Diplomarbeit

zum Thema

VERNER

ein Video EnkodeR uNd -playER für DROPS

Carsten Rietzschel
cr7@os.inf.tu-dresden.de
Technische Universität Dresden
Fakultät Informatik

10.09.2003

Diese Diplomarbeit ist meinem verstorbenen Vater Werner gewidmet.

Besonderer Dank geht an meinen Betreuer Lars Reuther - für seine zahlreichen nützlichen Hinweise und sein offenes Ohr für Vorschläge und Ideen. Dank auch an Norman Feske für seine DOpE-Tips sowie an Christian Helmuth für seine Unterstützung bei Sound-Problemen. Ebenfalls dankbar bin ich allen hier nicht namentlich genannten Probelesern und den Mitarbeitern des Lehrstuhls Betriebssysteme für ihre hilfreichen Anmerkungen.

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich diese Diplomarbeit selbständig erstellt und nur aufgeführte und legale Hilfsmittel genutzt habe.
Dürrröhrsdorf, den 10.09.2003

Carsten Rietzschel

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Vorarbeiten

Ziel dieser Projektarbeit ist die Entwicklung und Implementierung eines echtzeifähigen Videorecorders für das *Dresden Real-Time OPerating System*, kurz *DROPS*. Als Ausgangspunkt dienen dazu die in der Vorarbeit "Portierung eines Video-Codecs auf DROPS" [1] gewonnenen Erkenntnisse. Dieses Projekt wird im Rahmen der vorliegenden Diplomarbeit fortgesetzt.

Im Rahmen des Großen Belegs wurde der Video-Codec XviD auf DROPS portiert. Er soll die Kodierung der Bilddaten beim Videorecorder übernehmen. Einige der wichtigsten Erkenntnisse in bezug auf Echtzeitfähigkeit waren, daß es weder im Rahmen des Projektes möglich ist, einen echtzeitfähigen Codec neu zu entwickeln, noch einen bestehenden Video-Codec nachträglich echtzeitfähig zu machen. Daher war die logische Konsequenz, einen nicht-echtzeitfähigen Codec als Basis für den Videorecorder zu verwenden und eine Infrastruktur um einen nicht-echtzeitfähigen Codec zu entwerfen, die trotzdem Echtzeitzusagen so gut wie möglich einhält. Die Annahme, daß durch das Einplanen der Worst-Case-Zeiten des Codecs für das Scheduling trotzdem ein leistungsfähiger und echtzeitfähiger Videorecorder entwickelt werden kann, wurde widerlegt. Laut Entwicklern von XviD ist die Annahme realistisch, daß die übliche Kodierzeit um Faktor einhundert geringer sei als die Worst-Case-Zeit.

Um einen Kompromiß zwischen Echtzeitfähigkeit und hoher Leistung eines Videorecorders auf Basis eines nicht-echtzeitfähigen Codecs zu erreichen, wurde im Beleg das *Konzept der Qualitätsstufen-Anpassung* (*kurz QAP*) entwickelt. Dieses besagt, daß bei hoher Systemlast die visuelle Qualität gesenkt wird, um möglichst alle Daten kodieren zu können. Andererseits wird die Qualität und damit die benötigte Rechenzeit gesteigert, wenn die Auslastung des Systems dies zuläßt. Voraussetzung dafür ist es, daß der verwendete Codec dieses Vorgehen unterstützt. Für XviD konnte dies bestätigt werden.

Desweiteren wurde eine Testumgebung entwickelt, deren Infrastruktur bereits im Hinblick auf die Implementierung des Videorecorders ausgerichtet ist. Auch dabei konnten Erfahrungen gesammelt werden, die in diese Arbeit einflossen.

Offengeblieben sind in der bisherigen Arbeit hin zum Videorecorder eine Reihe von Aufgaben und Fragen. Beispielsweise wurde kaum auf die Aufnahme und Wiedergabe von Audio, sowie dem damit verbundenen Problem der Synchronität zwischen Bild und Ton eingegangen. Zur Audioverarbeitung wurde bisher ebenfalls keine Software implementiert. Eine weitere Aufgabe ist die Einbindung des Systems in das DROPS Scheduling Framework, zu dem bisher auch keine konkreten Pläne vorgestellt wurden. Der Lösung der offengebliebenen Fragen widmet sich diese Diplomarbeit.

1.2 Begriffsbestimmung

Die wichtigsten Begriffe, die in dieser Arbeit Verwendung finden, werden an dieser Stelle vorgestellt und sollen zum weiteren Verständnis beitragen. Diese und andere verwendete Begriffe sind ebenfalls im Glossar erklärt. Zu finden ist dies im Anhang A.

Akustisch Wahrnehmbares wird i. allg. als *Audio*-Material bezeichnet. *Video* dagegen bezeichnet im ursprünglichen Sinne eigentlich nur visuell erfaßbare Daten. Hingegen wird das Wort *Videos* (Plural) oft im Sinne von Filmen, also der Kombination von visuellem und akustischem Material verwendet. Ein Algorithmus, der Daten enkodiert *(*en*CO*de, engl.) und dekodiert (*DEC*ode, engl.), wird als

Codec bezeichnet. Ein *Video-Codec* dient entsprechend der De- bzw. Enkodierung von Bildmaterial, ein *Audio-Codec* der Verarbeitung von Tonsignalen. Ziel der Enkodierung ist es, das zu bearbeitende Ausgangsmaterial zu *komprimieren*, also die Datenmenge zu reduzieren. Moderne Codecs nutzen dazu eine verlustbehaftete Kompression, d.h aus dem kodierten Material können die Ausgangsdaten nicht mehr exakt rekonstruiert werden. Daraus resultiert das Bestreben, daß die dekodierten Daten den originalen Ausgangsdaten so ähnlich wie möglich sind. Im allgemeinen wird dies als ein Maß für die *visuelle Qualität* eines Video-Codecs bzw. als Maß für die *akustische Qualität* eines Audio-Codecs gewertet.

Bei aktuellen Video-Codecs erfolgt die Bearbeitung des Videomaterials auf Basis von Einzelbildern, den sogenannten *Frames*. Dabei bezeichnet die Framerate die Anzahl der Bilder innerhalb einer gewissen Zeit und wird meist in Frames pro Sekunde (kurz fps) angegeben. Es kann vorkommen, daß aus bestimmten Gründen einzelne Bilder ausgelassen werden. Dies wird *Framedrops* genannt. Das kontinuierliche Übertragen von Audio- oder Videomaterial über verschiedene Medien ohne die Notwendigkeit einer Zwischenspeicherung wird als *Streaming* bezeichnet. Die korrekte Darstellung von Videos erfordert, daß die Beziehungen zwischen Bild und Ton bekannt sind. Ebenso muß die genaue Abspielgeschwindigkeit ermittelbar sein. Zur Speicherung der eben genannten Informationen wird ein *Container* verwendet. Ein Container ist ein Konstrukt, welches unabhängig vom verwendeten Codec Bild und Ton enthält (contain, engl.). Moderne Container unterstützen neben Audio und Video auch weitere Inhalte wie z.B. Untertitel.

Viele andere Begriffe stammen aus dem Bereich der Betriebssysteme [2]: Ein *Task* bezeichnet eine durch die Computerverarbeitung berechenbare Aufgabe. Ein *Job* entspricht dabei einem Aufgabenteil. Charakterisiert ist ein Job durch seine *Ausführungszeit* - die Dauer, die zur Berechnung des Jobs gebraucht wird. Dabei kann es auftreten, daß in Abhängigkeit der Daten die Bearbeitung schneller oder langsamer erfolgt. Die maximal auftretende Zeit unter den ungünstigsten Bedingungen wird als *Worst-Case-Zeit* bezeichnet. Ein *Schedule* (engl. Ablaufplan) ordnet den Jobs *Betriebsmittel* zu. Im Rahmen dieser Arbeit ist unter diesem Betriebsmittel hauptsächlich der Zentralprozessor, also die CPU, zu verstehen. Das Binden eines Jobs an ein konkretes Betriebsmittel wird als *Scheduling* bezeichnet. Unter bestimmten Bedingungen kann ein Job nicht an ein Betriebsmittel gebunden werden. Die Entscheidung darüber, ob ein Job zugelassen werden kann oder nicht, wird *Admission* genannt.

1.3 Aufbau der Arbeit

Das folgende Kapitel informiert über die Grundlagen dieser Arbeit. Dabei werden aktuelle Entwicklungen rund um das *Dresden Real-Time OPerating System* (kurz *DROPS*) vorgestellt. Zum besseren Verständnis dieser Arbeit werden einige Inhalte der zugrunde liegenden Arbeit [1] erneut wiedergegeben. Das betrifft insbesondere die Abschnitte 2.1 (Echtzeitfähigkeit) und 2.2 (DROPS). Die beschriebenen Themen wurden bereits im gleichen bzw. ähnlichen Wortlaut innerhalb des zitierten Werkes veröffentlicht. Einige Abbildungen wurden ebenfalls in der genannten Arbeit verwendet. Desweiteren werden Details zu Video- und Audio-Codecs sowie zu Container-Formaten erläutert. Bereits vorhandene Software zum Thema Videorecorder wird ebenfalls kurz erwähnt.

Kapitel 3 widmet sich den Modellen des Videorecorders und -players. Diese werden erarbeitet und erläutert. Dabei wird der Bezug zum Echtzeitbetriebssystem hergestellt, sowie grundlegende Konzepte und Ideen vorgestellt. Der konzeptionelle Aufbau des zu entwerfenden Synchronisations-

mechanismus wird detailliert erörtert. Im vierten Kapitel wird die Implementierung der entwurfenen Modelle erklärt, sowie die dabei verwendete Software vorgestellt. Außerdem wird die Kommunikation und der Datentransfer zwischen den Komponenten der Modelle beleuchtet. Dem folgen Implementierungsdetails zur erstellten Synchronisationskomponente. Das vorletzte Kapitel beschäftigt sich mit der Bewertung der vorgestellten Konzepte sowie der Implementation. Dabei werden kurz Erwartungen geäußert, sowie Ergebnisse in bezug auf diese analysiert. Darauf folgt eine Zusammenfassung dieser Arbeit im letzten Kapitel.

2 Grundlagen und verwandte Arbeiten

2.1 Echtzeitfähigkeit

Echtzeitfähigkeit ist gleichbedeutend mit der Fähigkeit eines Systems, alle seine *Deadlines* zu erreichen, d.h. das System muß alle seine Berechnungen innerhalb einer durch die Umgebung festgelegten Zeitspanne, deren Ende als Deadline bezeichnet wird, abgeschlossen haben. Ein wichtiger Punkt der Echtzeitfähigkeit eines Systems (*Echtzeitsystem*, *Real-Time System*) ist die Kenntnis der Dauer der durchzuführenden Berechnungen. Eine mögliche Gruppierung von Echtzeitsystemen kann dahingehend vorgenommen werden, daß

1. die Deadlines unbedingt eingehalten werden müssen oder

2. die Deadlines mit bestimmter Wahrscheinlichkeit eingehalten werden sollten.

Der erstgenannte Fall wird als *hartes Echtzeitsystem* bezeichnet, letzterer als *weiche*s[3].

Relevant für die Wiedergabe von Videos sind die Eigenschaften der menschlichen Sinnesorgane. So nimmt der Mensch Filme, die mit 25 Bildern pro Sekunde und mehr dargestellt werden, flüssig wahr. Auch sollte das Bild und der zugehörige Ton nicht weiter als $\pm$80ms versetzt abgespielt werden, da dies sonst als nicht lippensynchron empfunden wird [4]. Diese Bedingungen bestimmen die Parameter für das Scheduling eines Echtzeitsystems zum Aufzeichnen und Wiedergeben von Videos. So ergibt sich ein sich wiederholender Vorgang mit einer *Periode* von höchstens 40ms. In dieser Periode muß ein komplettes Bild verarbeitet werden.

Desweiteren ist anzumerken, daß ein Aussetzen der Audiowiedergabe als störender empfunden wird, als ein Auslassen von Bildern. Daher ist die Audioverarbeitung gegenüber der von visuell erfaßbaren Material zu bevorzugen.

2.2 DROPS

Das *Dresden Real-Time OPerating System (DROPS)* ist ein an der TU-Dresden entwickeltes Forschungsprojekt. Ziel dieses Projektes ist es, Quality-of-Service-Forderungen von Anwendungen zu unterstützen [5]. DROPS ermöglicht das Betreiben von Echtzeitanwendungen zusammen mit Timesharing-Anwendungen. Es basiert auf einem Microkernel der 2. Generation - dem *Fiasco*-Microkernel. Ein wichtiger Bestandteil ist L4Linux. Dies ist ein Linux-Server, der auf dem Microkernel aufsetzt und die Möglichkeit bietet, "Time-sharing"-Komponenten betreiben zu können. So können weiterhin normale Linux-Anwendungen gleichzeitig mit Echtzeit-Anwendungen ausgeführt werden. Abbildung 1 zeigt schematisch die Struktur von DROPS.

2.2.1 DROPS Streaming Interface

Das DROPS Streaming Interface (kurz *DSI*) bietet eine schnelle und komfortable Möglichkeit, größere Datenmengen sowohl zwischen verschiedenen Adreßräumen als auch innerhalb eines Adreßraumes auszutauschen. Realisiert ist dies durch zwei shared-memory-Bereiche und durch Kontrollfunktionen, die in Form einer Bibliothek verfügbar sind. Durch ein vom Microkernel unterstütztes Einblenden von

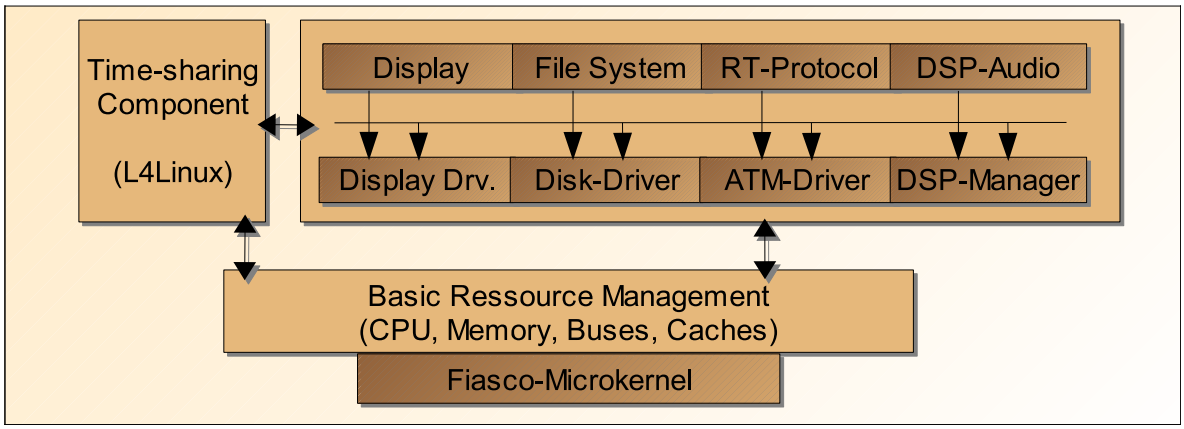

Abbildung 1: DROPS aus [5]

Seiten in die beteiligten Adreßräume, dem sogenannten *Mapping*, wird der Datenaustausch ohne zusätzliche und damit langsame Kopieroperationen realisiert. Eine typische Beispielapplikation ist das in Abbildung 2 gezeigte Erzeuger-Verbraucher-Problem.

