedes Betriebssystem und jede Rechnerarchitektur entwickelten Java-Virtual-Machine, auch JVM genannt. Diese JVM verwaltet ihren Speicher selbst und verbirgt vor dem Java-Programm alle maschinenabhängigen Parameter und emuliert so einen autarken Rechner in der Laufzeit des Systems. Dieses Sandbox-Charakteristikum hat Vorteile, wie zum Beispiel die Plattformunabhängigkeit und die Sicherheitsgewährleistungen gegenüber dem Hostsystem der JVM.
Für die Webapplikation und die Datenhaltung kommt der JBOSS-Applicationserver zum Einsatz [JBOSSLN_2003]. In der Standard-Version ist der JBOSS-Applikationserver ein EJB-Container und eine ServletEngine, die über den Applikationserver angesprochen werden. In der Referenzimplementation wird der JBOSS 3.2 mit einem „built-in“ Apache Tomcat 4.1 verwendet, die Applikation sollte aber auch auf zukünftigen Versionen lauffähig sein. Konzept beim Einsatz des Systems ist es, mit Hilfe der CMP 2 (Container Managed Persistance) die Datenhaltung über EJB zu abstrahieren und die Logik über Servlets. Bei der CMP kümmert sich der EJB-Container um die Serialisierung und Deserialisierung der Daten, die in ein persistentes Enterprise Java Bean (Entity Bean) geschrieben werden. Die Erstellung eines Datenmodells für EJB CMP wird dadurch und durch UML-Modeling Tools stark vereinfacht.

Für die Präsentationsebene wird XML und XSL/XSLT benutzt, das heißt, dass die Daten aus den Entity Beans in der Servlet-Logik verarbeitet werden und dort einen strukturierten XML-Output generieren, der dann wiederum in einem Servlet mit Hilfe von XSL in ein gewünschtes Format transformiert wird. Dieses Transformation-Servlet heißt in dieser Implementation Cocoon 2 [APCOCOON_2003] und wird von der Apache Software Group als OpenSource Projekt entwickelt.

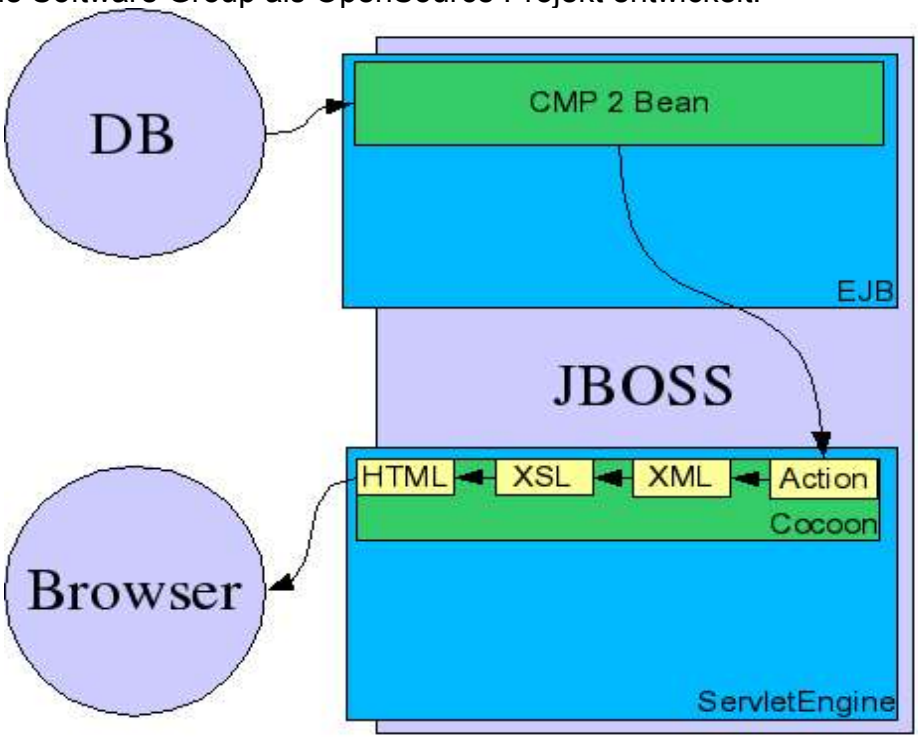

Abbildung 7 System Struktur der Referenzimplementation

Cocoon ist ein Servlet, dass als Grundfunktion die Transformation von XML über XSL in andere Formate übernimmt. Des weiteren bringt Cocoon noch einige nützliche Funktionen für die Erstellung und Verwaltung von Webapplikationen mit. Insbesondere sind das bestimmte Workflow- und Pageflowmechanismen, die sich hier speziell in Actions, Pipes, Generatoren usw. untergliedern. Im Kapitel 5.2. wird darauf noch näher eingegangen werden. Abbildung 7 zeigt den Datenfluss im System. Die Daten werden in einer Action vom CMP2-Entitiy-Bean angefordert, dieses holt sich die Daten aus einer der Action unbekannten Datenbank und übergibt sie der Action. Diese verarbeitet die Daten und erzeugt einen XML-Output. Nun transformiert das Cocoon-Servlet in der spezifierten Pipe den XML-Output der Action über ein in der Pipe spezifiertes Stylesheet in zum Beispiel HTML. Danach kann der Applikationserver den HTTP-Response schicken. Diese Lösung ist eine saubere und klar strukturierte MVC (Model-View-Control) Lösung und erlaubt, das System

modular und erweiterbar aufzubauen.

Für die Implementation des Systems wurde eine Testversion des Borland JBuilder 8 [JBUILDERLN_2003] verwendet, da dieser über ein komfortables EJB-Modeling-Tool verfügt. Für die Weiterentwicklung wurden jedoch nur noch OpenSource IDEs und Projektmanagement-Werkzeuge verwendet. Als IDE kommt vorrangig IBM's Eclipse [ECLIPSELN_2003] zum Einsatz. Als Buildtool und Projektmanagementwerkzeug kommt MAVEN [APMAVEN_2003] als Erweiterung von Apache's ANT [APANT_2003] zum Einsatz. Mit Hilfe von MAVEN oder auch Ant ist es möglich, die komplizierten Deploy-Strukturen, die ein EJB-Container und ein Webapplikation-Container wie Tomcat erwartet, komfortabel zu erzeugen. Ein Beispiel für ein Build-Script wird in Kapitel 5.2. näher erklärt.

Damit das Ziel der OpenSource-Entwicklung verwirklicht werden kann, wird die Referenzimplementation bei Sourceforge [SOURCEFORGELN_2003] angesiedelt, einer Webplattform für die teamorientierte Entwicklung von OpenSource Projekten. Für die Teamunterstützung wird dort das Versionisierungssystem CVS [CVS_2003] eingesetzt. Mit CVS können mehrere Entwickler zur gleichen Zeit an einem Projekt arbeiten und ihre Quellen versionieren und sichern. Einzelheiten über die Arbeit und den Umgang mit CVS werden hier nicht näher beschrieben.

Da die Webapplikation auf dem Client keine Multimedialen Daten wiedergeben, geschweige denn aufzeichnen kann, wurde in Kapitel 4.1. und 4.2. auf den Einsatz von Java-Applets und dem Java Media Framework hingewiesen. Da sich in der ersten Version aber Java-Applets als nicht zuverlässlich und als instabil erwiesen haben, wurde die Streaming-unterstützende Client-Applikation über Java Web Start realisiert. Dabei ist Java Webstart (JWS) [JWSLN_2003] eine Java-Anwendung die das Java Network Launch Protocol (JNLP) unterstützt, einem XML-basierten Protokoll, dass ähnlich wie das Applet-Tag bei HTML Java-Anwendungen vom Server lädt und auf dem Clientrechner ausführt. Im Unterschied zu Applets werden die Anwendungen außerhalb des Browsers in einer eigenen Sandbox ausgeführt, so dass die Unterschiede der Java-Unterstützung bei den verschiedenen Browserherstellern verborgen werden. Java Webstart bietet dabei auch eine bessere Unterstützung des Cachings der schon heruntergeladenen Anwendungen. Bei Applets müssen die Klassen der Anwendung ständig neu geladen werden, wogegen Webstart die Anwendungen speichert und bei jedem Start die Aktualität der Klassen überprüft und gegebenenfalls neu herunterlädt. Ein weiterer Vorteil des Einsatzes

des JNLP ist, dass es sich durch seine XML-Basiertheit gut in das Gesamtkonzept des MVC eingliedert.
Die Implementation sieht vor, für jeden Befehl eine Cocoon-Action und Pipe anzulegen, so dass Befehlserweiterungen leicht implementiert werden können.