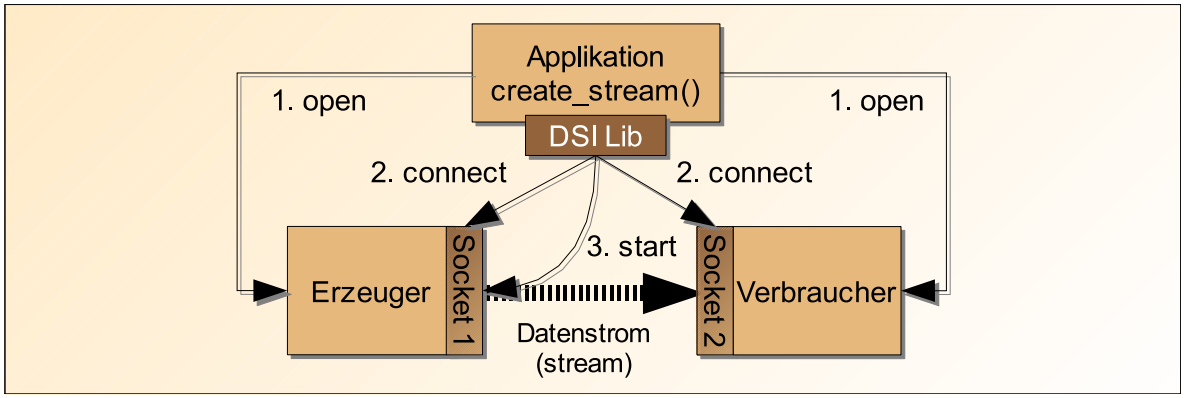

Abbildung 2: DSI: Struktur des DROPS Streaming Interface aus [6]

Nach dem Initialisieren des DROPS Streaming Interface mit *dsi_init* können die zur Datenübertragung nötigen Funktionen aufgerufen werden. Der Datenaustausch erfolgt über einen Datenstrom, der von der Anwendung mit *create_stream* erstellt wird. Er verbindet die *sockets* des Erzeugers und des Verbrauchers. Dieser *stream* sorgt danach selbständig für den Austausch der Daten. Als Transporter dienen dabei eine endliche Anzahl von Paketen, welche in einem Ringpuffer organisiert sind. Die Verwaltung der Pakete erfolgt in deren Kontrollbereich. Die für die zu transportierenden Daten zuständige Region des Hauptspeichers wird Datenbereich genannt. Er unterliegt keiner Verwaltung oder Kontrolle durch DSI. Für das DROPS Streaming Interface ist der Datenbereich ein unstrukturierter Teil im Adreßraum. Sowohl auf Erzeuger- als auch auf Verbraucherseite werden die gleichen Bibliotheksfunktionen genutzt. Zum Beispiel liefert die Funktion *get_packet* einen Paketdeskriptor, während *commit_packet* einen Deskriptor freigibt. Daten werden mit *add_data* vom Erzeuger bereitgestellt, während *get_data* verbraucherseitig die Daten liefert. Abbildung 3 demonstriert diese Vorgänge.

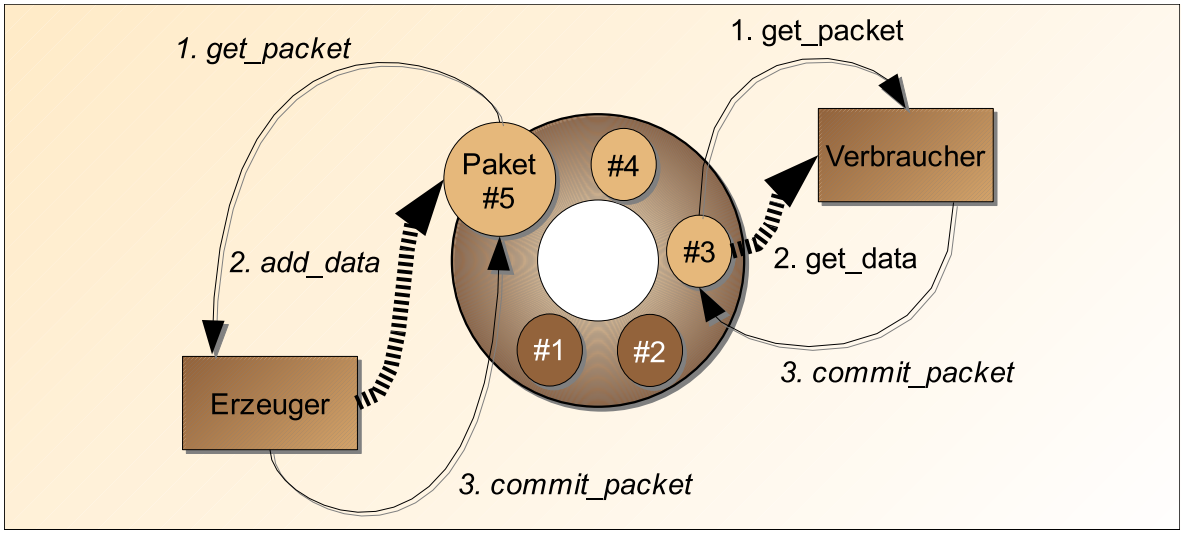

Abbildung 3: DSI: Ringpuffer mit Paketfunktionen

Zusammenfassend lassen sich folgenden Vorteile von DSI nennen:

- einfache Schnittstelle gekapselt in einer Client-Bibliothek (DSI Lib)
- Reduzierung der notwendigen Kopieroperationen
- die Struktur der zu übertragenden Daten ist irrelevant für das Funktionieren von DSI

2.2.2 Quality-Assuring Scheduling

Oft verwenden Scheduling-Verfahren die maximale Ausführungszeit für die Einplanung eines Jobs bei der Berechnung des Ablaufplans. Am Beispiel der Videoenkodierung läßt sich demonstrieren, daß dieser Ansatz i. allg. keine hohe Systemauslastung ermöglicht: Für ein Video, bei dem alle 33,36ms ein komplettes Bild verarbeitet werden muß, wurde eine Worst-Case-Bearbeitungszeit für das Enkodieren von ca. 30ms pro Frame gemessen (vgl. Anhang F). Diese Zeiten werden für das Scheduling eingeplant. Die durchschnittliche Bearbeitungszeit beträgt aber lediglich etwa 19ms. Damit ergibt sich, daß im Schnitt nur etwa 63 Prozent der reservierten Rechenzeit verwendet werden. Das System ist nicht vollständig ausgelastet.

Mit *Quality-Assuring Scheduling*, kurz *QAS*, wird versucht, das System so weit wie möglich auszulasten und trotzdem den Anwendungen gemachte Echtzeitzusagen einzuhalten [7]. Eine Voraussetzung ist, daß periodisch zu wiederholenden Aufgabe vorliegen, was bei Videoverarbeitung der Fall ist. Das zu QAS gehörige Taskmodell ist in Abbildung 4 dargestellt. Ein Task wird in mehrere Teile zerlegt: in maximal einen zwingend erforderlichen Teil (mandatory, engl.) und mehrere optionale Teile. Der mandatorische Teil wird mit der Worst-Case-Ausführungszeit eingeplant. Dies stellt sicher, daß dieser garantiert ausgeführt wird. Dies entspricht einem harten Echtzeitsystem.

Die optionalen Teile entsprechen Applikationen, bei denen das Überschreiten einer Deadline keine dramatischen Folgen hat. Optionale Teile sollen so oft wie möglich, aber mindestens mit einem vom Benutzer festgelegten Prozentsatz ausgeführt werden. Zur Berechnung der für das Scheduling einzuplanenden Zeit, auch *Reservierungszeit* genannt, in Abhängigkeit der gewünschten Qualität

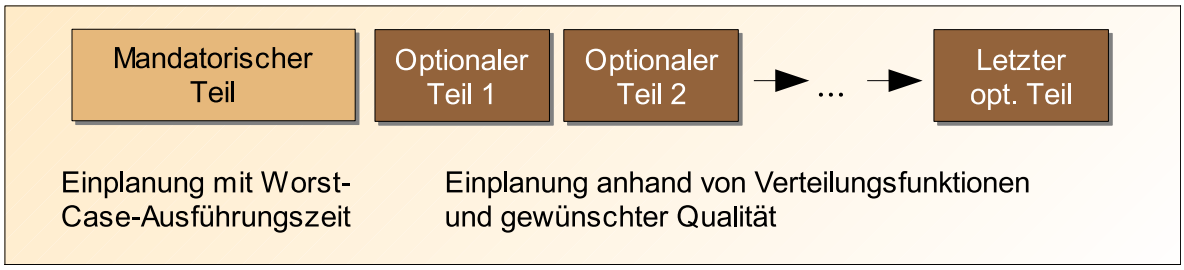

Abbildung 4: QAS: Taskmodell

nutzt QAS die Verteilungsfunktion der Ausführungszeiten für die optionalen Teile. Die Qualität bezeichnet dabei, wieviel Prozent der optionalen Teile komplett ausgeführt werden. Für den Fall, daß es nur einen unterbrechbaren optionalen Teil gibt, illustiert die Abbildung 5 die Bestimmung der Reservierungszeit. Diese Illustration ist erheblich vereinfacht, da es auch vorkommen kann, daß ein optionaler Teil nicht nur durch das Ende der Reservierungszeit vorzeitig unterbrochen wird, sondern aufgrund des Periodenendes gar nicht erst gestartet wird. Das QAS zugrunde liegende Modell berücksichtigt im Gegensatz zur Illustration auch diesen Fall.

Die Nutzung der Verteilungsfunktion anstelle der Worst-Case-Zeiten ermöglicht eine höhere Systemauslastung, da Zeiten kleiner der Worst-Case-Zeit eingeplant werden. Dies kann zu temporärer Überlast führen, obwohl trotzdem der gewünschte Prozentsatz erreicht wird. Dies erhöht die Anzahl der Anwendungen, deren Ausführung vom System zugelassen wird. Das zugrunde liegende mathematische Model wird genauer in [7] vorgestellt.

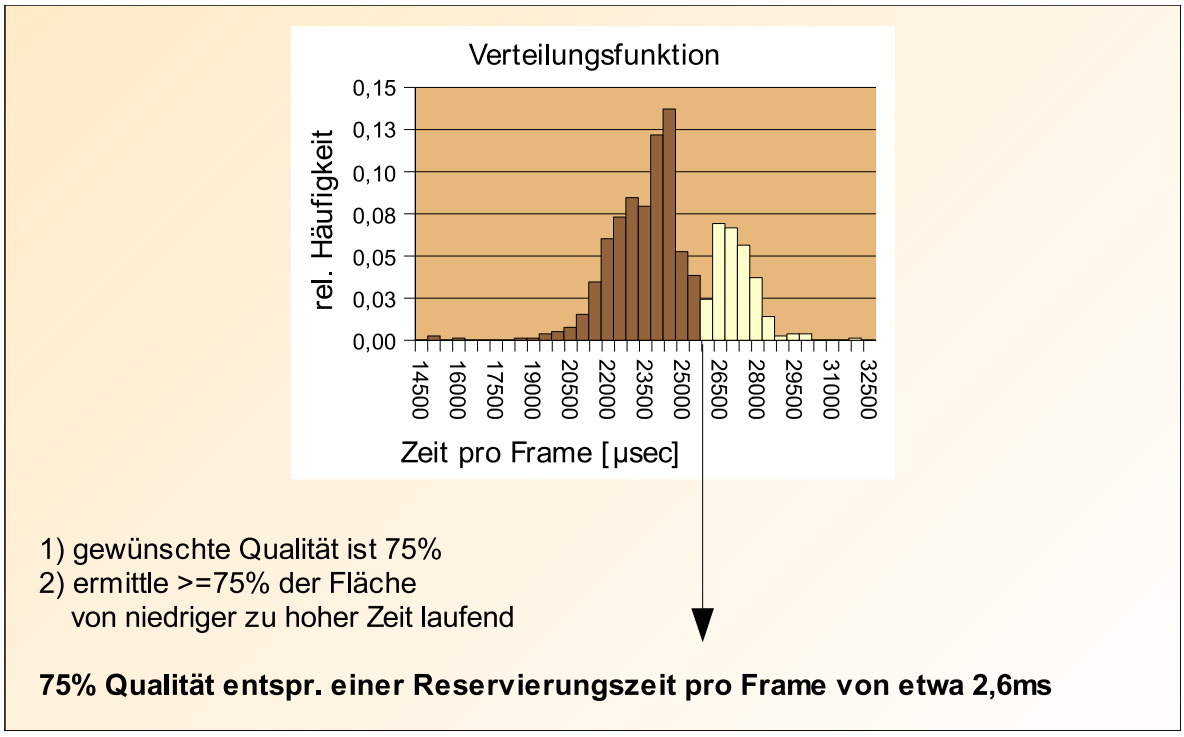

Abbildung 5: QAS: Illustration zur Bestimmung der Reservierungszeit (vereinfacht)

Als ein Beispiel für Leistungsfähigkeit des Quality-Assuring Schedulings sei aufgeführt, daß es im Gegensatz zur Worst-Case-Zeit-Planung mit QAS möglich ist, die doppelte Bandbreite einer Festplatte zu reservieren, ohne daß sich die zugesagte Qualität wesentlich verringert [7].

2.3 Video-Codecs

2.3.1 Frames

Wie bereits erwähnt, werden Videos auf Basis von Einzelbildern (Frames) verarbeitet. Um das Ziel einer möglichst hohen Kompression zu erreichen, verwenden moderne Video-Codecs eine *'inter'-Kodierung* [8]: Dabei werden, von einem vollständigen Einzelbild ausgehend nur die Unterschiede zwischen *(*inter~, engl.*)* diesem und dem folgenden Bild bzw. den folgenden Bildern kodiert und gespeichert. Beim Dekodieren muß zunächst das kodierte Referenzbild (*keyframe*, I-Frame) dekodiert werden. Erst danach können die durch 'inter'-Kodierung entstanden Frames (P- oder B-Frames) dekodiert werden. Einen Überblick über häufig verwendete Arten von Frames zeigt Tabelle 1. Als *Group of Pictures*, kurz *GOP*, wird eine Reihe von Frames von einem keyframe zum nächsten bezeichnet.

I-Frame	intra coded frame: vollständiges Einzelbild als Referenz für folgende Frames, auch als keyframe bezeichnet
P-Frame	predictive coded picture: Bildunterschiede zum vorausgehenden I- oder P-Frame
B-Frame	bidirectionally predictive coded picture: Daten zur Interpolation von benachbarten I- und P-Frame

Tabelle 1: MPEG-Frametypen aus [9]

2.3.2 Moving Pictures Experts Group (MPEG)

Die Moving Pictures Experts Group (kurz MPEG) ist eine Arbeitsgruppe der ISO/IEC, die bereits 1988 gegründet wurde [10]. Die MPEG fördert die Erstellung von Standards für kodiertes digitales Audio und Video. Unter anderem wurden verschiedene Video- und Audiostandards definiert. Darunter bekannte wie MPEG-1, MPEG-2 und MPEG-4 sowie bedeutende Audiostandards MP3 und AAC, zu denen im Abschnitt 2.4 weitere Informationen zu finden sind. Darüber hinaus sorgt die Organisation für die Definition von Konformitätstests, um eine herstellerübergreifende Kompatibilität zu fördern. Die MPEG legt im wesentlichen nur folgendes für ihre Standards fest:

- die Syntax des Video- bzw. Audioformates
- die zum Dekodieren nötigen Operationen

Dies stellt ein Abspielen der Video- bzw. Audiodaten auf kompatiblen Implementationen sicher. Die Funktionsweise der Enkoder sind jedoch nicht von der MPEG definiert. Das kann zu unterschiedlichen Qualitäten der Enkoder führen. In [11] wird dies bestätigt. Weitere Details zur Thematik MPEG können in [10], [12] sowie [13] gefunden werden.

2.3.3 MPEG-4 und XviD

Der 1998 fertiggestellte MPEG-4-Standard zur Kodierung von Video wurde 2000 als offizieller internationaler Standard anerkannt. Seitdem gab es einige Erweiterungen dieses Standards, wie MPEG-4 Version 2, welche u.a. ein Dateiformat spezifiziert. Ziel von MPEG-4 ist es, die Übertragung von Videos in guter akustischer und visueller Qualität via Internet zu ermöglichen. Dazu verwendet MPEG-4 eine flexible und objektbasierende Kodierung. Einige interessante Methoden der verwendeten Kodiertechniken wurden bereits in der zugrunde liegenden Arbeit [1] erläutert. An dieser Stelle sei dahin verwiesen.

Der MPEG-4-Standard definiert verschiedene Profile und zugehörige Level [14]. Diese Profile geben die Komplexität der Implementation vor. Sinn ist es, jeweils einige Mindestanforderungen an kompatible Implementationen zu stellen, die dafür sorgen, daß Anbieter von Hard- oder Software nicht den kompletten MPEG-4-Standard in ihren Produkten unterstützen müssen. Für diese Arbeit sind nur zwei der 19 definierten Profile relevant: das ”Simple Visual Profile” (SP) und das ”Advanced Simple Profile”, kurz ASP. Das SP definiert eine Liste von grundsätzlichen Methoden, die ein Dekoder unterstützen muß. Das ASP erweitert diese Liste um B-Frames, *Global Motion Compensation (GMC)* und *Quarter Pixel (QPel)*. Weiterführende Informationen zu diesen Stichworten sowie Betrachtungen von einzelnen MPEG-4-Implementationen sind in [14] und [11] zu finden.

Für die Implementation im Rahmen dieser Arbeit wurde der XviD-Codec verwendet. XviD ist ein unter der *GNU General Public License* (*GPL*) lizenzierter MPEG-4-konformer Video-Codec [15]. Gegenwärtig ist der XviD-Codec dem Ziel nah, dem MPEG-4-”Advanced Simple Profile” zu entsprechen. Dies trifft zumindest auf die gegenwärtige *CVS*-Version zu. Als Basis dieser Arbeit dient allerdings XviD in der Version 0.91, welcher noch nicht dem ASP genügt, jedoch ausgereifter und getesteter als die CVS-Version ist.

2.3.4 H.264/AVC

Im Jahre 2001 bündelten MPEG und die Video Coding Experts Group *(VCEQ)* ihre Entwicklungsarbeiten, um den Videostandard der nächsten Generation fertigzustellen: H.264/Advanced Coding (kurz H.264/AVC), auch bekannt als MPEG-4 Part 10 [8]. H.264/AVC verspricht gegenüber MPEG-2, der Basis für DVD-Kodierung und digitales Fernsehen, eine Steigerung der Kodiereffizienz um etwa Faktor zwei [16]. Die aus [8] auszugsweise übernommene Tabelle 2 zeigt den durchschnittlichen *Bitratengewinn* durch Kompression bei annähernd gleicher visueller Qualität gegenüber den aktuell verwendeten Videostandards MPEG-2 und MPEG-4 ASP.

Kodierer	*Bitratengewinn gegenüber MPEG-4 ASP*	*Bitratengewinn gegenüber MPEG-2*
H.264/AVC	38,62%	64,46%
MPEG-4 ASP	-	42,95%

Tabelle 2: Durchschnittlicher Bitratengewinn von H.264/AVC auszugsweise aus [8]

Im folgenden werden kurz einige Techniken vorgestellt, die zur Steigerung der Kodiereffizienz von H.264/AVC und zur Sicherung einer Übertragung via Netzwerk beitragen. Zum weiteren Verständnis sei erwähnt, daß H.264/AVC wie viele andere Codecs auch auf Blockbasis arbeitet. Dazu zerlegt es das zu kodierende Bild in quadratische Blöcke. Diese Blöcke dienen als Basis für die

Kodierung des Frames. Vorgestellt wird die Verwendung einer gegenüber MPEG-4 verbesserten Bewegungsschätzung *(Motion Estimation,* engl.*)* sowie das Bündeln von Blöcken zu *Slices*.

▷ **Slices** Die bereits erwähnten Bearbeitungsblöcke werden in sogenannten Slices zusammengefaßt: Eine Slice bezeichnet einen Bildbereich, also mehrere Blöcke, der sich unabhängig von anderen Bildbereichen kodieren läßt. Im Standard werden fünf Slice-Typen definiert, welche in Tabelle 3 erläutert werden. Diese ähneln den in der Tabelle 1 vorgestellten Frametypen. Der Zusammenfassung dieser Blöcke zu Slice-Gruppen sind dabei kaum Grenzen gesetzt - einen kleinen Teil möglicher Gruppierungen zeigt das aus [8] stammende Bild 6 .

Slice-Type	***Bemerkung***
I-Slice	ähnlich I-Frame, alle Daten können ohne Referenzen auf andere Slices dekodiert werden
P-Slice	ähnlich P-Frame, Daten können mit Hilfe vorheriger Slices dekodiert werden
B-Slice	ähnlich B-Frame, zum Dekodieren werden Daten aus vorherigen und folgenden Slices benötigt
SP-Slice SB-Slice	wie P- bzw. B-Frame, nur das diese Slice für das Umschalten (Switching, engl.) auf andere Bitraten optimiert sind

Tabelle 3: H.264/AVC: Slicegruppen aus [8]

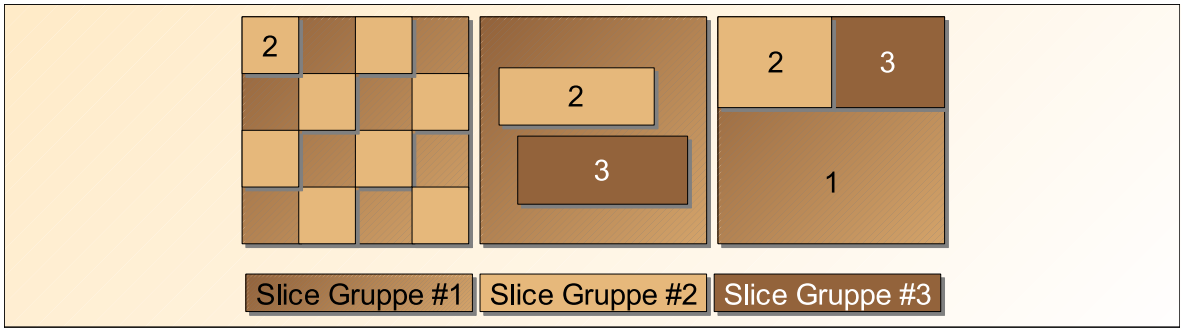

Abbildung 6: H.264/AVC: mögliche Muster für Slice-Gruppen aus [8]

Bei bisherigen Verfahren wurden jeweils komplette Frames übertragen. Im Gegensatz dazu, werden bei H.264/AVC Slice-Gruppen getrennt übermittelt. Durch die Nutzung von benachbarten, korrekt empfangenen Slice-Gruppen kann der Verlust einer Gruppe auf dem Übertragungsweg besser ausgeglichen werden. Damit sind wenigstens Teile eines Frames dekodierbar.

▷ **Motion Estimation** Die Bewegungsschätzung versucht zeitliche Redundanz innerhalb von Bildfolgen zu bestimmen und diese so weit wie möglich zu eliminieren, um Speicherplatz zu sparen und damit eine höhere Kompression zu erreichen. Neu gegenüber MPEG-1 und -2 ist bei H.264/AVC unter anderem, daß die kodierten Frames sich auf mehrere Referenzbilder beziehen können, was Abbildung 7 [8] verdeutlicht.

Ein konkreteres Beispiel ist in der aus [16] stammenden Illustration 8 dargestellt. Sie zeigt, wie sich mehrere Referenzframes dazu nutzen lassen, eine periodische Bewegung besser zu erkennen und entsprechend genauer zu kodieren.

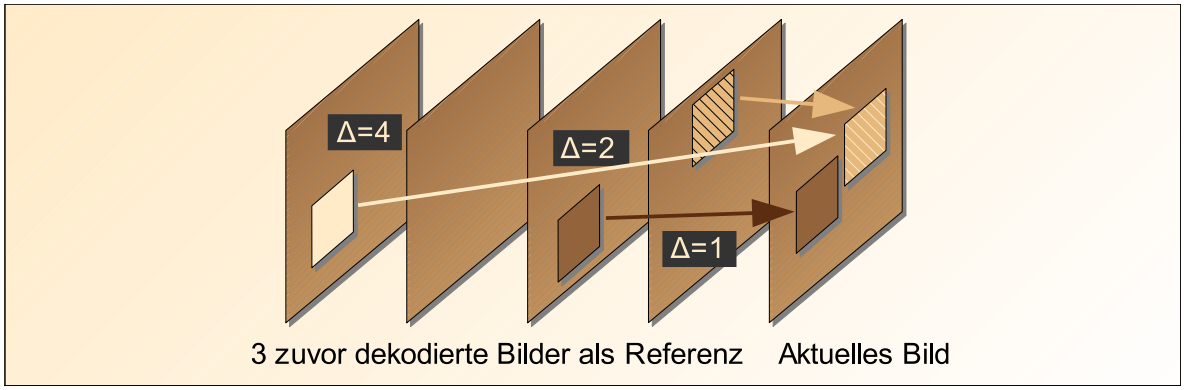

Abbildung 7: H.264/AVC: mehrere Referenzbilder aus [8]

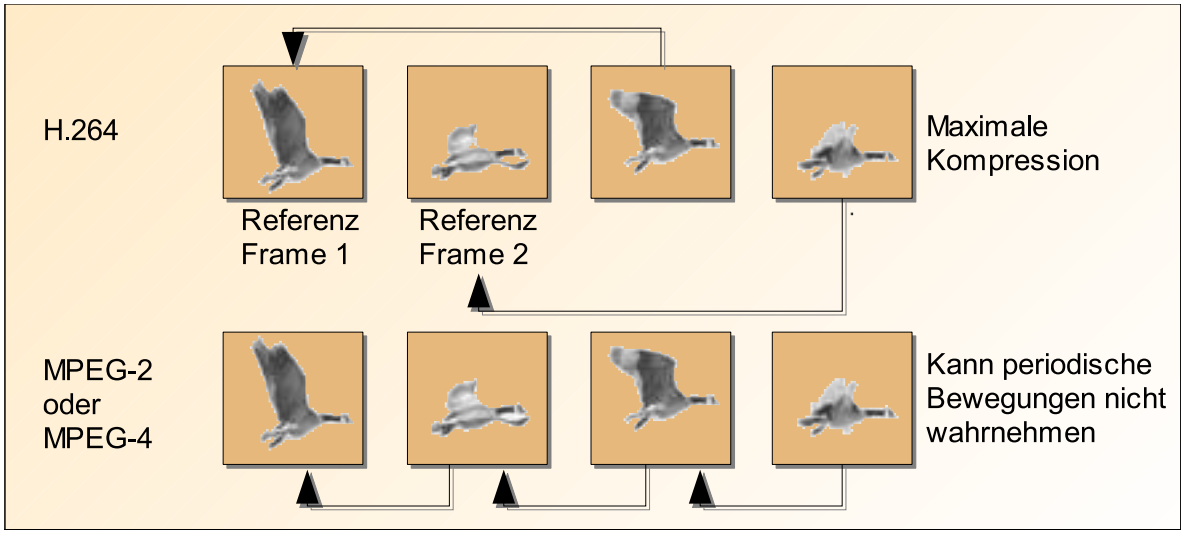

Abbildung 8: H.264/AVC: Motion Estimation aus [16]

2.4 Audio-Codecs

In diesem Abschnitt werden alle wesentlichen Informationen zum Thema Audio-Codecs vorgestellt, die grundlegend zum Verständnis der folgenden Kapitel sind. Dazu zählen in erster Linie das *PCM*-Format, sowie MPEG-Audio. Weiterhin sind auch aktuelle Informationen zu den neueren Audiokompressionsverfahren zu finden: *Ogg Vorbis* und *Advanced Audio Codec* (kurz AAC).

2.4.1 PCM

PCM ist das Kürzel für *Pulse Code Modulation* [9]. Die kodierten Daten werden dabei in Form von Samples repräsentiert. Diese Samples haben eine definierte Anzahl mit einer festgelegten Bit-Breite. Im allgemeinen werden die Samples als Integer-Werte gespeichert, aber auch Floating-Point-Werte werden eingesetzt. Wenn im folgenden Text von PCM-kodiertem Material gesprochen wird, sind stets Samples im Integer-Format gemeint.

2.4.2 MPEG-Audio, mpg123 und LAME

Zu den jeweils von *Moving Pictures Experts Group* (kurz MPEG) festgelegten Videostandards gibt es auch einen entsprechenden Audiostandard: Zu MPEG-1-Video gehört MPEG-1-Audio usw. Die Audiostandards MPEG-1 und MPEG-2 definieren jeweils eine Layer-Struktur (layer, engl. Ebene). Jedem Layer sind verschiedene Kodierungsalgorithmen zugeordnet [17]. Für MPEG-1/2 sind je drei Layer definiert, die mit römischen Zahlen bezeichnet werden: Layer I, II und III. Dabei besitzt der Algorithmus für Layer I die niedrigste, für Layer III die höchste Komplexität. Aktuell sehr oft verwendet wird Layer III, welcher für geringeren Speicherplatzbedarf optimiert wurde. Oft wird Layer III auch als MP3 bezeichnet, was in dieser Arbeit auch der Fall ist.

Um die von MP3 realisierten Kompressionen von ca. 1:10 bei guter Audioqualität zu erreichen, bedarf es einiger besonderer Techniken, von denen hier einige kurz vorgestellt werden. Weitere Informationen zu diesen Techniken finden sich in [18], welche hier auszugsweise wiedergegeben werden:

▷ **Maskierungseffekt** Das menschliche Gehör ist teilweise nicht in der Lage, bestimmte Töne wahrzunehmen, wenn diese von anderen Tönen überlagert werden. Zum Beispiel wird bei Stille nahezu jeder Ton wahrgenommen, doch sobald das Spielen der Musik beginnt, werden bestimmte Töne maskiert und nicht mehr gehört. MP3-Enkoder versuchen diese Eigenschaften des menschlichen Gehörs nachzubilden. Bezeichnet wird dies als ”Psychoakustisches Modell”. MP3 macht sich damit zunutze, daß ungehörte Töne auch nicht kodiert werden müssen. Dies führt zu einer Erhöhung der Kompressionsrate.

▷ **Joint Stereo** Zwei andere Techniken zum Verbessern der Kompressionsrate sind unter Joint Stereo (Stereo-Verbindung, engl.) bekannt: Das menschliche Gehör ist nicht in der Lage, die Quelle von sehr tiefen Tönen zu orten. Daher ist es irrelevant, auf welchen Kanal diese Daten kodiert werden, Es reicht dafür auch ein Kanal.

Weiterhin findet auch ein anderer Trick Verwendung: Wenn sich die linken und rechten Kanäle ähneln, werden die Daten mittig kodiert. Eine Differenz zum mittigen Ton beschreibt dann zusätzlich

sowohl den linken als auch rechten Kanal. Diese Differenzen lassen sich besser komprimieren, da redundante Informationen bereits nicht mehr enthalten sind.

Folgende Implementationen des MP3-Standards werden in dieser Arbeit verwendet: *LAME* für das Enkodieren, sowie *mpg123* zum Dekodieren. LAME bedeutet "LAME Ain't an Mp3 Encoder" [19]. LAME begann als Patch für die ISO-Implementierung des Layer III-Enkoders. Diese wurde schrittweise komplett ersetzt. Mittlerweile steht LAME unter der *LGPL* und wird ausschließlich im Quellcode vertrieben. Der kommandozeilenbasierte MPEG-Audio Player für MPEG-1 und -2 mit dem Namen mpg123 eignet sich zum Dekodieren von MP3 [20]. Die eigentliche Dekodierfunktionalität ist in einer Bibliothek gekapselt, was es ermöglicht, diese getrennt in eigenen Implementationen zu nutzen.

2.4.3 Ogg Vorbis

Ogg Vorbis ist ein komplett freier, nicht patentierter Audio-Codec, welcher unter dem Dach der Xiph-Organisation entwickelt wird [21]. Dabei steht Ogg für den den Namen des Container-Formats (siehe dazu auch Abschnitt 2.5.2). Vorbis ist der Name der Audiokompressionsmethode, die hier Verwendung findet. Prinzipiell ist Ogg Vorbis vergleichbar mit dem aktuell oft verwendeten MP3. Dabei erreicht es jedoch eine höhere akustische Qualität bei gleicher Bitrate, was in [22] belegt wird.

2.4.4 Advanced Audio Coding und Audio Coding 3

Audio Coding 3 (kurz *AC3*) gilt mittlerweile als Synonym für Dolby Digital [12]. Es wurde von der Firma Dolby Laboratories [23] entwickelt. AC3 hat durch die Verwendung zum Kodieren der Audiodaten auf DVDs enorm an Verbreitung gewonnen. Dieser Codec unterstützt das Kodieren von sechs separaten Audiokanälen, die üblicherweise wie folgt verteilt sind: Links, Rechts, Center, Surround Links, Surround Rechts und Bass.

AAC (Advanced Audio Coding) ist als Nachfolger zu AC3 konzipiert [12]. Basierend auf Dolby Digital erweitert es dies um mehrere Verbesserungen beim Kodieren, was zu einer erhöhten Qualität führt. Die akustische Qualität liegt dabei in etwa auf dem Niveau von Ogg Vorbis [22].

2.5 Container

Im folgenden werden kurz die bekanntesten Vertreter (*AV*I und *ASF*) vorgestellt. Das für diese Arbeit verwendete *Ogg Media* wird etwas detaillierter betrachtet. Desweiteren werden interessante Informationen rund um *Matroska*, der aktuellsten Entwicklung in Sachen Container, beschrieben.

2.5.1 AVI und ASF

Als bekanntestes Container-Format gilt das *Audio Video Interleave* (kurz AVI), welches von Microsoft entwickelt wurde [24]. Bild 9 zeigt eine mögliche Anordnung der Daten in diesem Container. Ein großer Nachteil von AVI ist, daß es nicht *streamingtauglich* ist. Daher wurde ebenfalls von Microsoft das *Advanced Streaming Format* (ASF) entwickelt, welches verbesserte Streaming-Eigenschaften aufweist [25]. Allerdings ist ASF nicht patentfrei, was erklärt, daß es keine praxistaugliche Implementation im freien Quellcode gibt, die als Basis für diese Arbeit genutzt werden konnte. An dieser Stelle sei auch auf den Abschnitt 3.7 verwiesen, in dem weitere Eigenschaften von AVI beleuchtet werden.