5.2. Methoden

In diesem Abschnitt werden auszugsweise bestimmte Implementationstechniken und Methoden erläutert, damit der Einstieg in das Gesamtkonzept leichter fällt. Im Anhang befindet sich der komplette Projektbaum mit folgenden Verzeichnissen:

maci	-->src	-->conf	-->	Konfigurationsdateien
		-->java	-->	Java Quellen
		-->webapp	-->	Webapplikation
	-->target	-->	normaler Weise erst nach build vorhanden	
		-->	maci_ejb.jar	
		-->	maci_war	
	-->maven.xml			
	-->project.xml			
	-->project.properties			

Die Datei maven.xml enthält dabei alle Ziele für das Kompilieren und Erstellen (build) des Projektes. Damit das Projekt kompiliert werden kann, muss das Tool MAVEN [APMAVEN_2003]installiert werden. Danach kann mit dem Aufruf von

<Project-Root>$maven all

das Projekt komplett erzeugt werden. Ein MAVEN-Buildfile ist eine XML-Datei mit einer Menge von definierten GOALS (Zielen):

```
<project default="all" >
  <goal name="myejb" >   -->Buildziel
    <jar destfile="${maven.build.dir}/maci_ejb.jar" >
      <fileset includes="META-INF/**" dir="${maven.build.dir}/classes" />
      <fileset includes="org/erminea/maci/ejb/**" dir="${maven.build.dir}/classes" />
      <fileset includes="org/erminea/maci/*.class" dir="${maven.build.dir}/classes" />
    </jar>
```

```
  </goal>
  <goal name="applet" >
    <jar destfile="${maven.build.dir}/maci_applets.jar" >
...
</project>
```

Alle diese Goals definieren einen möglichen Aufrufparameter des maven-Binaries, hier zum Beispiel erzeugt der Aufruf „maven myejb“ das EJB-Jar für den EJB-Container. In der Datei project.xml sind dagegen alle Projekt bezogenen Informationen wie Lizenz, benötige Bibliotheken, Autor usw. zu finden. Die Bibliotheken werden dabei von einer in der project.xml definierten Stelle heruntergeladen. In dieser Implementation werden die Bibliotheken von http://www.erminea.org/maci heruntergeladen. Das hat den Vorteil, dass die zum Teil großen Bibliotheken immer auf Anforderung heruntergeladen und nach Bedarf aktualisiert werden können. Ein Nachteil ist, dass das Projekt bei fehlendem Zugang zum entsprechenden Download-Ort nicht übersetzt werden kann.

Nach einem Aufruf von „maven all“ wird im Verzeichnis „target“ eine Datei maci_ejb.jar erzeugt. Dieses Jar-File enthält alle EJBs für den EJB-Container und wird auf dem JBOSS-Server deployed, indem es in das Verzeichnis (standardmäßig) <JBOSS>/server/default/deploy kopiert wird. Je nach JBOSS-Version kann das aber anders sein. Die EJBs unterteilen sich ein drei wesentliche Funktionsgruppen. Zum Einen die Entity-Beans, die komplett vom JBuilder-EJB-Modeler mit UML generiert worden sind. Diese Entity-Beans kapseln die Daten der Datenbank gegenüber den Wrapper-Beans. Die Wrapper-Beans sind dabei Beans, die die Remote-Funktionalität der Entitiy-Beans kapseln. Da die Daten der Entity-Beans verteilt sein können, verfügen Entitiy-Beans standardmäßig nicht über eine Remote-Schnittstelle mit Key-Funktion (Tabellenreferenzen oder auch Foreign Keys genannt), können also außerhalb ihres Containers keine Referenzen ansprechen.

Dafür wurden diese Wrapper-Beans geschrieben, die sich die Referenzen aus allen bekannten Containern zusammensuchen und die Schnittstelle nach außen bilden. Zum Beispiel kapselt das Bean org.erminea.maci.ejb.PermissionWrapper die Funktionalität des Wrapper-Beans org.erminea.maci.ejb.FilePermission, welches wiederum die Daten des Entity-Beans org.erminea.maci.ejb.AbstractFile kapselt. Die Zusammenhänge sind in den UML-Diagrammen von Abbildung 9, Abbildung 8 und Abbildung 10 erkennbar.

Eine weitere EJB-Gruppe sind die Manager-Beans, die bestimmte Konsistenzlogiken implementieren, zum Beispiel das Anlegen eines Nutzers oder das Überprüfen der Berechtigung eines Downloads.

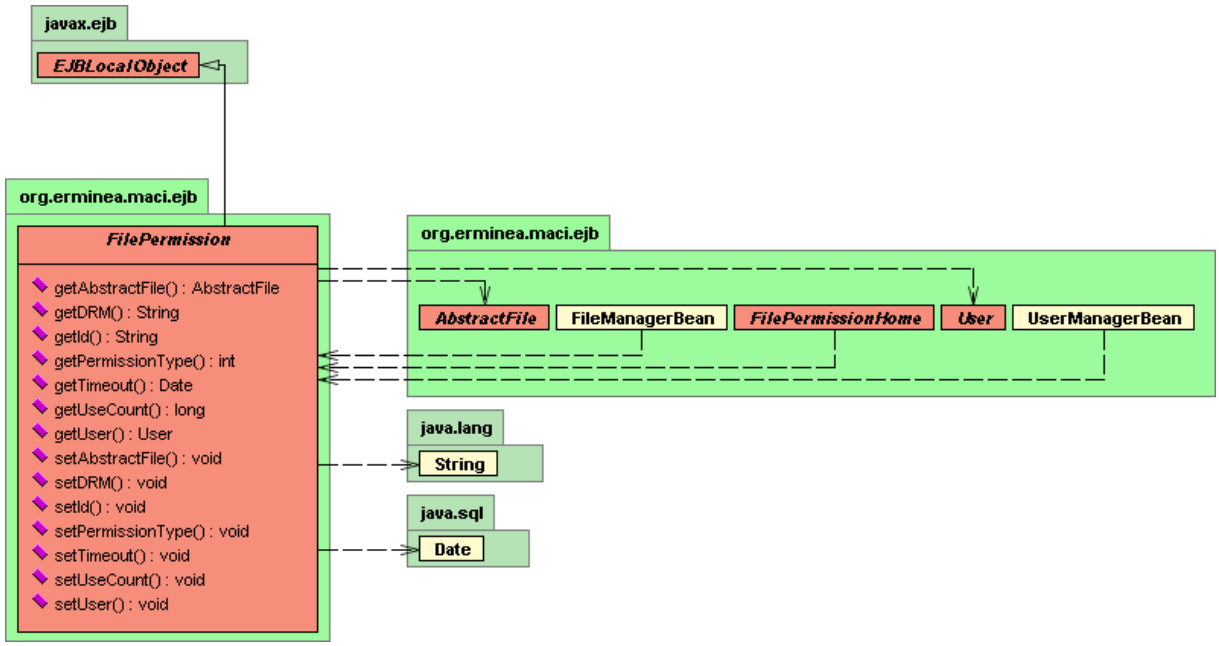

Abbildung 8 UML Diagramm für das FilePermission Wrapper Bean

Diese Manager-Beans werden dann in den Actions über RMI (Remote Method Invocation) instanziert und rufen dann über die Wrapper-Beans bestimmte Daten ab. Die Actions sind Klassen, die von Cocoon verwaltet werden und Logik kapseln. Innerhalb der Actions kann jeder beliebige Java-Code aufgerufen werden. Die einzige Möglichkeit der Actions Daten nach außen zu übertragen, sind Parameter und Rückgabewerte. Eine Action ist im System die zentrale Datenverarbeitung-Einheit, die HTTP-Requests entgegen nimmt, auswertet und gewisse Aktionen durchführt. Eine Action sollte keinen XML-Output erzeugen, sondern liefert Daten an eine so genannte XSP (eXtensible Server Page), die diese Daten dann in strukturiertes XML umwandelt.