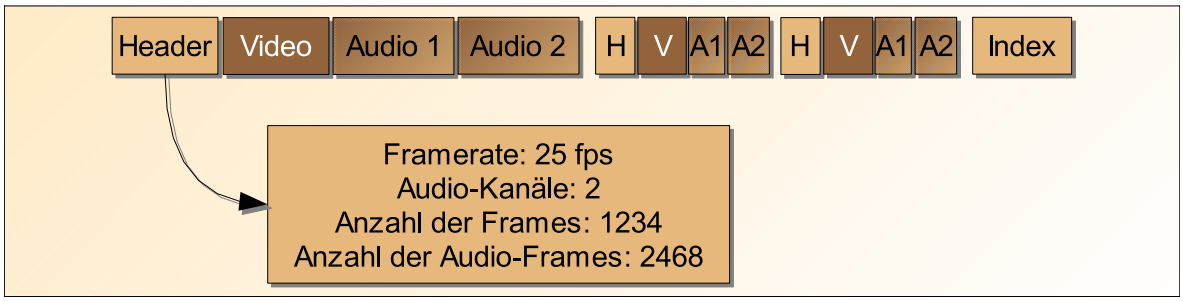

Abbildung 9: AVI: Aufbau des Container-Formats

2.5.2 Ogg Media

Ogg Media (*OGM*) ist der Name eines Container-Formates, welches als patentfreie und streamingfähige Alternative zu AVI entwickelt wird. Der Ogg-Container wird beispielsweise für Ogg Vorbis genutzt. Allerdings ist OGM so flexibel ausgelegt, daß er neben Ogg Vorbis auch MP3, AC3, viele Videoformate (XviD, DivX, u.a.), Kapitelinformationen und auch Untertitel aufnehmen kann [12]. Die Patentfreiheit und die Streamingfähigkeit machen OGM für aktuelle Anwendungen sehr attraktiv. Abbildung 10 zeigt den prinzipiellen Aufbau des OGM-Containers.

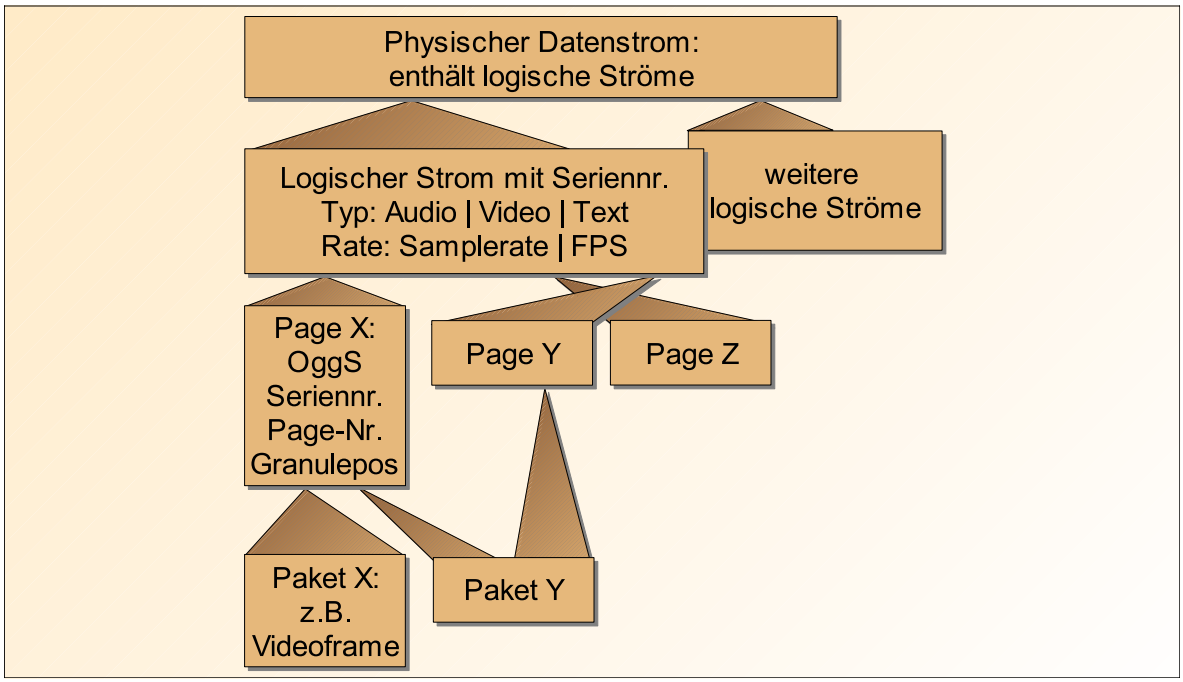

Abbildung 10: OGM: prinzipieller Aufbau des Containers

Ein *physischer Datenstrom* kann mehrere logische Ströme enthalten. Einem *logischen Strom (Stream*, engl.*)* ist immer der Typ der enthaltenen Daten (Audio, Video, Text) sowie die zur Wiedergabe nötigen Informationen (*Samplerate, Framerate*) zugeordnet. Eine Seriennummer ermöglicht die Zuordnung der Seiten (*Pages*) zu einem logischen Strom. Die Seiten-Header dienen u.a. der Synchronisation des Stromdekoders mit dem physischen Strom (Kennung: *OggS*). Zu beachten ist, daß die Pages nicht unbedingt in der Reihenfolge der Wiedergabe im logischen Strom enthalten sein müs-

sen. Dies bedarf einer nachträglichen Sortierung. Die Pages enthalten die eigentlichen Daten (Bild, Ton, Text) in Form von *Paketen*. Dabei ist zu beachten, daß ein Paket in mehreren Pages kodiert sein kann. Weiterhin können in einem Paket mehrere Frames kodiert sein. Ein Paket kann ein Flag *IS_SYNCPOINT* gesetzt haben, das zum Erkennen von I-Frames bei Video dient. Bei Audio ist praktisch jedes Paket eine Syncpoint, da es unabhängig von anderen Paketen dekodiert werden kann.

Besonderes Augenmerk gilt der *Granulepos* - einer Zeitinformation. Aus der Granulepos läßt sich in Abhängigkeit der kodierten Informationen (Format etc.) eine Zeit in Millisekunden errechnen. Im Gegensatz zu anderen Containern handelt es sich hierbei jedoch nicht um die Startzeit der betreffenden Daten, sondern um die Endzeit. Dies ermöglicht beispielsweise beim Aufnehmen Framedrops damit ”zu kodieren”, daß die Dauer des letzten gültigen Frames einfach verlängert wird.

Praktisch ist die Umsetzung dieses Konzeptes durch eine Bibliothek realisiert, die Daten einfach entgegennimmt und sich selbständig synchronisiert. Durch Rückgabewerte von Funktionen werden Informationen wie z.B. ”mehr Daten benötigt” behandelbar. Bei erfolgreichem Auspacken sind die enthaltenen Daten frameweise entnehmbar. Lediglich die Ordnung der Pages muß ggf. korrigiert werden, indem die Pakete anhand der Paketnummer sortiert werden. Weitere Informationen zu Ogg Media befinden sich im Abschnitt 3.7.

2.5.3 Matroska

Matroska ist ein Projekt mit dem ehrgeizigen Ziel, der Standard für Multimedia Container-Formate zu werden [26]. Matroska arbeitet mit *Extensible Binary Meta Language (EBML)*, einer Art binärem *XML*. Ein Vorteil dieser Methode ist, daß auch nach zukünftigen Erweiterungen am Container alte Parser immer noch in der Lage sein sollen, die Dateien zu lesen. Matroska ist als offener Standard konzipiert, bietet aber auch die Möglichkeit, von Firmen in eigene Produkte eingebaut zu werden. Um modernen Anforderungen an ein Container-Format gerecht zu werden, bietet Matroska neben Streamingtauglichkeit u.a. folgendes:

- Erweiterbarkeit für spätere Bedürfnisse durch Nutzung von EBML
- Menüs (wie bei DVDs), sowie Hardwareunterstützung für Standalone-Player
- Kapitelinformationen
- mehrere Untertitel und Audioströme
- Schnelles Springen innerhalb des Containers
- Möglichkeiten zur Fehlerkorrektur

Bei Fertigstellung sollen nahezu alle bedeutenden Video- und Audioformate unterstützt werden: die komplette MPEG-Familie, AC3, PCM u.v.m. Diese und detailliertere Informationen sind in [26] zu finden.

2.6 Vorhandene Software

Dieser Abschnitt gibt eine Übersicht über vorhandene Videorecorder-Software. Weitere Informationen zu Software für die Bearbeitung von Videos sind ebenfalls in der Grundlagenarbeit [1] zu finden.

Das Smart-MPEG-Projekt wird hier erneut vorgestellt, da es essentiell für weitere Betrachtungen innerhalb dieser Diplomarbeit ist.

2.6.1 Smart-MPEG

Smart-MPEG ist ein am Lehrstuhl Betriebssysteme der TU-Dresden entwickelter MPEG-1- und MPEG-2-Dekoder mit Unterstützung von *Quality-of-Service*-Anforderungen (*QoS*-Anforderungen) [27]. Die Strategie zur QoS-Realisierung ist, daß bei steigender Systemlast einzelne Frames ausgelassen werden. Bei zu hoher Systemauslastung werden zuerst B-Frames ausgelassen, danach P-Frames und I-Frames. Die Qualität der Audiowiedergabe bleibt dagegen unverändert. Es werden keine Audiodaten verworfen, so daß der Ton kontinuierlich ausgegeben werden kann [28]. Detailliertere Informationen zur konkreten Realisierung sind in [27] und [28] zu finden. Das Prinzip des Quality-Assuring Scheduling wurde für Smart-MPEG erstmals implementiert und real getestet.

2.6.2 VCR

VCR ist ein oft verwendetes Programm für Linux, welches es ermöglicht von *Video4Linux*-Geräten Videos aufzunehmen. Es bietet Möglichkeiten, verschiedene Codecs zur Datenkompression zu verwenden. Gespeichert werden die Videos in AVI-Containern. Ein Vorteil ist, daß VCR kommandozeilenbasiert gesteuert wird. Dies vereinfacht es, z.B. über Fernzugriff ein Video aufzunehmen [29].

2.6.3 Nvrec

Diese Videorecorder-Software nutzt ebenfalls das Video4Linux-Interface zur Aufnahme. Sie bietet eine Vielzahl von verschiedenen Codecs (DivX, MPEG-1, Nuppelvideo) und Containern [30]. Desweiteren wird Netzwerk-Streaming unterstützt. Nvrec nutzt große Puffer zur Vermeidung von Framedrops bei hoher Systemlast. Durch den modularen Aufbau wird es erleichtert, nvrec in bestehende Anwendungen zu integrieren, sowie eigene Ausgabeformate einzubauen.

2.6.4 Video Disk Recorder

Der Video Disk Recorder, kurz VDR, bezieht das aufzuzeichnende Video direkt von einer Hardwarekarte zum Empfang von *Digital Video Broadcasting (DVB)* [31]. Im Gegensatz zu den genannten Programmen liegen die Daten dabei bereits digital kodiert vor. Dies erspart eine nachträgliche Kompression und die Daten können nahezu unverändert auf die Festplatte gespeichert werden. Die Steuerung erfolgt über ein On-Screen-Display mit Tastatur oder via Infrarot-Fernbedinung. Interessant ist die Time-Shift-Funktion: Diese bietet die Möglichkeit, während der Videoaufzeichnung, das bisher Aufgenommene gleichzeitig und zeitversetzt wiederzugeben. Viele weitere Funktionen machen VDR zu einem der vielseitigsten Videorecorder. Einige davon sind:

- Programmierung mit Hilfe des elektronischen Programmführers *(EPG)*
- Unterstützung von Dolby Digital und mehreren Tonspuren
- Time-Shift und Videoschnittfunktionen
- Netzwerk-Unterstützung ermöglicht die Steuerung auch über Fernzugriff

- Automatisches Ausschalten und Anschalten des PCs
- Plugins ermöglichen es, die Funktionalität zu steigern (z.B. Abspielen von MP3 oder DivX-Videos)
- Mehrere parallele Aufnahmen von einem Gerät
- Gleichzeitige Aufnahme und Wiedergabe auf einem Gerät

2.6.5 Konsequenzen für den Entwurf

Bei einer Hardwarelösung zur Aufnahme von Videos wird die CPU des Aufnahmesystems wesentlich entlastet. Die Verantwortung zur korrekten und fehlerfreien Aufnahme unterliegt damit hauptsächlich der verwendeten Hardware. Bei dem vorgestellten Video Disk Recorder (VDR) unterstützt beispielsweise eine DVB-Karte das Aufnehmen, womit an die restlichen PC-Komponenten nur moderate Leistungsanforderungen gestellt werden müssen. Damit sind auch nach heutigem Standpunkt leistungsschwächere Systeme zur fehlerfreien Aufnahme von Videos geeignet.

Anders bei den hier vorgestellten reinen Softwarelösungen: Diese nutzen ausschließlich Puffermechanismen zur Vermeidung von Framedrops bei zu hoher Systemlast. Die CPU übernimmt allein die Kodierung der Video- und Audiodaten. Eine Folge ist, daß Framedrops bei unzureichend dimensionierten Puffern und hoher Systemauslastung auftreten können. Andererseits ist die Puffergröße durch die Größe des Hauptspeichers begrenzt. Ein weiteres Problem dieser Lösungen ist, daß während der Aufnahme parallel laufende Software die Möglichkeit hat, die Aufzeichnung des Videos zu stören. Daher ist es nötig, für reine Softwarelösungen andere Möglichkeiten als die Dimensionierung der Puffergröße zur Vermeidung von Framedrops zu entwickeln. Mögliche Lösungen dieser Aufgaben werden im nächsten Kapitel dargestellt.

3 Entwurf

3.1 Probleme bei der Aufnahme und Wiedergabe von Videos

Die menschlichen Sinnesorgane stellen bestimmte Bedingungen an die Wiedergabe und somit auch an die Aufnahme von Videos. Die hier kurz erwähnten Probleme resultieren aus den im Abschnitt 2.1 vorgestellten Eigenschaften der menschlichen Wahrnehmung:

1. Ein Versatz von mehr als 80ms zwischen Bild und Ton ist nicht mehr als lippensynchron zu bezeichnen. Es ist daher nötig, die Synchronisation von Audio und Video bei der Wiedergabe herzustellen und anschließend zu erhalten. Bei der Aufnahme des Videos muß dafür gesorgt werden, daß die nötigen Zeitinformationen bei der Präsentation rekonstruiert werden können.

2. Schwankungen der Bearbeitungszeit gefährden die Synchronität von Audio und Video, da beispielsweise pro Bild nur eine von der Framerate abhängige maximale Verarbeitungsdauer zulässig ist.

3. Der Ton ist wichtiger als die Bilder. Audio sollte daher bevorzugt verarbeitet werden.

4. Framedrops sowie das Auslassen von Audiodaten sollten vermieden werden, weil sie wahrnehmbar und damit störend wirken.

Diese Probleme gilt es beim Entwurf und der Implementierung zu lösen bzw. zu vermeiden. Die folgenden Abschnitte nehmen bezug auf diese Aufgabe und skizzieren mögliche Lösungen.

3.2 Umsetzung des Quality-Assuring Scheduling

Die von den menschlichen Sinnesorganen festgelegten Anforderungen legen klar fest, daß es sich bei der Videowiedergabe und -aufnahme um Echtzeitanforderungen handelt. Durch die Verwendung des *Dresden Real-Time OPerating System* als Grundlage des zu erstellenden Videosystems sollen diese Anforderungen erfüllt werden (vgl. Abschnitt 2.1). DROPS nutzt als Basis das Quality-Assuring Scheduling. Um ein Modell für die Aufnahme und Wiedergabe von Videos unter DROPS zu erstellen, ist es daher nötig, die Implementierung des QAS näher zu betrachten.

Gegenwärtig wird an der Implementierung des Schedulers gearbeitet. Wie bereits im Abschnitt 2.2.2 vorgestellt, werden die Tasks in mandatorische und optionale Teile zerlegt. Der Fiasco-Microkernel bietet 256 Prioritätsklassen, wobei 255 die höchste und Null die niedrigste Priorität ist. Alle Threads werden entsprechend ihrer Priorität geschedult. Das QAS-Scheduling wird daher auf diese 256 Klassen abgebildet. Die mandatorischen Teile erhalten eine höhere Priorität als die optionalen Teile. Daher werden zuerst alle mandatorischen Teile ausgeführt. Danach werden die optionalen Teile nach Höhe ihrer Priorität geschedult. Die Schnittstellen-Spezifikation für die Kommunikation mit Scheduler sind gegenwärtig bereits geplant [32]. Dem Scheduler wird die Aufteilung des Jobs in die mandatorischen und optionalen Teile in Form einer verketteten Liste bekannt gemacht. Dabei werden die Dauer der Periode des Tasks und die Priorität sowie die einzuplanenden Ausführungszeiten der einzelnen Teile mitgeliefert. Abbildung 11 zeigt dies beispielhaft. Später soll ein Admission-Server anhand der gewünschten Zeiten die Priorität bestimmen und überprüfen, ob die Anwendung für einen ausführbaren Ablaufplan zugelassen werden kann. Gegenwärtig ist der Admission-Server noch nicht

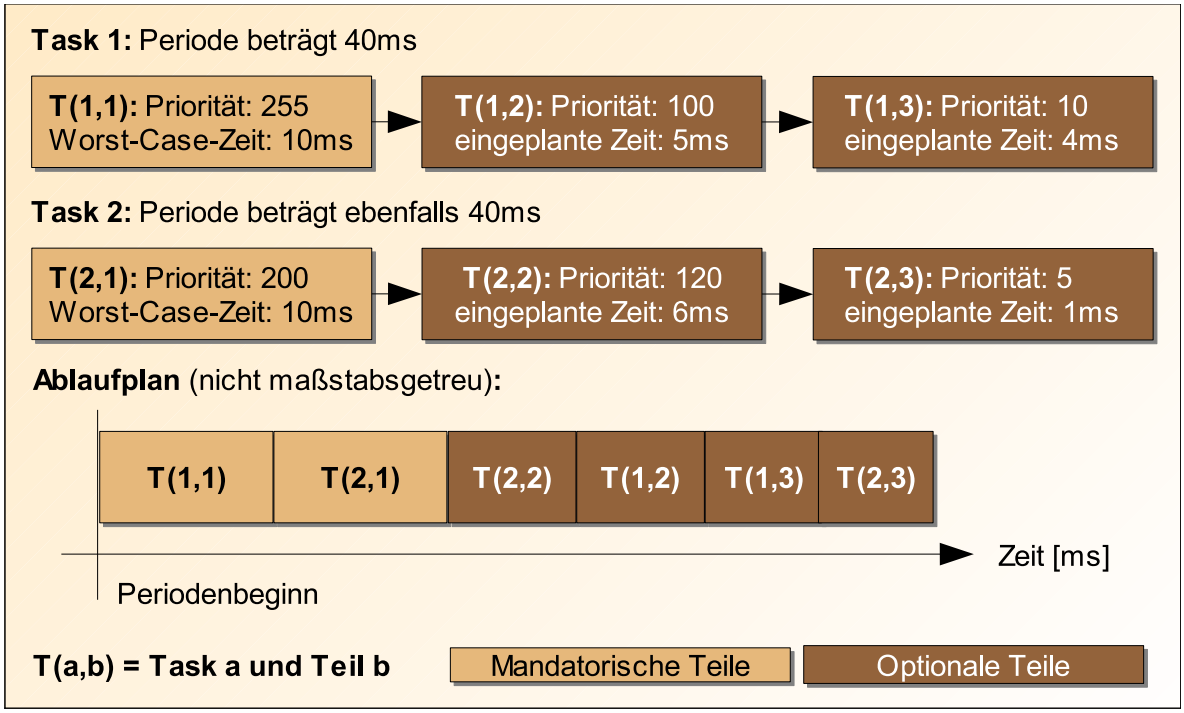

Abbildung 11: QAS: Beispiel einer Taskbeschreibung für den Scheduler

implementiert. Daher müssen Anwendungen direkt mit dem Scheduler kommunizieren. Dazu wird das Scheduler-Interface die vier folgenden Funktionen anbieten. Eine beispielhafte Anwendung dieser Aufrufe wird im Abschnitt 3.3 dargestellt.

▷ **begin_period(abs_time)** Mit dieser Funktion wird dem Scheduler signalisiert, daß die periodische Abarbeitung zum absoluten Zeitpunkt *abs_time* gestartet werden soll.

▷ **next_reservation()** Ein Aufruf dieser Funktion bedeutet, daß von einem fertig bearbeiteten Teil zum nächsten gewechselt werden soll, beispielsweise vom mandatorischen Teil zum ersten optionalen oder vom ersten zum zweiten optionalen Teil usw.

▷ **next_period()** next_period() dient der Benachrichtigung, daß alle Teile der aktuellen Periode fertig ausgeführt sind. Anschließend wird auf den Beginn der nächsten Periode gewartet.

▷ **end_period()** Dieser Aufruf dient der Beendigung der periodischen Abarbeitung.

Die Anwendung sollte wissen, ob die Abarbeitung der jeweiligen Teile fertig war bzw. ob diese unterbrochen wurden, um darauf reagieren zu können. Dazu ist es angedacht, daß *next_period()* und *next_reservation()* Informationen zurückliefern, wieviel Zeit in der aktuellen Periode bzw. für den jeweiligen Teil noch zur Verfügung gestanden hätte, ohne die eingeplante Zeit zu überschreiten. Um festzustellen, daß ein Teil seine Zeit überzogen hat, ist angedacht, daß die Anwendung einen ”*preemption ipc*” zugestellt bekommt. Basierend auf diesen Informationen werden im folgenden Modelle zur Videoaufnahme und -wiedergabe entwickelt.

3.3 Strategie hinter Smart-MPEG

Das bereits im Abschnitt 2.6 ”Vorhandene Software” vorgestellte Smart-MPEG nutzt folgendes Konzept zur Realisierung von Quality-of-Service-Unterstützung beim Dekodieren von MPEG-1- und MPEG-2-Video: Bei steigender Systemlast werden einzelne Frames ausgelassen. Zuerst werden B-Frames ausgelassen, danach P-Frames und anschließend I-Frames. Dieser Ansatz ist für die Verwendung von Quality-Assuring Scheduling sehr gut geeignet. Bei MPEG-1 und -2 ist die weitestgehend anzutreffende *Group of Pictures (GOP)*, also eine festgelegte Reihenfolge von I-, P- und B-Frames, IBBPBBPBBPBB. Innerhalb einer Periode wird dabei die gesamte GOP bearbeitet. Der mandatorische Teil besteht nun daraus, alle I-Frames der GOP zu dekodieren. Der erste optionale Teil dekodiert P-Frames, der zweite die B-Frames. Dazu wird der zu dekodierende Datenstrom in drei Ströme entsprechend der Frametypen zerlegt. Der folgende Algorithmus zeigt die prinzipielle Umsetzung in Pseudo-Code mit Hilfe der in 3.2 eingeführten Scheduler-Schnittstelle:

Algorithmus QAS: Beispiel für die Nutzung der Scheduler-Schnittstellen

```
begin_period(Startzeit);
while(running)
{
   //Beginne Bearbeitung einer Group of Pictures
   //mandatorischer Teil
   decode(I-Frame);
   next_reservation();
   //erster optionaler Teil
   decode(3 x P-Frames) //erhöht die visuelle Qualität
   next_reservation();
   //zweiter optionaler Teil
   decode(8 x B-Frames) //erhöht die visuelle Qualität noch weiter
   //alle Teile dieser Periode sind abgearbeitet
   next_period();
}
//die periodische Abarbeitung ist beendet
end_period();
```

Damit gelingt es, Quality-of-Service Anforderungen beim Dekodieren von MPEG-1 bzw. MPEG-2-Videos umzusetzen. Außerdem wird auf das Zusammenspiel von Smart-MPEG mit dem Quality-Assuring Scheduling in [7] detaillierter eingangen. Dieses Konzept funktioniert bei MPEG-4 nicht, da dort keine festgelegte Group of Pictures mehr existiert. So ist nicht festgelegt, daß nach einem I-Frame eine definierte Anzahl von P- oder B-Frames folgt. Ein I-Frame kann durchaus nur alle paar hundert Frames auftauchen. Dies dient der Senkung der Bitrate, da I-Frames im Verhältnis zu P- und B-Frames einen großen Speicherbedarf haben. Wenn eine Periode nun das Zeitintervall von einem I-Frame bis zum nächsten ist, wie es einer GOP entspricht, dann wäre die Periodendauer nicht mehr konstant. Daher müßte als Periode für das Scheduling der Mindestabstand zwischen zwei I-Frames eingeplant werden. Dieser könnte aber nur durch Vorablesen des gesamten Videos bestimmt werden. Wird andererseits eine bestimmte Anzahl von Frames als Periode eingeführt, dann gäbe es unter Umständen kein I-Frame und somit keinen mandatorischen Teil.

Desweiteren ist zu bezweifeln, daß die Strategie von Smart-MPEG auch bei H.264/AVC sinnvoll einsetzbar ist: Wie bereits im Abschnitt 2.3.4 vorgestellt, verwendet H.264/AVC Slices. Dabei ist es möglich, daß ein Frame beispielsweise in zwei P-Slices und ein I-Slice zerlegt und enkodiert wurde.

Um das komplette Frame vollständig dekodieren zu können, müssen sowohl das I-Slice als auch die beiden P-Slices dekodiert werden. Danach wird das "Original-Bild" rekonstruiert, indem die dekodierten Slice-Bereiche zusammengesetzt werden. Die beiden P-Slices brauchen dazu Referenzslices, z.B. vorherige I- oder P-Slices. Das I-Slice ist intrakodiert. Fehlt ein Slice, z.B. durch einen Übertragungsfehler, dann kann der Bildteil, der durch dieses Slice abgedeckt war, nur interpoliert werden, was sichtbar ist.

Eine Folge von Frames bestehend aus jeweils verschiedenen Slice-Typen wird bei Nutzung des Smart-MPEG-Ansatzes zum Problem. Es ist denkbar, daß im ersten Frame ein P-Slice, aber nicht das I-Slice verworfen wird. Im zweiten Bild wird beispielsweise nur ein B-Slice verworfen usw. Dies wirkt sich wie eine Reihe von Übertragungsfehlern aus: Alle Frames, bei denen mindestens ein Teil verworfen wurde, können nicht mehr vollständig dekodiert werden. Somit kann es passieren, daß mehrere Frames hintereinander unvollständig dekodiert werden. Zusätzlich sei kurz bemerkt, daß auch Bilder mit B-Slices als Referenzframes dienen können, was zusätzlich problematisch sein dürfte.

Die genannten Probleme ließen sich lösen, wenn beim Enkodieren definierte GOPs bzw. nur definierte Slice-Aufteilungen verwendet würden. Damit verliert aber die Strategie hinter Smart-MPEG ihre allgemeine Verwendbarkeit. Der Smart-MPEG-Ansatz eignet sich daher nur eingeschränkt zur Nutzung mit aktuellen und zukünftigen Codecs beim Dekodieren. Für das Enkodieren ist die Verwendung dieser Strategie ebenfalls nicht sinnvoll, da alle Frames unkodiert vorliegen. Damit gibt es keine systematische Entscheidungsmöglichkeit mehr, welche Frames zu verwerfen sind. Ebenfalls ist keine sinnvolle Aufteilung in mandatorische und optionale Teile möglich. Somit wird sowohl für das Enkodieren als auch für das Dekodieren ein neues Konzept benötigt.

3.4 Konzept der Qualitätsstufen-Anpassung

Wie bereits mehrfach erwähnt, liegt dieser Arbeit der Große Beleg ”Portierung eines Video-Codecs auf DROPS” [1] zugrunde. Das dort entwickelte Konzept der Qualitätsstufen-Anpassung (QAP) findet auch in dieser Arbeit Verwendung. Es wird an dieser Stelle nochmals kurz vorgestellt sowie durch aktuelle Meßwerte bestätigt.

Das Konzept der Qualitätsstufen-Anpassung besagt, daß eine dynamische Anpassung der Kodierzeiten abhängig von der Systemlast erfolgt. Diese spiegelt sich in der akustischen oder visuellen Qualität der verarbeiteten Daten wieder. In der hier auszugsweise wiedergegebenen Grundlagenarbeit [1] ist es angedacht, die benötigte Kodierzeit auf ein Minimum zu reduzieren, um Framedrops zu vermeiden bzw. zur Steigerung der Qualität auch längere Kodierzeiten zuzulassen. Die Abhängigkeit der benötigten Rechenzeit von der visuellen bzw. akustischen Qualität wird als *Rechenzeit-Qualitäts-Funktion (RQF)* bezeichnet. Das Definieren von Qualitätsstufen, wobei höhere Qualitätsstufen höherer akustischer bzw. visueller Qualität entsprechen, sowie das Vorhandensein einer monoton wachsenden RQF sind die Grundlagen dieses Konzeptes. Dabei ist zu beachten, daß diese Funktion vom verwendeten Codec abhängig ist, was als *spezifische Rechenzeit-Qualitäts-Funktion (sRQF) eines Codecs* bezeichnet wird. Als ein in [1] genanntes Problem gilt es, für die verwendeten Codecs die spezifische Rechenzeit-Qualitäts-Funktion zu ermitteln. Die benötigten Rechenzeiten unterscheiden sich für das Dekodieren von denen beim Enkodieren.

▷ **Enkodieren** Die Abbildung 12 zeigt, daß sowohl für den bereits portierten XviD-Codec, als auch für LAME eine spezifische Rechenzeit-Qualitäts-Funktion für das Enkodieren zu finden ist. Dazu wurden für XviD und LAME Qualitätsstufen bestimmt. So werden beispielsweise bei XviD für die unterschiedlichen Qualitäten unterschiedlich genau arbeitende Algorithmen für die Bewegungserkennung verwendet[1]. Bei LAME wird z.B. bei niedrigerer Qualitätsstufe ein ungenaueres und damit schnelleres psychoakustisches Modell benutzt [33]. Die genauen Spezifikationen der definierten Qualitätsstufen und Details zu den vorgenommen Messungen sind im Anhang B bzw. F nachzulesen. Wie aus den genannten Diagrammen ersichtlich wird, ist bei XviD eine durchschnittliche Rechenzeitersparnis von ca. 60% ausgehend von der höchsten Qualitätsstufe möglich. Die vorgenommenen Messungen bestätigen die Angaben in [1], wobei diesmal andere Testvideos genutzt wurden als bisher. Dies untermauert die Gültigkeit dieser sRQF für verschiedene Videos. Bei LAME ist sogar eine Verringerung der durchschnittlichen Verarbeitungsdauer um 65% möglich. Die konkreten Werte dieser Funktionen wurden bereits mit den auf DROPS portierten Codecs ermittelt.

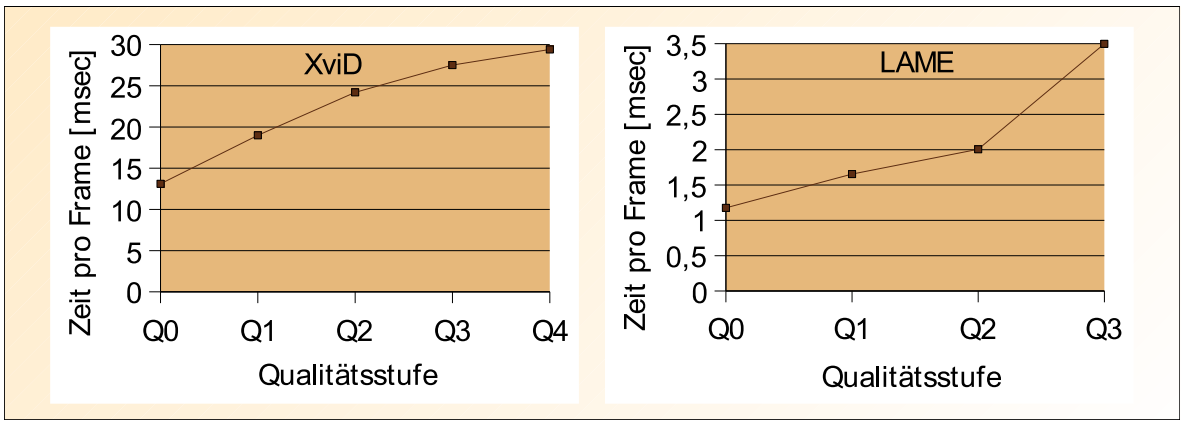

Abbildung 12: XviD und LAME: spezifische Rechenzeit-Qualitäts-Funktion beim Enkodieren

▷ **Dekodieren** Für das Dekodieren sind die Qualitätsstufen und die damit verbundene spezifische Rechenzeit-Qualitäts-Funktion schwierig zu bestimmen. Dies ist damit zu begründen, daß das Material bereits kodiert vorliegt. Somit sind die Dekoder gezwungen, die im kodierten Material festgelegten Operationen auszuführen, wobei kaum Platz zum Variieren der Parameter bleibt (vgl. Abschnitt 2.3.2). Zumindest für das Dekodieren mit XviD ließ sich ein anderes Verfahren zur Qualitätssteigerung finden: *Postprocessing*. Dabei wird beispielsweise versucht, die durch die Kodierung auf Blockbasis entstandenen Artefakte zu glätten - *Deblocking* genannt. Dabei wird nochmals zwischen vertikalem und horizontalem Deblocking unterschieden. Auch gibt es die Möglichkeit, die visuelle Qualität durch *Deringing* zu erhöhen. Deringing versucht Ringartefakte, die bei der Kodierung mit niedriger Bitrate an harten Kontrastübergängen entstehen, zu kaschieren. Zu erwähnen ist aber, daß sowohl Deblocking als auch Deringing bei übermäßiger Anwendung das Bild weichzeichnen und daher zum Verlust von Bilddetails führen können. Im übrigen wird bei H.264/AVC bereits im Rahmen des Standards ein Postprocessing vorgeschrieben. Die Abbildung 13 zeigt den Unterschied zwischen Frames ohne und mit aktivem Deblocking und Deringing. Der vergrößerte Bereich zeigt die Blockartefakte (beide Holzbalken) besonders gut, aber auch Ringartefakte sind am Übergang von den Steinen zum Himmel zu entdecken.