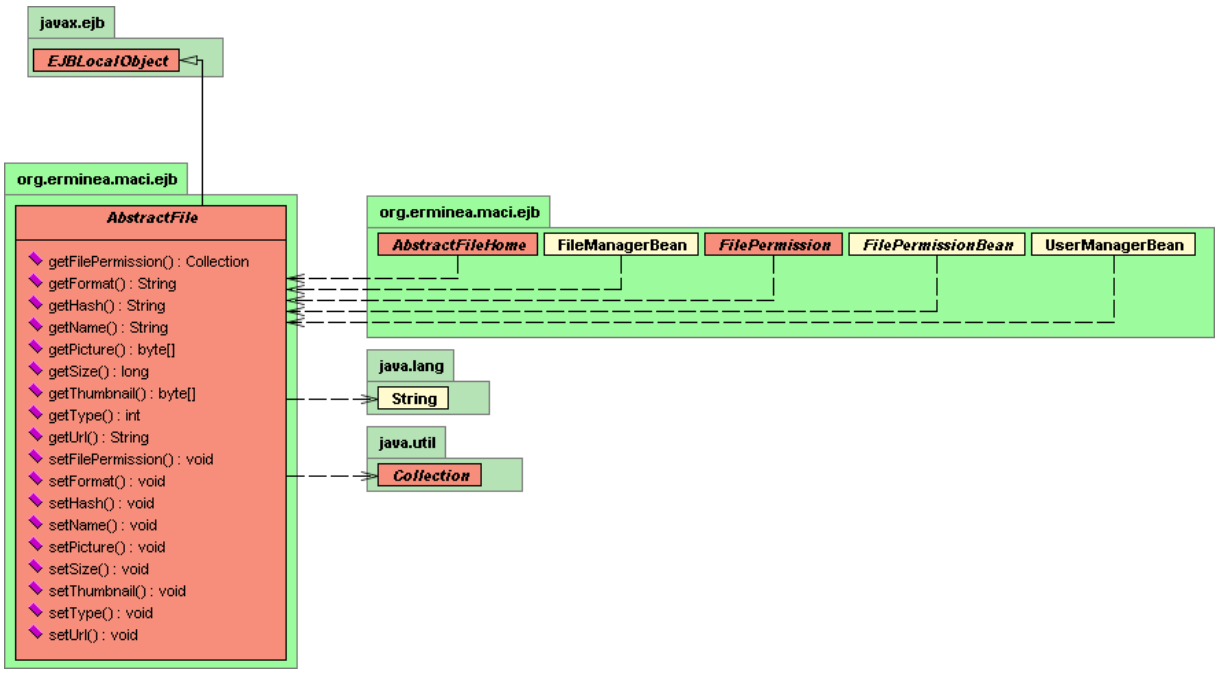

Abbildung 9 UML Diagramm für das AbstractFile Entity Bean

Um diese komplizierten Abläufe zu strukturieren, bedient sich Cocoon eines Sitemap-Konzeptes, mit dem so genannte Pipes definiert werden können. In den Pipelines können dann Matches definiert werden:

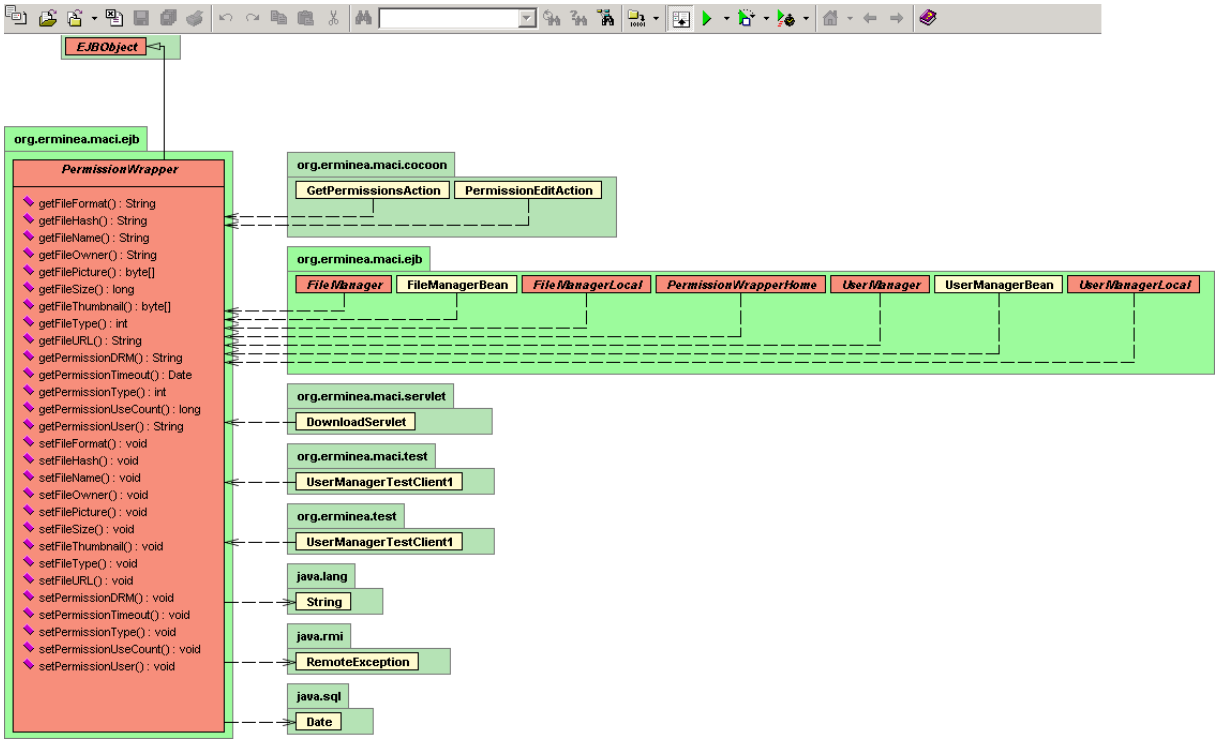

Abbildung 10 UML Diagramm für den PermissionWrapper

<map:pipelines>

<map:pipeline>

<map:match pattern="*/pmanager"> -->matched auf jede URL mit dem Ende

„pmanager“

<map:act type="login"> -->ruft zunächst die Login-Action auf, um zu überprüfen, ob der Nutzer angemeldet ist

<map:parameter name="page" value="pmanager"/> --> die Login Action bekommt die ursprüngliche Seite übergeben, um den Kontext wieder herstellen zu können

<map:generate type="serverpages" src="{page}.xsp"/> --> nach Abhandlung der Action wird eine XSP-Seite mit dem von der Login-Seite übergebenen Namen als XML-Output generiert. Wenn der Nutzer schon angemeldet war, wird das die Seite pmanager.xsp sein, in der dann die Struktur der Seite festgelegt ist.

<map:transform src="{../1}.xsl"/> --> transformiert die Seite mit dem in der URL übergebenen Stylesheet. Zum Beispiel würde die URL http://..../maci/erminea/pmanager die pmanager.xsp mit dem erminea.xsl Stylesheet transformieren.

<map:serialize/> -->Serialisiert den HTML-Output, zum Beispiel wandelt **ä nach ä**

</map:act>

</map:match>

...

Mit diesen Matches wurden alle Methoden und Verwaltungsfunktionen realisiert. Die jeweils entsprechenden XSP-Seiten folgen dabei einer festgelegten Seitenstruktur, die dann mit Hilfe des in der URL spezifierten Stylesheets transformiert werden. Die Struktur der Seiten sieht dabei wie folgt aus. Die DTD-Darstellung wird bewusst vermieden, da sie keine Strukturmerkmale aufzeigen kann:

```
<?xml version="1.0"?>
<xsp:page language="java"
    xmlns:xsp="http://apache.org/xsp"
    xmlns:esql="http://apache.org/cocoon/SQL/v2"
    xmlns:xsp-request="http://apache.org/xsp/request/2.0"
>
<page>
```

```
  <title>MaCI Project</title>
  <menubar>
    <menu>
      <menuitem>
        <link>
          <name>Startseite</name>
          <ref>pmanager</ref>
        </link>
      </menuitem>
  <container name="maci Manager">
     <component>
       <label>Willkommen im MaciManager</label>
       <label>
Hier koennen Sie Dateien und Livestreams hochladen, Berechtigungen
anlegen,bearbeiten und loeschen, sowie
diese herunterladen.
       </label>
     </component>
 </container>
</page>
</xsp:page>
```

Aus dieser Beispielseite wird dann mit Hilfe des im Folgenden spezifierten Stylesheets eine HTML-Seite erzeugt:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html" indent="yes"/>
  <xsl:output encoding="UTF-8"/>
  <xsl:strip-space elements="*"/>
  <xsl:template match="page">
    <html>
      <head>
        <title><xsl:value-of select="title"/></title>
```

```
      <link rel="stylesheet" type="text/css" href="template.css" title="ka"/>
    </head>
    <body background="#ffffff">
      <img align="left" src="pic/macilogo.gif" width="140" height="30"/>
      <br/><br/>
       <xsl:apply-templates select="menubar"/><br/>
       <xsl:apply-templates select="container"/>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Dabei entspricht jedes Template einem Tag in der XML-Datei. In diesem Beispiel wird nur das <page>-Tag gematched, alle anderen nicht spezifierten Tags, werden dann nicht weiter berücksichtigt. Prinzipiell kann mit dieser Technik jedes beliebige Format erzeugt werden. Da das Stylesheet in der URL ausgewählt wird, lassen sich schnell neue Stylesheets entwickeln und sofort über die Angabe in der URL quasi als „Skin“ laden. Die DTD der XML-Seiten ist dabei nicht generisch sondern auf die Applikation angepasst, damit die Skin- oder Stylesheetentwicklung einfach zur realisieren ist. Zur Spezifikation des Dateisystems gehört die DTD nicht, wird deshalb auch in der Referenzimplementation nicht weiter festgelegt.
Eine vordefinierte DTD existiert jedoch für das Java Network Launch Protocol (JNLP). Hier soll nur kurz die Struktur und die Funktionsweise der Clientanwendung für Streaming erklärt werden. Die Client-JNLP-Dateien werden von XSP-Seiten generiert und erhalten von diesen alle benötigten Daten, um zum Beispiel einen Stream abspielen zu können:

```
<?xml version="1.0"?>
<xsp:page language="java"
    xmlns:xsp="http://apache.org/xsp"
    xmlns:esql="http://apache.org/cocoon/SQL/v2"
    xmlns:xsp-request="http://apache.org/xsp/request/2.0"
>
```

<jnlp spec="1.0+" codebase="http://www.erminea.net:8080/maci/jnlp"> --> Informationen über die Herkunft der Anwendung

<information>

<title>MaCI Media Player</title>

<vendor>Erminea Open Software Group</vendor>

<homepage href="http://www.erminea.org" />

<description kind="one-line">basic Media Player for MaCI Streaming Environment</description>

<description kind="tooltip">Media Player</description>

<offline-allowed />

</information>

<security>

<all-permissions /> --> um das zu ermöglichen müssen alle JAR-Files signiert sein

</security>

<resources>

<j2se version="1.4 1.3 1.2" /> -->dieses verlangt bestimmte Versionen der JVM

<jar href="http://localhost:8080/maci/lib/fplayer.jar" />--> in diesem JAR-File steckt die Hauptapplikation des Media Players

</resources>

<application-desc main-class="org.erminea.maci.player.FPlayer"> --> die Hauptklasse des Players

<xsp:logic> -->an dieser Stelle wird XSP-Logik eingefügt, welche bestimmte benötigte Parameter aus dem Request extrahiert

```
String user="",pw="",url="",server="",port="",ftype="",hash="";
try {
  user=(String)request.getSession().getAttribute("user");
  pw=(String)request.getSession().getAttribute("password");
  java.util.Properties props=(java.util.Properties)request.getSession().
getAttribute("properties");
  url=(String)props.getProperty("download_url");
  server=(String)props.getProperty("corona_server");
  port=(String)props.getProperty("corona_port");
```

}catch(Exception e){}

ftype=(String)request.getParameter("type");

hash=(String)request.getParameter("hash");

...

</xsp:logic>

<argument><xsp:expr>url</xsp:expr></argument> -->hier werden der JNLP-Anwendung Parameter übergeben, die es zum Abspielen eines Streams benötigt

<argument><xsp:expr>user</xsp:expr></argument>

<argument><xsp:expr>pw</xsp:expr></argument>

<argument><xsp:expr>ftype</xsp:expr></argument>

<argument><xsp:expr>port</xsp:expr></argument>

<argument><xsp:expr>server</xsp:expr></argument>

<argument><xsp:expr>hash</xsp:expr></argument>

</application-desc>

</jnlp>

</xsp:page>

Die Codesnippets sind alle stark gekürzt, können aber alle im Verzeichnis maci->src-webapp im Anhang gefunden werden.

Der Streamingserver ist in der Referenzimplementation im Cocoon integriert, bzw. wird mit dem Cocoon-Server gestartet und beendet. Der Server kann im Package org.erminea.corona gefunden werden. Der Streamingserver implementiert, wie in der Spezifikation verlangt, ein einfaches Relay von Objekten. Um den Streamingserver anzusprechen, wird die Benutzung der Klassen org.erminea.corona.UploadClient und org.erminea.corona.DownloadClient empfohlen. Diese Klassen kapseln die in der Spezifikationen beschriebenen Requestparameter. Ein Upload eines Streams geschieht dann auf folgende Weise:

```
client=new UploadClient(ObjectInputStream in, ObjectOutputStream out,
                boolean save, String filename, String format, String user,
                String pw, String channel);
client.sendRequest();
String return_code=client.receiveResponse();
```

Wobei nun die Daten über den ObjectOutputStream gesendet werden können. Sollen Mediadaten versendet werden, empfiehlt sich die Benutzung der Klasse org.erminea.media.RelayDataSink, mit der eine Datenquelle in einen ObjectOutputStream umgeleitet wird. Zum Empfangen eines solches Streams sollte der org.erminea.corona.DownloadClient verwendet werden:

client=DownloadClient(ObjectInputStream in, ObjectOutputStream out,
String hash, String user, String pw);

Um einen Mediastream dann wiedergeben zu können, sollte die Klasse org.erminea.media.ObjectDataSource verwendet werden, die einen ObjectInputStream in eine JMF-konforme DataSource umleitet, welche dann zum Beispiel mit

javax.media.protocol.Player player
=javax.media.Manager.createRealizedPlayer(ObjectDataSource ds);

wiedergegeben werden kann.

6. Test und Erprobung

6.1. Performance

Nach der Referenzimplementation muss nun noch sichergestellt werden, das sowohl Server und Client bestimmten Ansprüchen an Leistung und Performance genügen. Zu diesem Zweck wurde das System auf einem Pentium 3 Rechner mit 1GHz Taktfrequenz, 256 MB DDR-Ram und RedHat Linux 8 installiert und ein Dauertest durchgeführt. Ziel des Tests war vordergründig die Stabilität und Verfügbarkeit des Systems zu testen. Die Testergebnisse eignen sich nicht zur Evaluation bezüglich einer geplanten Optimierung des Systems, sie sollen lediglich die Machbarkeit des Systems demonstrieren.
Um an diese Lastdaten zu kommen wurden Lastwerte eines einzelnen Nutzers, in

der Annahme, dass sich der Lastanstieg linear verhält, einfach interpoliert. Die Abweichungen können dabei +/- 5% betragen ist. Für den Lasttest mit einem Nutzer wurden jeweils 10 Versuche durchgeführt und der nur jeweils höchste Lastwert als worst-case-Wert zur Interpolation benutzt.