Abbildung 13: Postprocessing: ohne und mit Deblocking und Deringing

Damit gelingt es auch für das Dekodieren mit XviD, die spezifische Rechenzeit-Qualitäts-Funktion zu erstellen, welche in Abbildung 14 dargestellt ist. Dies ermöglicht eine Verringerung der durchschnittlichen Verarbeitungsdauer um ca. 28% ausgehend von der höchsten Qualitätsstufe mit aktivem Deblocking und Deringing. Die Parameter der Qualitätsstufen sind ebenfalls im Anhang B zu finden.

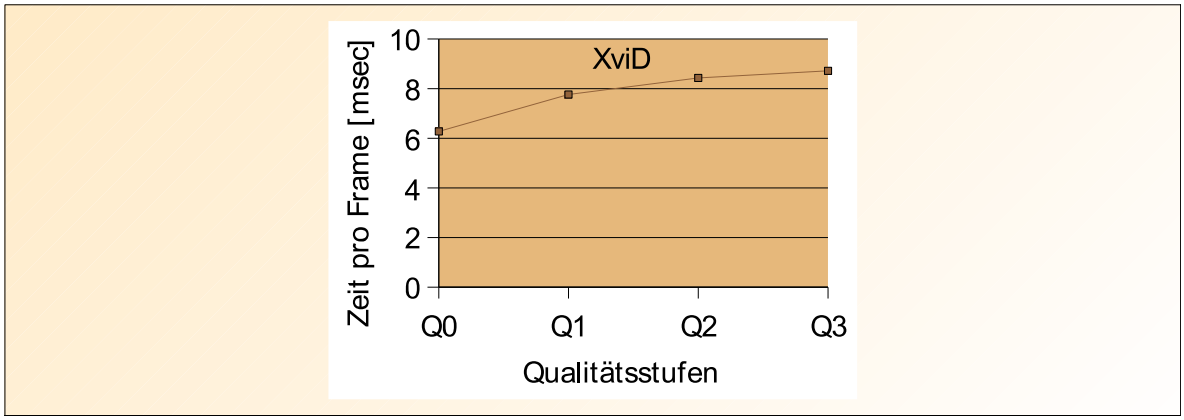

Abbildung 14: XviD: spezifische Rechenzeit-Qualitäts-Funktion beim Dekodieren

Beim Dekodieren von MP3 via mpg123 war es nicht sinnvoll möglich, verschiedene Qualitätsstufen zu definieren. Daher besteht die sRQF aus lediglich einer Qualitätsstufe mit einer zugeordneten Rechenzeit. Zu den Anforderungen an eine Rechenzeit-Qualitäts-Funktion ist dies nicht widersprüchlich. Allerdings beraubt man sich der Möglichkeit, die Rechenzeit abhängig von der Systemlast sinnvoll zu variieren, was dem Konzept der Qualitätsstufen-Anpassung nicht dienlich ist. Daher ist dieses Konzept für das Dekodieren zumindest für den verwendeten mpg123-Codec nur begrenzt geeignet.

3.5 Modelle für Videorecorder und -player

3.5.1 Allgemeines Modell mit Beschreibung der Einzelkomponenten

Das allgemeine Modell besteht aus verschiedenen Funktionsblöcken. Diese im folgenden vorgestellten Komponenten dienen sowohl zur Erklärung der Wiedergabe als auch als Basis für das Modell der

Aufnahme. Dabei entspricht es weitgehend der im Beleg [1] entwickelten Infrastruktur.

Alle Komponenten sind nach Möglichkeit mit dem DROPS Streaming Interface verknüpft, was die Wiederverwendbarkeit steigert. Dazu ist es nötig, daß der Datenstrom in Pakete zerlegt wird: Bei Video entspricht ein Paket beispielsweise einem Frame, bei Audio z.B. einer gewissen Anzahl von Samples. Die Abbildung 15 illustriert dieses Modell. Die Verarbeitung von Audio und Video erfolgt weitgehend getrennt. Es folgen einige Hinweise zum verwendeten Konzept sowie eine Beschreibung der einzelnen Komponenten. Details zur geplanten Intergration des Modells in das DROPS Scheduling Framework werden im Abschnitt 3.6 wiedergegeben.

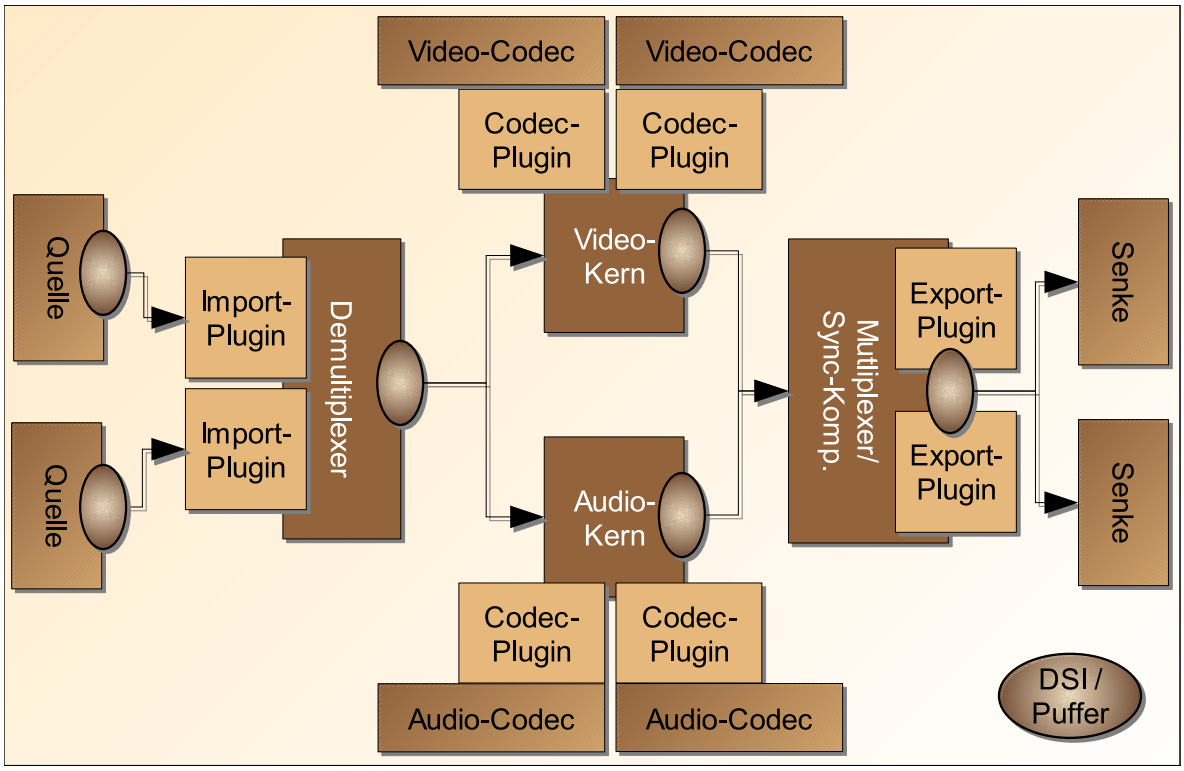

Abbildung 15: Allgemeines Modell

▷ **Konzept** Das Grundmodell entspricht dem im Abschnitt 3.4 vorgestellten Konzept der Qualitätsstufen-Anpassung. Dieses sieht vor, daß in Abhängigkeit der dem System zur Verfügung gestellten Rechenzeit die akustische oder visuelle Qualität gesenkt bzw. gesteigert wird, um die benötigte Rechenzeit zu sparen bzw. zu verlängern.

▷ **Plugins** *Plugins* sind Softwarekomponenten zur Ergänzung der Funktionalität. Diese sind speziell auf bestimmte Anwendungsgebiete ausgerichtet, z.B. auf verschiedene Container-Formate. Die Nutzung von Plugins erfolgt durch übergeordnete Komponenten. Dabei ist der Zugriff standardisiert und bietet damit eine Abstraktion der spezifischen Anwendungsgebiete einzelner Plugins. Durch einen Austausch kann daher die Funktionalität der übergeordneten Softwarekomponente verändert werden. Genauere Beschreibung der im Modell verwendeten Plugins folgen in den Beschreibungen der entsprechenden Einzelkomponenten.

▷ **Quellen** Als Quellen für den Videorecorder können für die Aufnahme eines Videos beispielsweise eine Webcam oder TV-Karte zusammen mit einer Soundkarte dienen. Für die Wiedergabe ist die Nutzung eines Dateisystems oder eines Netzwerkes denkbar.

▷ **Demultiplexer** Der Demultiplexer holt einzelne Frames aus einem Container und gibt die Daten an den Kern paketweise via DSI weiter. Dazu nutzt er in Abhängigkeit der verschiedenen Quellen und Container-Formate **Import-Plugins**, also speziell auf diese zugeschnittene und austauschbare Software. Durch das Wechseln der Plugins wird es möglich, z.B. verschiedene Container-Formate wie AVI oder OGM an das Gesamtsystem anzubinden. Beim Import von RAW-Daten direkt von der Webcam oder Soundkarte entfällt die Nutzung des Demultiplexers, da Daten ohne Container vorliegen und direkt in den Kern mit Hilfe dessen Import-Plugins eingelesen werden können.

▷ **Kern** Die Kerne bekommen die Audio- bzw. Videodaten von ihren jeweiligen Quellen. Auch hier entscheidet die Nutzung verschiedener **Import-Plugins** über die Auswahl der Quelle (nicht dargestellt). Für den Fall, daß die Daten noch zu dekodieren bzw. zu enkodieren sind, werden die Daten an **Codec-Plugins** weitergegeben. Dieser Plugin-Typ ist daher für die Dekodierung (Wiedergabe) bzw. Enkodierung (Aufnahme) mit Hilfe entsprechender Codecs verantwortlich. Dafür bieten sie eine einheitliche Schnittstelle zum Zugriff auf diese an. Desweiteren ist der Kern verantwortlich für das Postprocessing und affine Bildtransformationen (z.B. Größenänderungen) bzw. Änderungen des Audiomaterials (z.B. Lautstärkeanpassungen oder *Downmixing*). Die fertig verarbeiteten Daten werden anschließend an den Multiplexer bzw. die Synchronisationskomponente übergeben.

▷ **Multiplexer** Der Multiplexer schreibt die kodierten Daten in einen Container. Dies geschieht nur bei der Aufnahme, nicht jedoch bei der Wiedergabe. Hierbei bestehen Abhängigkeiten zum verwendeten Container-Format. Daher ist der Multiplexer darauf abzustimmen, was durch die Nutzung von spezifischen **Export-Plugins** ermöglicht wird.

▷ **Synchronisationskomponente** Diese Komponente dient der Synchronisation von Bild und Ton bei der Wiedergabe. Desweiteren ist diese Komponente für die Weitergabe von Audio- und/oder Videodaten an die Senke verantwortlich. Dabei können wieder senkenspezifische **Export-Plugins** zum Einsatz kommen. Für die Aufnahme von Videos ist diese Komponente nicht erforderlich.

▷ **Senken** Als Senken für die Wiedergabe können beispielsweise die Grafikkarte bzw. die Soundkarte dienen. Bei der Aufnahme sollte nach Möglichkeit auf ein Dateisystem zum Speichern des Videos zurückgegriffen werden, allerdings ist auch ein Streaming via Netzwerk o.ä. nicht ausgeschlossen.

▷ **Puffer** Zur Vermeidung von Aussetzern bei der Aufnahme bzw. Wiedergabe und dem Ausgleich von Schwankungen der Verarbeitungsdauer innerhalb von Einzelkomponenten dienen Puffer. Diese sind durch die Verwendung des *DROPS Streaming Interface* bereits automatisch im System enthalten. Durch eine Erweiterung an DSI kann auch der Pufferfüllstand ausgelesen werden und als Maß für die Systemlast dienen.

▷ **Controller (nicht dargestellt)** Die Steuerung wird über den Controller erfolgen, welcher der Interaktion mit dem Nutzer sowie der Kontrolle der Qualitätsstufen-Anpassung dient. Dazu werden alle gerade aufgelisteten Komponenten dem Controller entsprechende Schnittstellen und Informationen über deren aktuellen Zustand anbieten. Beispielsweise werden Codec-Plugins Informationen über ihre spezifische Rechenzeit-Qualitäts-Funktionen (sRQF) bereitstellen. Durch Austausch des Controllers werden spätere Änderungen der Regelung für die Qualitätsstufen-Anpassung ermöglicht.

3.5.2 Modell für die Aufnahme

Das Modell für die Aufnahme von Videos wird am Beispiel von Webcam und Soundkarte erläutert. Dies entspricht der Aufgabenstellung. Daten von diesen beiden Quellen sollen nach dem Enkodieren z.B. in einem OGM-Container auf ein Dateisystem geschrieben werden. Ein Demultiplexer ist unnötig, da die Audio- und Videodaten bereits ohne eine Einbettung in einen Container vorliegen. Die Abbildung 16 stellt dieses Szenario mit Hilfe der im Abschnitt 3.5.1 eingeführten Komponenten schematisch dar.

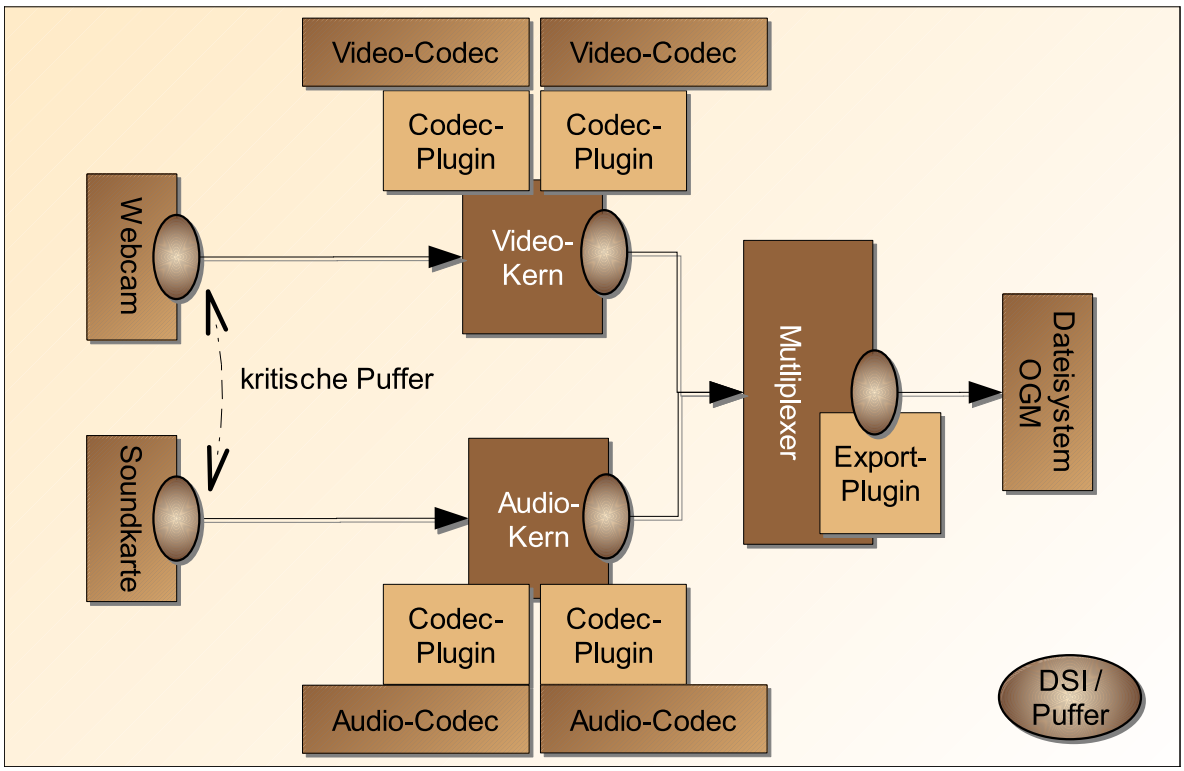

Abbildung 16: Modell für Aufnahme

Besondere Beachtung gilt den beiden Puffern zwischen den Quellen und dem jeweiligen Audio- bzw. Videokern. Beim Überlauf dieser Puffer kommt es zu Framedrops oder zum Auslassen von Audio-Samples. Allerdings sind auch die anderen Puffer insofern kritisch, als daß die Daten nicht schnell genug von der Senke, z.B. einem Dateisystem, in Empfang genommen werden können. Auf dieses Problem wird im Abschnitt 3.7 genauer eingangen. An dieser Stelle sei angenommen, daß diese Puffer ausreichend dimensioniert sind. Um Framedrops zu verhindern, kommt das Konzept der Qualitätsstufen-Anpassung zum Einsatz: Realisiert wird dies durch Anpassung der Verarbeitungszeit

innerhalb des Video- bzw. Audiokernes für die Enkodierung in Abhängigkeit der gegenwärtigen Systemlast. Für die Ermittlung der Systemlast gibt es im wesentlichen zwei Möglichkeiten: Die erste Methode nutzt Informationen über den Füllstand der beiden kritischen Puffer. Ein fast leerer Puffer zeigt an, daß die zur Verfügung stehende Rechenzeit ausreicht. Ein voller Puffer signalisiert Systemüberlast. Eine vom Pufferfüllstand abgeleitete Größe, z.B. die Geschwindigkeit der Pufferänderung, läßt sich ebenfalls verwenden.