Nutzer	*Upload*	*Download*	*Upstream*	*Downstream*
1	0,1%	0,3%	0,1%	0,3%
10	1%	3%	1%	3%
100	100%	300%	100%	300%
500	500%	1500%	500%	1500%

Tabelle 7 Interpolierte Worst-Case CPU-Lastwerte für den Server

In Tabelle 7 sind diese interpolierten Werte für maximal 500 verbundene Nutzer dargestellt. Die Werte über 100% sind dabei nicht als reelle Lastwerte sondern als Maß der Reaktionsfähigkeit des Servers zu interpretieren, es soll demonstrieren, wie bei linearem Anstieg der Last der Server skaliert werden muss, um Anfragen in (nahe) Echtzeit zu bearbeiten. Der Testrechner wäre also nicht in der Lage mehr als 30 Benutzer in Echtzeit zu bedienen. Es ist allerdings nicht zu erwarten, dass die Last im Grenzfall linear anwächst und 100 % übersteigt, dennoch sollte mit dieser Annahme das System bei mehr als einhundert angemeldeten Nutzern (also Nutzer, die das System potentiell zum gleichen Zeitpunkt verwenden könnten) auf mehrere Rechner verteilt werden. Bei Messungen hat sich gezeigt, dass der JBOSS-Applikationserver ein effizientes Speicher und Lastmanagement hat, so dass die Auslastung zwischen 0,1% und maximal 70% schwankte. Kurzzeitig wurde diese maximale Auslastung von 70% auch bei Null verbundenen Benutzern beobachtet. Dieser Effekt ist durch den periodischen Aufruf des Java-Garbage Collectors zu erklären, der periodisch alle Objekte auf ihre Referenzen überprüft und gegebenenfalls löscht.

Bei einem ausführlichen Lasttest während der CeBIT 2003 wurden Livestreams (Abbildung 11) über das CeBIT-Netz nach Leipzig (HTWK) und von dort wieder zum CeBIT-Stand zurück übertragen. Das Netz war am Stand mit einer 10 Mbit Leitung ausgelegt. Durch das starke Verkehrsaufkommen im gesamten CeBit-Netz kam es zu zeitweisen Engpässen beim Datentransfer ins öffentliche Internet, so dass die Übertragung bei 64 kbit/s schon nur noch ruckelnde Videoströme (also starke

Jittereffekte) lieferte. Dabei wurden Datenraten zwischen 500 kbit/s und 2 Mbit/s über einen MJPEG und DIVX Codec übertragen. Zeitweise waren bis zu 5 Teilnehmer mit dem Server verbunden und riefen den bis zu 2 Mbit/s großen Videostream ab.

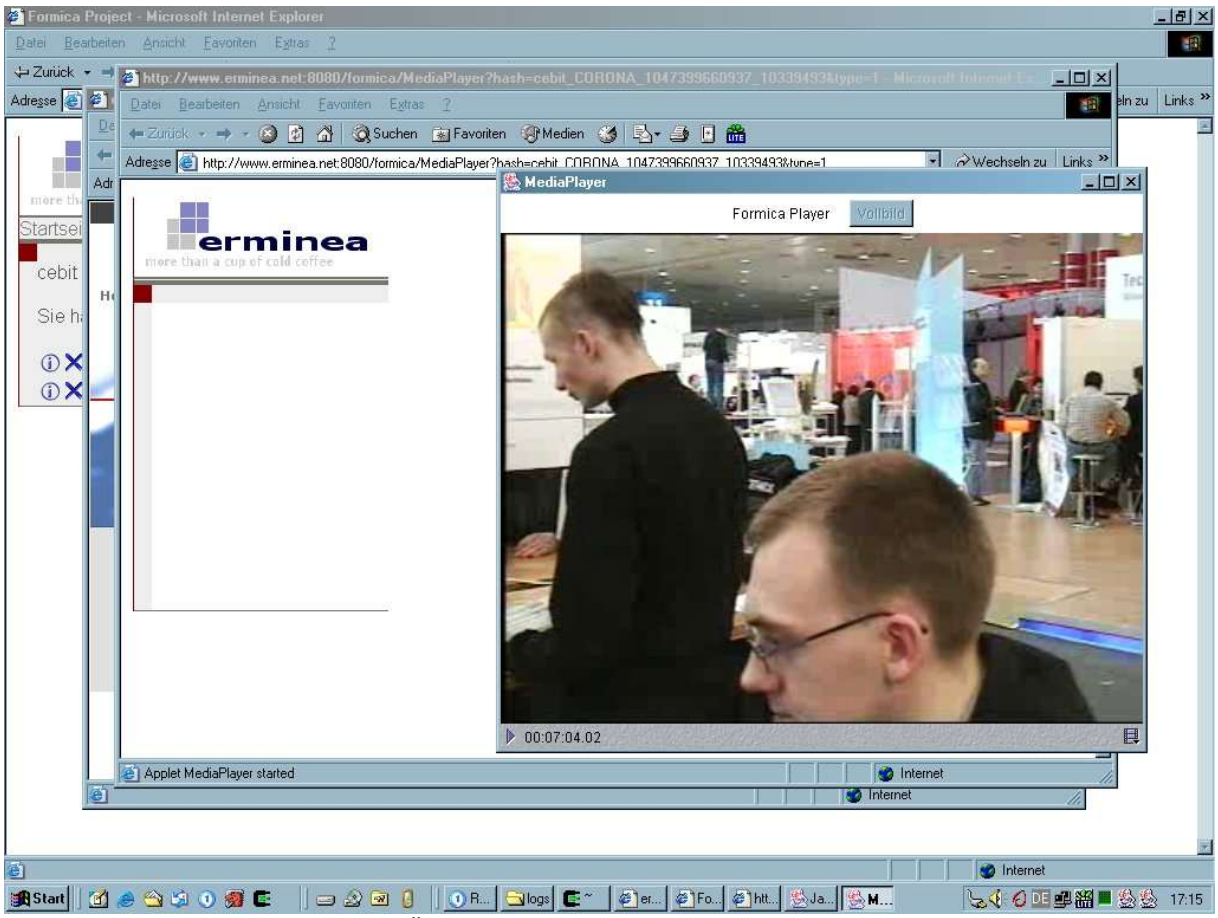

Abbildung 11 Screenshot von einer Übertragung während der CeBIT 2003

Der Server arbeite dabei stabil und war bis maximal 5% ausgelastet. Einzig das Netzwerk verlangt für solche Streams eine hohe Bandbreite (im konkreten Testfall mindestens 20 Mbit/s), da sich so Jitter-Effekte teilweise eliminieren lassen und die Videos flüssig (15 Bilder/s) übertragen werden.

Abbildung 12 Upstream-Client der Referenzimplementation

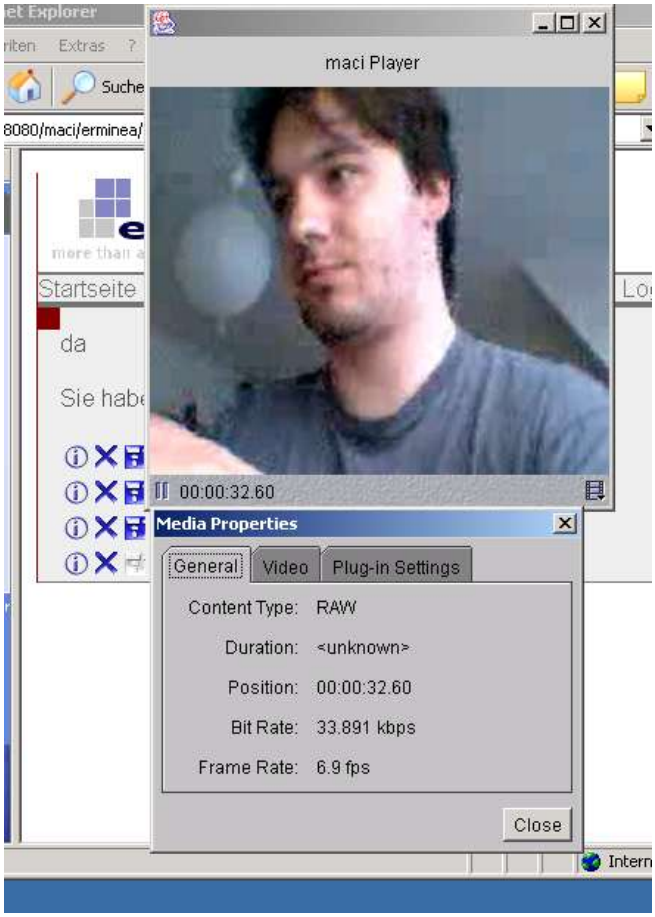

Abbildung 13 Downstream von einer Webcam im H.263 Format und einer Auflösung von 176x144

Auf der Clientseite lassen sich Lastparameter wesentlich besser testen und

auswerten.

Codec	*Auflösung*	*Framerate*	*Systemlast*	*Java-Last*	*Gemessene Bandbreite*	*Medienbandbreite*
H.263	352x288	15	20%	40%	ca. 140 kbit/s	ca. 130 kbit/s
H.263	176x144	15	20%	30%	ca. 80 kbit/s	ca. 60kbit/s
H.263	128x96	15	20%	30%	ca. 56 kbit/s	ca. 30kbit/s
MJPEG	176x144	15	20%	15%	> 150 kbit/s	> 150 kbit/s
MJPEG	128x96	15	20%	15%	130 kbit/s	130 kbit/s

Tabelle 8 Systemlast bei der Videoübertragung über A-DSL (768/128 kbit/s)

Bei der Messung wurde Microsoft Windows XP mit DSL Anbindung genutzt (128 kbit/s Uploadbandbreite). Der Rechner verfügt über einen 1.7 Ghz Intel Pentium 4 Prozessor und 256 MB PC-133 RAM. Die Messwerte haben eine maximale Abweichung von +/- 10% die durch Rundung und zeitliche Schwankungen bei der Prozessor- und Netzwerkauslastung entstand.

Die 20% Systemauslastung die in Tabelle 8 zu beobachten ist, resultiert aus dem WDM-Capture Treiber, der die Aufnahme über die genutzte Webcam regelt. Da ein DSL-Anschluss genutzt wurde, sind Datenraten über 130 kbit/s allerdings nicht mehr eindeutig zu messen, diese spielen aber auch keine Rolle, da die Messergebnisse Aufschluss über die zu erwartenden Rechenlasten bei Benutzung des Systems geben sollen. Interessant ist die hohe Java-Auslastung von über 30% bei den H.263-Codecs. Diese kommt zu Stande, da das Java Media Framework einen Software Codec für die Komprimierung benutzt. Der MJPEG-Codec dagegen benutzt native JPEG-Kompression. In früheren JMF-Versionen lag die Auslastung bei der H.263-Komprimierung allerdings wesentlich niedriger (System- und Java-Last weniger 50% auf einem PIII 700MHz siehe [BEHA_2001]).

Die Qualität bei der Videoübertragung ist gut (15 Bilder/s und kaum sichtbare Makroblöcke) und wegen der fehlenden Zwischenpufferung (wie sie zum Beispiel bei Real oder Microsoft Mediaplayer verwendet wird) nahezu verzögerungsfrei (weniger als 100ms). Aber aufgrund der fehlenden Implementation eines Jitter-Ausgleichs, ist die Qualität von Audio bei stark belastetem Netz eher schlecht. Die schlechte Qualität is dabei deutlich hörbar durch Knackgeräusche die durch das zu

gleichzeitige Abspielen zweier Audiopakete, die durch einen Jittereffekt nicht mit vorgesehener Verzögerung beim Empfänger ankommen, entstehen.
Durch die Kapselung der Medienpakete in Objekte entsteht gerade bei den kleinen Audiopaketen von wenigen Bytes ein beträchtlicher Overhead, so dass anstelle der bei GSM ursprünglichen 13 kbit/s nun ca. 30 kbit/s übertragen werden.

6.2. Usability

Zur Erprobung der Usability wird hier die Funktionsweise der wesentlichen Funktionen des Systems unter Zuhilfenahme von einigen Screenshots beschrieben. Dabei sollen einige wichtige Abläufe erläutert und dargestellt werden.
Wenn das System aufgerufen wird, im konkreten Beispiel ist das die Testinstallation auf http://www.erminea.net:8080/maci/erminea/pmanager, so wird das Stylesheet „erminea.xsl“ benutzt, welches auch zur Präsentation auf der CeBIT2003 zum Einsatz kam. Die erste Seite, die zu sehen ist, ist die Login-Seite (Abbildung 14), auf der zunächst ein neuer BenutzerAccount angelegt werden muss, sofern das noch nicht geschehen ist.

Abbildung 14 Login-Seite der Referenzimplementation

Das Anlegen eines neuen NutzerAccounts geschieht über den Link „Neuer Nutzer“. Dort muss ein Nutzerkürzel, Passwort und Name angegeben werden. Ein Screenshot

kann in der Anlage im Verzeichnis „screenshots“ gefunden werden. Nach dem Anlegen des NutzerAccounts kann der Nutzer mit diesem Nutzerkürzel und Passwort angemeldet werden. Das Fehlermanagement des Systems ist so ausgelegt, dass der Nutzer bei jeglichen Fehlern (Fehleingaben, Browserfehlern, Netzwerkfehlern) immer, egal auf welcher Seite er sich gerade befindet, zur Login-Seite mit einer entsprechenden Fehlermeldung zurück gelangt. Auf der Hauptseite des Systems kann der Nutzer nun navigieren und alle Funktionen des Systems benutzen.

Um eine Datei hochzuladen, wird der Upload-Link („Datei hochladen“) benutzt. Der Nutzer bekommt nun einen Dialog zu sehen, bei dem er eine Datei auswählen und hochladen kann. Wenn dann auf den Button Upload geklickt wird, wird die Datei hochgeladen. Eine Statusanzeige fehlt allerdings bei den meisten Browsern. Viele Browser sind nicht in der Lage große Dateien (>50 MB) hochzuladen. Nach einem erfolgreichen Upload erscheint eine entsprechende Erfolgsmeldung im gleichen Fenster. Danach können mit einem Klick auf „Dateien“ im Hauptfenster alle verfügbaren Berechtigungen (Abbildung 15) eingesehen werden.

Abbildung 15 Dateiliste der Referenzimplementation

In dieser Berechtigungsliste kann der Nutzer nun Dateien herunterladen, im mitgelieferten Webstart Player abspielen und Berechtigungen erstellen und löschen. Der Download einer Datei wird mit einem Klick auf das Disketten-Icon ausgelöst.

Bei einem Klick auf das Play-Icon, wird initial eine Java Webstart Anwendung heruntergeladen (ca. 80 kbyte), die dann versucht die Datei oder den Stream wiederzugeben.

Bei einem Klick auf das Häkchen-Icon, öffnet sich der Dialog zum Bearbeiten und Anlegen einer Berechtigung. Um eine neue Berechtigung anzulegen, muss der

Nutzer das Nutzerkürzel des zu Berechtigenden kennen. In der Referenzimplementation können der Berechtigungstyp, der Ablauf und die Anzahl der Benutzungen angegeben werden. Wird das Ablauffeld leer gelassen, so gilt die Berechtigung unbeschränkt. Bei der Anzahl der Benutzungen muss jedoch eine „-1“ eingetragen werden, wenn die Anzahl nicht beschränkt werden soll.
Wenn die Java Webstart Player und Streamingclients genutzt werden sollen, so muss vor der Benutzung noch das Java Runtime Environment (JRE) [JRELN_01] und das Java Media Framework installiert werden (JMF) [JMFLN_01]. Danach kann mit einem Klick auf „Stream hochladen“ im Hauptfenster ein Stream hochgeladen werden (Abbildung 12). In der Referenzimplementation, beschränkt sich die Auswahl auf Audio und Video von einem Aufnahmegerät (CaptureDevice).
Der Eintrag bei Kanalname bestimmt dabei den Namen, der später in der Dateiliste sichtbar ist. Nachdem das Aufnahmeformat und Übertragungsformat festgelegt wurde, kann der Stream gestartet und danach unter „Dateien“ wieder abgespielt werden (Abbildung 13).
Ein Nachteil beim Umgang mit Streams ist die Trennung von Audio und Video, so ist es momentan nicht möglich Audio und Video zu einem Stream zu kombinieren. Das Problem und wurde hier nicht gelöst, da nur die Machbarkeit demonstriert werden soll.