Für die zweite Möglichkeit zur Ermittlung der Systemlast wird die geplante Implementierung des Quality-Assuring Schedulings herangezogen. Details zu einer möglichen Integration des Modells in das DROPS Scheduling Framework werden im Abschnitt 3.6 beschrieben. Es sei bereits vorweggenommen, daß die beiden Kerne die Audiodaten und Videoframes periodisch bearbeiten. Für beide ist pro Periode eine gewisse Rechenzeit reserviert. Beim Überziehen dieser Rechenzeit, wird ein "*preemption ipc*" gesendet. Dies deutet darauf hin, daß die eingeplante Zeit nicht ausreicht und die Qualität gesenkt werden muß. Bei zu geringer Auslastung wird die Rechenzeit erhöht. Die Kerne bemerken dies, indem sie beim Aufruf *next_period()* bzw. *next_reservation()* mitgeteilt bekommen, wieviel von der eingeplanten Rechenzeit innerhalb dieser Periode noch übrig ist. Mit diesen beiden Informationen, welche der Systemlast entsprechen, kann man abschätzen, ob die Qualität zu steigern oder zu senken ist, um innerhalb einer Periode durchschnittlich ein Frame fertig zu kodieren.

Die Intelligenz zur Kontrolle der Qualitätsstufen-Anpassung unterliegt der Verantwortung des Controllers. Dazu wird er von den beiden Kernen mit den nötigen Informationen zur Ermittlung der Systemlast versorgt. Im Controller eingebaute oder vom Benutzer festgelegte Steuerungen ändern nach Bedarf die benötigte Rechenzeit anhand der spezifischen Rechenzeit-Qualitäts-Funktion der verwendeten Codecs. Es ist denkbar, daß es der Benutzer beispielsweise vorzieht, Frames auszulassen anstatt die Qualität zu senken.

Ebenso kontrolliert der Controller die Prioritätsverwaltung zwischen Bild und Ton, da gefordert wird, daß Audio bevorzugt gegenüber Video verarbeitet werden soll. Es sollte allerdings vermieden werden, daß der Ton mit der höchsten Qualität kodiert wird, während das Bild gar nicht kodiert wird. An dieser Stelle muß ein Kompromiß zwischen Bild- und Tonqualität, also zwischen Audio- und Videopriorität gefunden werden. Denkbar ist dazu ein Modell, bei dem der Nutzer die maximale Differenz zwischen Video- und Audioqualitätsstufe wählt. Zur Verallgemeinerung sollte diese Einstellung prozentual vorgenommen werden.

Bei Framedrops oder Verlust von Audiodaten liegt die Verantwortung beim Multiplexer, mit geeigneten Maßnahmen zu reagieren. Um überhaupt feststellen zu können, daß Aussetzer aufgetreten sind, werden alle Datenpakete mit einem Zeitstempel und ihrer Gültigkeitsdauer von ihrer Quelle markiert. Bei Video z.B. ist die Gültigkeitsdauer gleich dem Kehrwert der Framerate. Evtl. erfolgte Datenverluste können so anhand eines Vergleichs des letzten Zeitstempels und dessen Dauer mit dem aktuell erhaltenen Zeitstempel festgestellt werden. Desweiteren sind im Zeitstempel alle zur Wiedergabe nötigen Informationen über den zeitlichen Bezug der Audio- und Videodaten untereinander, welche beim Multiplexer zusammenlaufen, enthalten.

Zusammenfassend sind bezugnehmend auf die im Abschnitt 3.1 beschriebenen Probleme in Verbindung mit der menschlichen Wahrnehmung folgende Lösungen angedacht:

1. Die Synchronität wird durch den Zeitstempel gewährleistet und unterliegt der Verantwortung des Multiplexers.

2. Zu welchem Zeitpunkt die Daten geschrieben werden, ist irrelevant. Wichtig ist nur, daß alles gespeichert wird. Die Schwankung der Verarbeitungsdauer ist daher unproblematisch, zumindest solange die Kerne noch arbeiten und Daten schnell genug geliefert sowie entnommen werden können (vgl. 2.2.2).

3. Die Audioverarbeitung wird bevorzugt behandelt. Dabei sorgt der Controller für ein sinnvolles Verhältnis von Audio- und Videoqualität durch Auswählen entsprechender Qualitätsstufen.

4. Alle Datenpakete werden mit einem Zeitstempel und ihrer Gültigkeitsdauer von der Quelle markiert. Dies dient der Feststellung über evtl. erfolgte Datenverluste und der für die Wiedergabe nötigen Informationen über den zeitlichen Bezug der Daten untereinander.

3.5.3 Modell für die Wiedergabe

Das Modell für die Wiedergabe unterscheidet sich dahingehend von der Aufnahme, daß andere Quellen und Senken genutzt werden. Angestrebt ist beispielsweise die Nutzung eines Dateisystems als Quelle und von Grafikkarte und einer Soundkarte als Senken für Video- bzw. Audiodaten. Üblicherweise liegen aufgenommene Videos in Form von in einen Container gepackten Bild- und Tonmaterial vor. Ein Demultiplexer sorgt für die Extraktion der Daten aus diesem Container und gibt diese dann zur Dekodierung an die entsprechenden Kerne weiter. Ein wichtiger Unterschied gegenüber dem Aufnahmemodell ist außerdem, daß anstelle des Multiplexers eine Synchronisationskomponente dafür sorgt, daß Bild und zugehöriger Ton synchron ausgegeben werden. Zu der im Rahmen dieser Arbeit entwickelten Synchronisationskomponente werden im Abschnitt 4.5 weitere Details vorgestellt. Eine schematische Darstellung des Modells ist in Abbildung 17 zu finden. Auch bei diesem Modell kommt das in 3.4 vorgestellte Konzept der Qualitätsstufen-Anpassung zum Einsatz.

Die beiden kritischen Puffer befinden sich zwischen den Kernen und der Synchronisationskomponente. Wiederum existiert hier die bereits beim Aufnahmemodell beschriebene Möglichkeit, die Regelung über das Scheduler-Interface oder über die Pufferfüllstände zu realisieren: Wenn diese leer sind, dann gibt es bei der Ausgabe der Daten Aussetzer, was zu vermeiden ist. Daher wird in Abhängigkeit der Pufferfüllstände bzw. einer Ableitung davon, die visuelle bzw. akustische Qualität gesenkt, um Rechenzeit einzusparen. Wenn der Puffer voll ist, darf dies im Gegensatz zur Aufnahme nicht in Framedrops auf der Seite der Quelle führen. Daher sollte hier ein blockierender Mechanismus angewendet werden, der dafür sorgt, daß der Demultiplexer bzw. die Quelle selbst vorübergehend die Datenlieferung zu den Kernen blockiert. Dies vermeidet, daß Daten verloren gehen.

Unter Umständen lassen sich Aussetzer bei der Wiedergabe nicht vermeiden, obwohl die zum Dekodieren benötigte Verarbeitungszeit bereits minimiert wurde. Daher muß die Synchronisationskomponente dies erkennen. Dazu werden die Daten wieder mit Zeitstempel und Gültigkeitsdauer versehen, was nach Möglichkeit der Demultiplexer oder der Kern vornehmen sollte. Um Rechenzeit zu sparen, sollten die zu verwerfenden Daten bereits vor dem Dekodieren aussortiert werden. Allerdings sind im Falle von Videodaten Frameabhängigkeiten zu beachten. So sind beispielsweise P-Frames von I-Frames abhängig. Die Methodik entspricht dem Smart-MPEG-Ansatz. Da bei zukünftigen Codecs dieser Ansatz aller Voraussicht nach nicht mehr funktioniert, sollte die Synchronisationskomponente oder direkt der Demultiplexer für das Verwerfen von Frames sorgen.

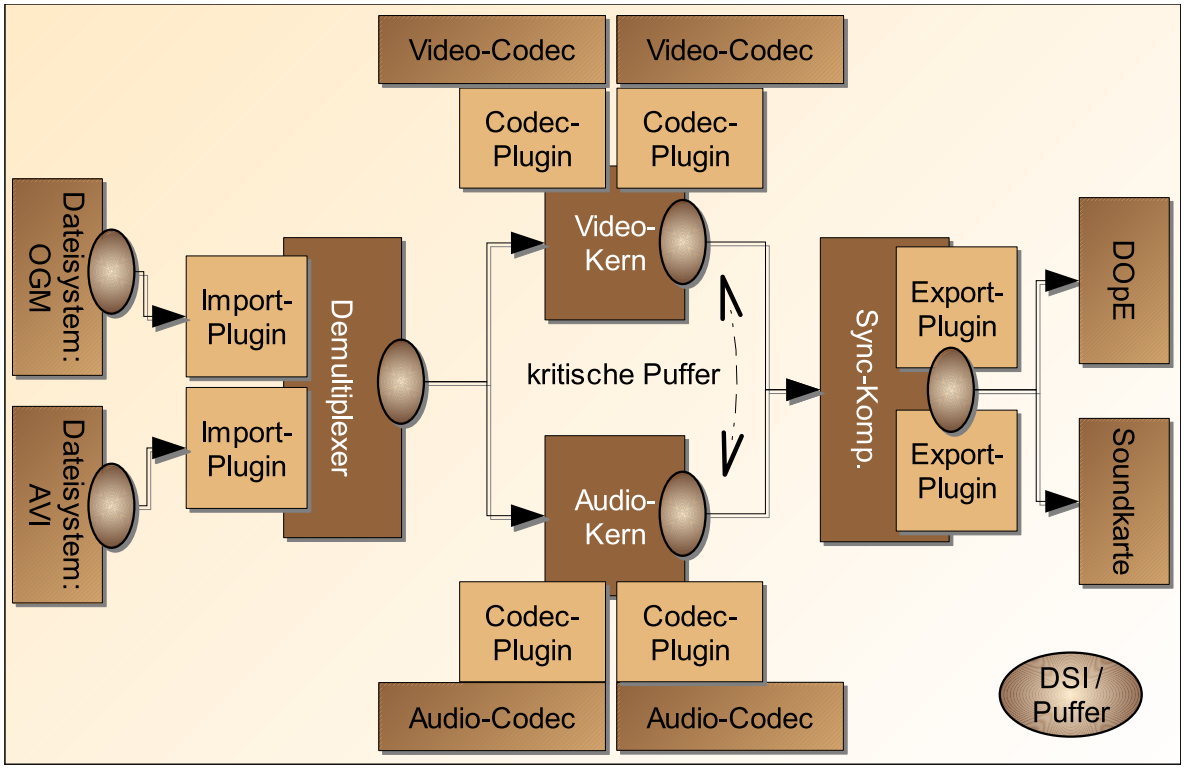

Abbildung 17: Modell für Wiedergabe

In [34] wird ein anderer Ansatz zur Realisierung von Quality-of-Service-Unterstützung vorgeschlagen: Dazu wird das Video nach Frametypen sortiert und in verschiedene Dateien zerlegt, also beispielsweise je eine Datei für I- , P- und B-Frames. Die QoS-Anforderungen werden durch das Quality-Assuring Scheduling der Festplatte realisiert. Die verschiedenen Frametypen werden mit der gewünschten Qualität von der Festplatte geliefert, z.B. 100% I- und 50% P- und 30% B-Frames einer Group of Pictures. Damit wird der Dekoder reguliert, da er nie alle Frames darstellen kann, weil sie einfach nicht innerhalb der GOP-Darstellungsperiode geliefert werden. Ein entscheidener Nachteil dieser Methode ist, daß sie nach jetzigem Kenntnisstand nicht mit H.264/AVC funktionieren wird: Die Verwendung von Slices und meheren Referenzbildern führt zu den gleichen Problemen wie beim Smart-MPEG-Konzept. Das Video kann nicht mehr eindeutig nach Frametypen zerlegt werden, da ein Frame aus unterschiedlichen Slice-Typen bestehen kann. Würde stattdessen nach Slice-Typen zerlegt werden, dann würden Frames mit verschiedenen Slice-Typen nicht vollständig dekodiert werden können, was sichtbar wäre.

Zusammgefaßt läßt sich feststellen, daß zur Lösung der im Abschnitt 3.1 vorgestellten Probleme bei der Wiedergabe andere Techniken als bei der Aufnahme zum Einsatz kommen:

1. Die Synchronität wird durch den Zeitstempel gewährleistet und unterliegt der Verantwortung der Synchronisationskomponente. Dazu kennt diese Komponente die benötigte Zeit vom Absenden der Daten bis zur Präsentation bei den Senken.

2. Die Schwankungsdauer der Verarbeitung ist problematisch, da die Eigenschaften der menschlichen Wahrnehmung zur akkuraten Präsentation von Audio und Video zwingen. Es wird versucht, durch die Verwendung von Puffern in Kombination mit dem Konzept der Qualitätsstufen-

Anpassung die Schwankungen so weit wie möglich zu neutralisieren.

3. Die Audio- soll gegenüber der Videoverarbeitung bevorzugt behandelt werden.

4. Alle Datenpakete werden mit einem Zeitstempel und ihrer Gültigkeitsdauer vom Demultiplexer bzw. Kern versehen. Dies dient auch der Feststellung über evtl. erfolgte Datenverluste und der für die Wiedergabe nötigen Informationen über den zeitlichen Bezug der Daten untereinander.

3.5.4 Ergänzungen zum allgemeinen Modell

Das im Abschnitt 3.5.1 vorgestellte Modell läßt sich nicht nur als Basis für das Abspielen bzw. die Aufnahme von Videos nutzen, sondern bietet zugleich die Möglichkeit als *Transkodierer* verwendet zu werden. Durch die gleichzeitige Nutzung von Demultiplexer als Quelle und Multiplexer als Senke für den Kern, ist es möglich, Videos in ein anderes Format zu überführen bzw. nur bestimmte Eigenschaften eines Videos, z.B. die Bildgröße oder die Lautstärke zu verändern.

Dank der flexiblen Auslegung des Modells durch die Verwendung von DSI gepaart mit Plugins sind viele weitere Nutzungsmöglichkeiten denkbar, z.B. Daten aus unterschiedlichen Quellen zu mischen. Beispielsweise könnten die Videodaten via Netzwerk eingelesen werden, die Audiodaten von der Soundkarte kommen und die Speicherung des Videos auf einem anderen Server erfolgen.

3.6 Intergration der Modelle in das DROPS Scheduling Framework

Die Integration der erstellten Modelle in das DROPS Scheduling Framework bedarf einiger detaillierterer Informationen zum Dekodieren bzw. Enkodieren von Video- und Audiodaten. So ist bei Video die Framerate ausschlaggebend für die Dauer einer Periode: $Periode_{Video} = 1/fps$. Bei Audiodaten ist die Ermittlung einer Periodendauer etwas schwieriger. Die kleinste Einheit für digitalisierte Audiodaten ist ein Sample. Die zeitliche Gültigkeit eines Samples hängt von dessen Samplerate ab. Allerdings läßt sich dies nur eingeschränkt zur Berechnung der Periodendauer nutzen, da aktutelle Codecs Samples gruppenweise verarbeiten. Bei MP3 sind beispielsweise 1152 Samples zu einem MP3-Frame zusammengefasst. Daher ist die mögliche Dauer einer Periode für MP3 $Periode_{MP3} = n * 1152 * 1/samplerate$ $(n \in \mathbb{N})$. Die Periodendauer ist immer ein Vielfaches der Sampledauer. Als nächstes wird ein mögliches Scheduling des Audio- und Videokerns betrachtet. Hierbei ist nicht nur die Intergration der Komponenten in das DROPS Scheduling Framework zu verdeutlichen, sondern auch wie dies in Kombination mit dem vorgestellten Konzept der Qualitätsstufen-Anpassung erfolgen kann.