7. Ausblick und Zusammenfassung

In der vorliegenden Arbeit ist ein System entwickelt worden, mit dem man beliebige Daten im Internet verwalten, abrufen und verbreiten kann. Durch die Flexibilität, die erreicht wurde, ist es nun möglich mit Hilfe des Dateisystems sowohl multimediale Daten in Echtzeit zu übertragen als auch kooperativ mit mehreren Nutzern an Dokumenten zu arbeiten. Es wurde eine erweiterbare, plattformunabhängige Protokollschnittstelle durch XML und ein erweiterbares, plattformunabhängiges API (Application Programming Interface) durch Java geschaffen, mit dem es möglich ist, aufbauend auf dem Dateisystem, beliebige Clients zu entwickeln, die dann unter anderem Videokonferenzprogramme, Videodatenbanken oder Dateimanager sein können.
Da das System eine quelloffene Referenzimplementation beeinhaltet, ist es möglich

die umgesetzte Spezifikation ausgiebig zu testen, zu nutzen und eventuell neue Funktionen hinzuzufügen. Die im Kapitel 6 durchgeführten Tests zeigen, dass das System sowohl als Dateisystem und als Streamingserver alle im Kapitel 3 formulierten Anforderungen erfüllt. Obwohl in der Referenzimplementation nicht der gesamte Funktionsumfang der Spezifikation umgesetzt wurde, ist es dennoch möglich mit dem System über die atomare Benutzerschnittstelle Daten zu verwalten und Video- bzw. Audiostreams zu übertragen.

Auf der CeBIT2003 wurde gezeigt, dass auch Videostreams mit hoher Qualität (kaum sichtbare Komprimierungsartefakte und Frameraten über 15 Bilder/s) über ein Netzwerk übertragen werden können, in dem zwar die geforderten 3 Mbit/s zur Verfügung stehen, aber mit vielen Lastspitzen und Jittereffekten zu rechnen ist.

Da die Spezifikation jedoch eine generelle Formatunabhängigkeit vorschreibt und die Referenzimplementation weder Jitterberechnung noch Flußkontrolle für die Audio- und Videoübertragung implementiert hat, kann davon ausgegangen werden, dass die Übertragungsqualität noch gesteigert werden kann.

Viele höhere Funktionen eines Dateisystems wie zum Beispiel das Suchen von Dateien, das Kategorisieren in einer Ordnerstruktur oder Datencaching wurden nicht in das Dateisystem integriert, da diese Funktionen zum Einen von speziellen Clients realisiert werden können und zum Anderen die Flexibilität verringern würden.

Die entwickelte Hashfunktion sichert dabei die eindeutige Identifikation der Daten und sichert die Unempfindlichkeit der Daten gegenüber kleinen Veränderungen wie Verrrauschen oder Abschneiden kurzer Stücke. Generell ist die Hashfunktion austauschbar, kann also noch verfeinert werden. Da die Hashfunktion ein wichtiger Teil des Systems ist, ist abzusehen, dass diese auch weiterentwickelt und verbessert wird. Dabei wird zu untersuchen sein, wie zuverlässig die Hashfunktionen Kollisionen vermeiden und Veränderungen an den Dateien ignorieren kann.

Der entwickelte Streamingserver wurde mit vollem Funktionsumfang implementiert. Dabei besitzt er zusätzlich die nicht spezifierte Funktion der Archivierung. Da eine Archivierung bei bestimmten Streams wie Audio- und Video wünschenswert ist, müssen für eine Weiterentwicklung diesbezüglich noch Untersuchungen über Archivierungsformate durchgeführt werden. In der Referenzimplementation archiviert der Streamingserver Audio- und Videodatenströme generell in eine AVI-Datei, also eine Videodatei. Dadurch benötigt die Archivierung allerdings jeweils den Codec, der zur Codierung des Datenstromes auf dem Client verwendet wurde.

Da die Spezifikation und die Referenzimplementation auch der OpenSource Gemeinde nach Abschluss der Arbeit zur Verfügung gestellt werden, wird die Weiterentwicklung des Systems eine Menge Verbesserungen der Referenzimplementation zur Folge haben, wodurch die Benutzbarkeit noch einmal verbessert werden kann. Durch die Unterstützung mehrerer Entwickler kann Know-How zum Beispiel aus der Codecentwicklung dazu führen, dass mehrere in Java implementierte Video-und Audiocodecs die Video- und Audiounterstützung erweitern und die Benutzbarkeit um noch ein Weiteres steigern. Des weiteren kann die Entwicklung einer Flusskontrolle bei Video- und Audiodatenströmen die Benutzbarkeit erhöhen.

Als Ausblick kann also eine starke Weiterentwicklung und Verbesserung der Referenzimplementation in Aussicht gestellt werden. Es könnte zum Beispiel ein Client zum Wiedergeben aller unterstützter Audio- und Videoformate entwickelt werden, mit dem auch Playlisten erstellt und im System gespeichert werden können. Eine weiterer Client könnte den Workflow für Videokonferenzen zur Verfügung stellen. Dabei sollte dieser Client einen Audio- und einen Videostream getrennt hochladen und für alle Konferenzteilnehmer eine Berechtigung mit eindeutiger Sitzungsnummer erstellen. Des weiteren muss dieser Client regelmäßig nach neuen Berechtigungen mit dieser Sitzungsnummer schauen um neue Teilnehmer aufzunehmen. Eine Dauerverbindung über einen Kommunikations- und Steuerstream könnte dabei ankommende Gespräche signalisieren.

8. Literaturverzeichnis

MASCHU_1996:
Martin, Hans-Peter; Schumann, Harald, Die Globalisierungsfalle , 1996, Reinbek (Rowohlt)

SEMPERE_1997:
Javier Gozálvez Sempere, An overview of the GSM system, 1997, University of Strathclyde, http://www.comms.eee.strath.ac.uk/~gozalvez/gsm

APANT_2003:
Apache, Apache ANT Homepage, 2003, Apache, http://ant.apache.org/

APMAVEN_2003:

Apache, Apache MAVEN Homepage, 2003, Apache, http://www.maven.org

GOEHR_2001:
Dennis Göhr, Bewertung der Dienstgüte von Audio- und Videodiensten, 2001, Institut für Datentechnik und Kommunikationsnetze, http://www.rvs.uni-hannover.de/arbeiten/diplom/da-goehr

JBUILDERLN_2003:
Borland, Borland JBuilder, 2003, Borland, http://www.borland.com/jbuilder

APCOCOON_2003:
Apache, Cocoon, 2003, Apache, http://cocoon.apache.org/2.0

CVS_2003:
Concurrent Version System, CVS-Homepage, 2003, Collabnet, http://www.cvshome.org/

AC3_1995:
ATSC, Digital Audio Compression (AC3), 1995, ATSC, http://www.linuxvideo.org/devel/library/ac3-standard

WALSH_2003:
Norman Walsh, DocBook Homepage, 2003, Oasis-Open, http://www.docbook.org

ECLIPSELN_2003:
IBM, Eclipse Homepage, 2003, IBM, http://www.eclipse.org

EDONKEY_2003:
eDonkey, eDonkey Netzwerk, 2003, MetaMachine, http://www.edonkey2000.com

DEMICHIEL_2003:
Linda G. DeMichiel, Enterprise Java Beans Specification, Version 2.1, 2003, Sun Microsystems, http://java.sun.com/products/ejb/docs.html

ETSIGSM_1992:
ETSI, European digital cellular telecommunications system, 1992, ETSI, http://www.etsi.org

FFMPEGLN_2003:
Unbekannter Autor, FFMPeg Project Page, 2003, FFMPEG, http://ffmpeg.sourceforge.net/

APFOP_2003:
Apache Software Group, FOP Homepage, 2003, Apache Software Group, http://xml.apache.org/fop

JMFLN_2003:
Sun Microsystems, Java Media Framework Homepage, 2003, Sun Microsystems,

http://java.sun.com/products/java-media/jmf

JRELN_2003:
Sun Microsystems, Java Runtime Environment, 2003, Sun Microsystems, http://www.java.com

JWSLN_2003:
Sun Microsystems, Java Web Start Homepage, 2003, Sun Microsystems, http://java.sun.com/products/javawebstart

JBOSSLN_2003:
JBOSS, JBOSS Homepage, 2003, JBOSS, http://www.jboss.org

MISOLN_2003:
Microsoft, Microsoft Homepage, 2003, Microsoft, http://www.microsoft.com

SIKORA_2003:
Thomas Sikora, MPEG-1 and MPEG-2 Digital Video Coding Standards, 2003, Heinrich-Hertz-Institut Berlin, http://wwwam.HHI.DE/mpeg-video/papers/sikora/mpeg1

MP2LN_2003:
Fraunhofer IIS, MPEG-2 Advanced Audio Coding: Data Compression for the 21st Century, 2003, Fraunhofer IIS, http://www.iis.fraunhofer.de/amm/techinf/aac/index

FIREI_2000:
F. Fitzek; Martin Reisslein, MPEG-4 and H.263 Video Traces for Network Performance Evaluation, 2000, TU Berlin, http://www-tkn.ee.tu-berlin.de/~fitzek/conferences

MP3LN_2003:
Fraunhofer IIS, MPEG Audio Layer-3, 2003, Fraunhofer IIS, http://www.iis.fraunhofer.de/amm/techinf/layer3

BEHA_2001:
Norman Beck; Ulrich Hanff, NVACS - ein Tool zur schmalbandigen Audio- und Videokommunikation, 2001, Universität Leipzig

HANNAN_1995:
J.H.L. Hansen; S. Nandkumar, Objective Quality Assessment and the RPE-LTP Vocoder in Different Noise, 1995, The Journal of the Acoustical Society of America, http://cslr.colorado.edu/rspl/PUBLICATIONS/PDFs

FREYERMUTH_2001:
Gundolf S. Freyermuth, Offene Geheimnisse, Artikel über Historie und Entwicklung der OpenSource Gemeinde, 2001, Heise Verlag, http://www.heise.de/ct/01/20/176

CTOS_2003:
c't, OpenSource entlastet die Firmenkasse, 2003, Heise Verlag,

http://www.heise.de/newsticker/data/ola-07.05.03-0

IFSLN_2003:
Oracle, Oracle Internet File System, 2003, Oracle,
http://otn.oracle.com/products/ifs/content.html

KOENEN_2002:
Rob Koenen, Overview of the MPEG-4 Standard, 2002, ISO/IEC,
http://mpeg.telecomitalialab.com/standards/mpeg-4

REALHP_2003:
Real Networks, Real Helix Producer, 2003, Real Networks,
http://www.realnetworks.com/products/producer

REAL_2003:
Real Networks, Real Networks Homepage, 2003, Real Networks,
http://www.realnetworks.com/

REALPL_2003:
Real Networks, Real Player Homepage, 2003, Real Networks, http://www.real.com

H263LN_2003:
ITU-T, Recommendation H.263, "Video coding for low bit rate communication",Jahr unbekannt , ITU-T

HANFF_2001:
Ulrich Hanff, RTP/RTSP versus HTTP als Transportprotokoll für Streaming, 2001, Universität Leipzig

CASSCHU_1996:
H. Schulzrinne; S. Casner; R. Frederick; V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, 1996, RFC 1889 (Proposed Standard), http://www.rfc-editor.org/rfc/rfc1889.txt

SVG_2003:
W3C, Scalable Vector Graphics (SVG) 1.0 Specification, 2003, W3C,
http://www.w3.org/TR/2001/REC-SVG-20010904

SOURCEFORGELN_2003:
Sourceforge, Sourceforge Homepage, 2003, Sourceforge, http://www.sf.net

MANDUCHI_2002:
Roberto Manduchi, Speech Compression, 2002, CMPE 250,
http://www.cse.ucsc.edu/classes/cmpe250/Fall02

NETCRAFT_2003:
Netcraft, Statistiken über Marktanteile bei Webservern, 2003, Netcraft,
http://news.netcraft.com/archives/2003/04/13/april

SVCD_2003:

Unbekannter Autor, Super Video CD Standard IEC 62107, 2003, private, http://lea.hamradio.si/~s51kq/V-SVCD-S.HTM

TCP_1981:
University of Southern California, Transmission Control Protocol, 1981, RFC 793 (RFC793), http://www.faqs.org/rfcs/rfc793.html

POSTEL_1980:
J. Postel, User Datagram Protocol, 1980, RFC 768 (RFC768), http://www.faqs.org/rfcs/rfc768.html

NIANLI_1996:
Ze-Nian Li, Video Compression, 1996, CMPT 365, http://www.cs.sfu.ca/undergrad/CourseMaterials

STEIN_2003:
Greg Stein, WebDAV Homepage, 2003, Collabnet, http://www.webdav.org

WEBSTATLN_2003:
WebHits, Web Statistiken, 2003, WebHits, http://www.webhits.de/deutsch/webstats.html

SCHULTE_2001:

Karl-Wilhelm Schulte, Wem gehört das Web? Statistiken und Schlussfolgerungen, 2001, Bergische Universität Wuppertal, http://www.hrz.uni-wuppertal.de/infos/hrz-info/hrz

9. Abbildungsverzeichnis

10. Stichwortverzeichnis

11. Anlagenverzeichnis

Alle Anlagen befinden sich auf der CD. Der Verzeichnisbaum untergliedert sich wie folgt:

artikel	Verzeichnis
grafiken	Verzeichnis
lsR.txt	Verzeichnis
maci	Verzeichnis
maci_umls	Verzeichnis
openoffice	Verzeichnis
screenshots	Verzeichnis
videos	Verzeichnis

./artikel: **Artikeltexte**

CT_Artikel_zu_TCO_bei_OpenSource.txt

CT_Artikel_zur OpenSource_Oekonomie.txt

./grafiken: **Grafiken, Abbildungen, Screenshots**

corona_concept.png

h263_64_1.png

h263p_64-1.png

mpg4_64-1.png
mpg_64-1.png
original.png
system_impl_structure.png
system_structure.png
wmv_64-1.png

./maci:	**Projektimplementation**
./maci/src:	Quelltexte
./maci/src/conf:	Konfigurationsdateien
./maci/src/conf/META-INF:	Konfigurationsdateien
./maci/src/java:	Java-Quelltexte
./maci/src/webapp:	Quelltexte der Webapplikation
./maci/src/webapp/WEB-INF:	Bibliotheken und Konfigurationsdateien
./maci/src/webapp/pic:	Bilder für die Webapplikation
./maci/target:	Projektausgabepfad

./maci_umls: **UML-Diagramme**

abstractfile.png
filepermission.png
permissionwrapper.png

./openoffice: **Original-Grafiken im OpenOffice-Format**

corona_concept.sxd
system_impl_structure.sxd
system_structure.sxd

./screenshots: **Screenshots der Webapplikation**

auslastung.png
create_user.png
download_archived_Stream.png
downstream1.png
downstream2.png

files.png
files_with_stream.png
haupt_1.png
login.png
screen1.jpg
upload.png
upstream_1.png
upstream_choose_format.png

./videos:	**Beispielvideos für die Kodierung**
em2.mpg	in MPEG1 umkodierter HFTV-Trailer
emotion.mpg	original HDTV-Trailer
h263_1000.avi	
h263_2000.avi	
h263_256.avi	
h263_4000.avi	
h263_512.avi	
h263_64.avi	
h263p_1000.avi	
h263p_2000.avi	
h263p_256.avi	
h263p_4000.avi	
h263p_512.avi	
h263p_64.avi	
mpg4_1000.avi	
mpg4_2000.avi	
mpg4_256.avi	
mpg4_4000.avi	
mpg4_512.avi	
mpg4_64.avi	
mpg_1000.mpg	
mpg_2000.mpg	
mpg_256.mpg	
mpg_4000.mpg	

mpg_512.mpg

mpg_64.mpg

wmv_1000.avi

wmv_2000.avi

wmv_256.avi

wmv_4000.avi

wmv_512.avi

wmv_64.avi

Urheberrechtliche Erklärung:

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Diplomarbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.