▷ **Audio- und Videokern** Da als Basis für die vorgestellten Modelle jeweils bestehende Codecs genutzt werden, bei denen der laufende Dekodier- bzw. Enkodiervorgang nicht abgebrochen werden kann, sind begonnene Frames jeweils zu Ende zu kodieren. Dabei ist möglichst je ein Frame innerhalb einer Periodendauer zu bearbeiten. Dies wäre sichergestellt, wenn als Grundlage für die beim Scheduling eingeplante Berechnungsdauer pro Periode die Worst-Case-Zeit genutzt wird. Dies gilt, sofern die Worst-Case-Rechenzeit kleiner als die Periode wäre und der Job vom Admission-Server zugelassen wird. Im Quality-Assuring Scheduling-Verfahren entspräche dies einem mandatorischen Teil. Da aber die durchschnittliche De- bzw. Enkodierzeit wesentlich kleiner ist als die Worst-Case-Zeit, ist eine Einplanung basierend auf der Worst-Case-Berechnungsdauer nicht sinnvoll[1]. Außerdem ist i.

allg. die Enkodierung bzw. Dekodierung von Video- oder Audiodaten auch nicht sicherheitskritisch, was die Nutzung der Worst-Case-Zeit rechtfertigen würde.

Daher ist es sinnvoll, eine kleinere Berechnungszeit als die Worst-Case-Zeit zur Einplanung für das Scheduling einzukalkulieren. Bei bekannter Verteilungsfunktion für die Bearbeitungszeiten bietet sich das von den optionalen Teilen des QAS bekannte Konzept an: Die gewählte Qualität gibt an, wieviel Prozent der optionalen Teile komplett ausgeführt werden. Aus der Verteilungsfunktion läßt sich damit für den optionalen Teil die Reservierungszeit ermitteln. Die Abbildung 5 im Abschnitt 2.2.2 illustriert das Vorgehen anhand eines Diagrammes. Es werden somit sowohl die Verteilungsfunktionen der Audio- und Videoenkodierung sowie -dekodierung genutzt. Damit wird es möglich, die Bearbeitung als optionale Teile in das DROPS Scheduling Framework einzubeziehen. Im Gegensatz zum Smart-MPEG-Ansatz gibt es pro Periode nur einen optionalen Teil: die Dekodierung bzw. Enkodierung eines Frames. Bereits im Abschnitt 3.3 wurde gezeigt, daß die Einplanung einer GOP als Periode nicht sinnvoll ist. Aufgrund der durch die menschliche Wahrnehmung bedingten Anforderungen, wird Ton bevorzugt gegenüber Video behandelt. Dies wird realisiert, indem die Audioverarbeitung eine höhere Priorität als die Videoverarbeitung erhält.

Bei Verwendung der Kodierung im Rahmen des optionales Teils, kommen für den mandatorischen Teil nur andere Aufgaben in Frage, beispielsweise die notwendige Kommunikation mit dem Controller und das Übernehmen eventueller Änderungen der gewünschten Qualitätsstufen. Die Abbildung 18 zeigt die angedachte Einbindung in das Scheduling-Schema von DROPS.

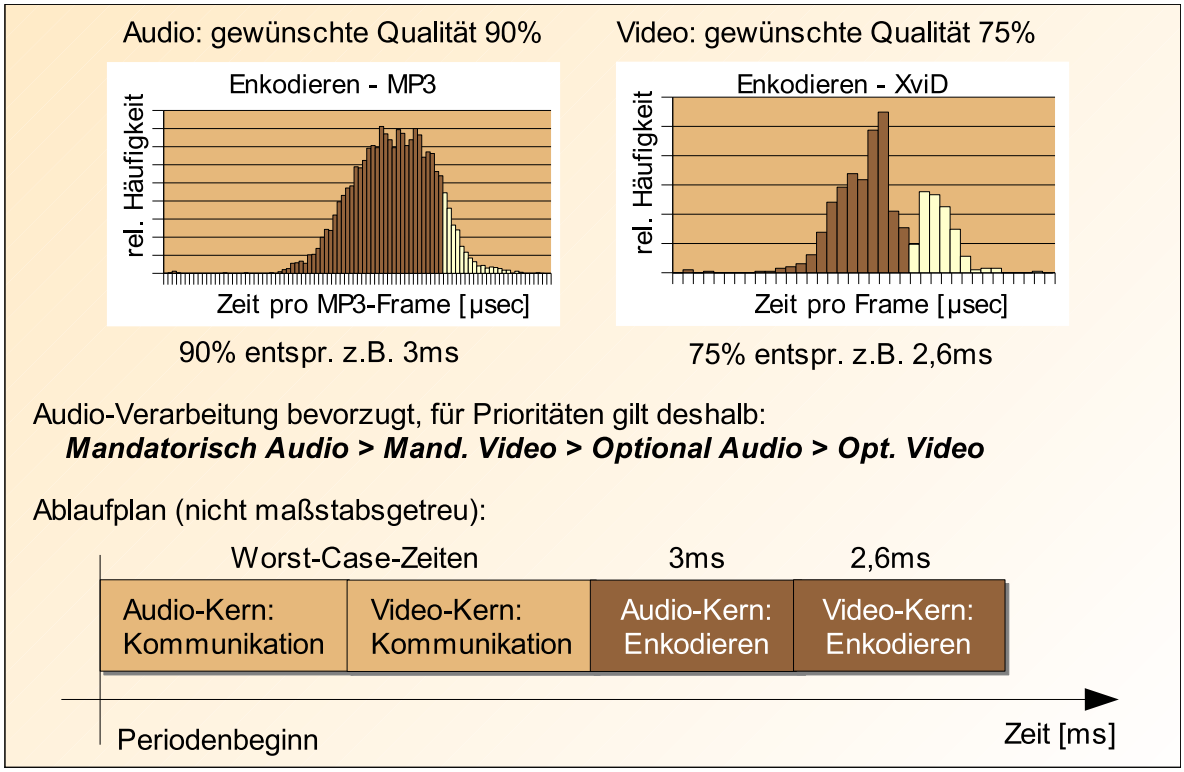

Abbildung 18: Angedachte Einbindung in das DROPS Scheduling Framework

Ein Problem dabei ist, daß sich die Verteilungsfunktionen für die verschiedenen Qualitätsstufen eines Codecs unterscheiden. Dies erschwert das Einbinden des Konzeptes der Qualitätsstufen-

Anpassung. Sinnvollerweise wird für Video und Audio daher nur jeweils eine Verteilungsfunktion zur Ermittlung der Reservierungszeit herangezogen. Beispielsweise kann es dem Nutzer überlassen werden, welche visuelle bzw. akustische Qualität er wünscht. Zur Wahrung des Verhältnisses zwischen visueller und akustischer Qualität werden beispielsweise die Verteilungsfunktionen von unterschiedlichen Qualitätsstufen für Audio und Video vom Benutzer bzw. vom Controller ausgewählt. Das Konzept der Qualitätsstufen-Anpassung sorgt idealerweise für eine Annäherung der benötigten Rechenzeit an die Reservierungszeit durch Variieren der visuellen oder akustischen Qualität.

Im allgemeinen ist zu beachten, daß die Verteilungsfunktionen nur dann exakt bekannt sind, wenn das komplette Video vorher vermessen wurde. Dies dürfte in der Praxis selten der Fall sein - vor allem bei dem angestrebten Einsatzgebiet als Videorecorder. Einen guten Anhaltspunkt für unbekannte Videos geben die durchgeführten Messungen (vgl. Anhänge E und F), da die Testvideos einen vernüftigen Schnitt aus ruhigen, bewegten und detailreichen Szenen bilden [11]. Ein genaue Vermessung der Testvideos in der gewünschten Auflösung auf dem Zielsystem ermöglicht damit eine Abschätzung der Reservierungszeit. Durch die dynamische Anpassung der Qualitätsstufen im Rahmen des genannten Konzeptes wird es ermöglicht, daß auch bei nicht exakt bestimmbarer Reservierungszeit möglichst wenige Framedrops auftreten und eine möglichst hohe visuelle bzw. akustische Qualität erreicht wird. Damit eignet sich die Kombination aus Quality-Assuring Scheduling und dem Konzept der Qualitätsstufen-Anpassung auch für nur nährungsweise bekannte Verteilungsfunktionen. Eine Klassifizierung von Videos nach den jeweiligen Berechnungszeiten für unterschiedliche Inhalte und Qualitätsstufen würde die Erfolgsaussichten des vorgestellten Modells weiter erhöhen. Im Rahmen dieser Diplomarbeit wird dies jedoch nicht näher untersucht. Verschiedene Forschungsprojekte und Belege beschäftigen sich mit der Klassifizierung von Videos oder artverwandten Themen, stellvertretend zu nennen wären [35], [36] und [37].

▷ **andere Komponenten des Modells** Die bisherigen Ausführungen in diesem Abschnitt behandelten in Entsprechung zu den vorgestellten Modellen nur den Audio- und den Videokern. Die anderen Komponenten sollten anders eingeplant werden. Der Controller muß so eingeplant werden, daß Vorgaben vom Benutzer akkurat umgesetzt werden. Zu beachten ist dabei, daß der Benutzer das Gefühl haben sollte, die Anwendung reagiere prompt auf seine Eingaben. An dieser Stelle sei auf [38] verwiesen. Die Einbeziehung der anderen Komponenten des Modells in das DROPS Scheduling Framework ist sehr stark abhängig von der in der Implementierung verwendeten Soft- und Hardware. So ist die Synchronisationskomponente abhängig von der Art, wie die Daten an die Grafikkarte bzw. Soundkarte übergeben werden. Die verwendeten Container-Formate bestimmen ihrerseits die Implementierung des Mulitplexers und Demultiplexers. Daher sind für diese Komponente keine allgemeinen Empfehlungen zur Einbindung in das Scheduling-Schema sinnvoll, außer daß dafür gesorgt werden muß, daß für die Kerne ausreichend Daten zur Verarbeitung zur Verfügung stehen. Auch müssen die verarbeiteten Daten bei begrenzter Puffergröße schnell genug von den Kernen abgeholt werden. Weitere wichtige Erkenntnisse zu diesen Forderungen werden im folgenden Abschnitt ”Anforderungen an Quellen, Senken und Übertragungskanäle” erklärt. Dies trifft insbesondere auf die Senken bzw. Quellen zu, z.B. Webcam, Soundkarte, Dateisystem oder Grafikkarte. Aber auch für die beiden Kerne, den Demulitplexer und den Multiplexer werden notwendige Anforderungen erklärt.

3.7 Anforderungen an Quellen, Senken und Übertragungskanäle

3.7.1 Grundlagen und Begriffe

Die Daten werden sowohl bei der Wiedergabe als auch bei der Aufnahme in Form von Strömen zwischen Quelle und Senke ausgetauscht. Bei kodierten Daten wird eine oder mehrere Verarbeitungskomponenten (z.B. Demultiplexer, Multiplexer und/oder Codec) zwischengeschalten, die dafür sorgen, daß die Daten dekodiert bzw. enkodiert werden können. Die Abbildung 19 stellt dies schematisch dar. Aus den im Abschnitt 2.1 geschilderten Forderungen und der vom Anwender gewünschten akustischen bzw. visuellen Qualität lassen sich auch gewisse Bedingungen an die Datenströme ableiten.

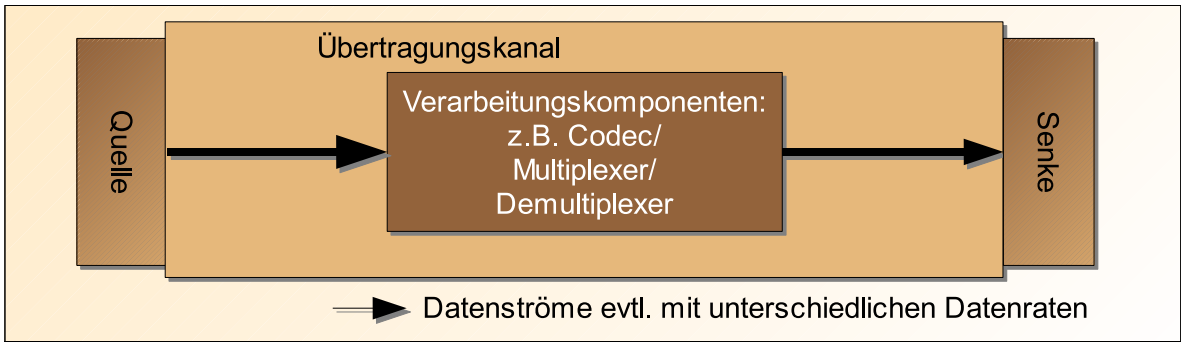

Abbildung 19: Schema der Datenströme

Bei unkodierten Daten, beispielsweise nach dem Dekodieren bzw. vor dem Enkodieren, bestimmen die Auflösung und das verwendete Farbformat die Datenmenge und zusammen mit der Framerate, also der Anzahl der Bilder pro Sekunde, die Datenrate der Videobilder. Die Datenrate ist das Verhältnis von Datenmenge und eines bestimmten Zeitintervalls und wird üblicherweise in Kilobit pro Sekunde (kBit/s) angegeben und somit als Bitrate bezeichnet. Beim Ton bestimmen die Samplerate, die Bits pro Sample sowie die Anzahl der Ton-Kanäle die Datenmenge respektive die resultierende Bitrate. Die Datenrate für unkodiertes Material ist konstant, da sich innerhalb eines Videos im Normalfall die gerade genannten Parameter für die Datenmenge nicht ändern. Durch Ermittlung der verwendeten Datenrate läßt sich die nötige Bandbreite des Übertragungskanals bestimmen.

Im Gegensatz zu unkodierten Video- oder Audiodaten, deren Bitraten konstant sind (*CBR*, constant bitrate, engl.), ist diese bei kodiertem Material in Abhängigkeit vom eingesetzten Codec auch variabel (*VBR*, variable bitrate, engl.). Teilweise können variable Ströme durch ihre durchschnittliche Datenrate charakterisiert werden. Diese Eigenschaft erschwert das Vermeiden von Aussetzern, da die Datenrate temporär auch größer sein kann als die durchschnittliche. In der Praxis wird versucht, Aussetzer durch die Verwendung von Puffern zu vermeiden. Angenommen wird dabei, daß der Puffer ausreichend dimensioniert wird. Das ermöglicht auch die Verwendung der durchschnittlichen anstelle der maximalen Bitrate bei der Verarbeitung des Videos.

Bei gegenwärtigen Übertragungsmedien und -wegen ist die Datenrate der limitierende Faktor für die visuelle bzw. akustische Qualität, selbst wenn kodiertes Videomaterial verwendet wird. Dies gilt sowohl für den verteilten Fall, z.B. Internetstreaming, als auch für den lokalen, bei dem dann meist der Durchsatz von Festplatte und Grafikkarte relevant ist. Andererseits limitiert der zur Verfügung

stehende Speicherplatz die Datenrate ebenfalls. Typische Einsatzgebiete und Bitraten für Videodaten zeigt die Tabelle 4, deren Inhalt teilweise aus [11] stammt. Tabelle 5 zeigt typische Datenraten für Audio.

Unkodiertes Bildmaterial (YUV420)			
	Auflösung	Framerate [fps]	Bitrate [kBit/s]
	176x96	12,5	2475
	640x352	25	66000
Kodiertes Bildmaterial (z.B. MPEG-4)			
Typische Einsatzgebiete	Auflösung	Framerate [fps]	Bitrate [kBit/s]
Modem-/Mobil-/ISDN-Streaming	176x96	12,5	ca. 30
DSL-Streaming	352x192	25	ca. 250
Download	640x352	25	ca. 750
Archivierung	704x400	25	> 1200

Tabelle 4: Typische Bitraten für Video (teilweise aus [11])

Unkomprimierte Audiodaten (PCM)
bei 44100Hz Samplerate und 16 Bits pro Sample mit 2 Kanälen: ca. 1378kBit/s (ca. 172kB/s)
Komprimierte Audiodaten (MP3, OGG Vorbis, ...)
typische Bitraten zwischen 64 kBits/s und 192kBit/s typische Sampleraten 22050Hz, 44100Hz, 48000Hz typisch 16 Bits pro Sample, Stereo

Tabelle 5: Typische Bitraten für Audio

Für die Realisierung von echtzeitfähigen Komponenten ist eine bekannte Datenrate enorm wichtig, da eine gleichbleibende Bitrate meist in konstanter Verarbeitungszeit resultiert. Schwankungen beeinflußen die Verarbeitungszeit und gefährden unter Umständen den Echtzeitbetrieb und die vom System gemachten Zusagen. Daher wird eine konstante Datenrate reserviert, wodurch die konstante Verarbeitungszeit für das Scheduling eingeplant werden kann. Zur Vermeidung von Aussetzern gilt es, diese Datenrate für die gesamte Dauer der Wiedergabe bzw. Aufnahme kontinuierlich zu liefern und zu verarbeiten. Ein weiterer Faktor ist die Latenz, also die Verzögerung der Datenübertragung durch den Übertragungskanal. Diese ist bei der Videowiedergabe bzw. Aufnahme unkritisch, da unter diesen Umständen das Video etwas später präsentiert bzw. aufgenommen wird.

Eine weitere Betrachtungsweise der Senken ist zum weiteren Verständnis notwendig: die unterstützten Zugriffsarten. Dabei werden zwei Zugriffsarten unterschieden: *linearer Zugriff* und *wahlfreier Zugriff*. Lineare Zugriff auf eine Senke bedeutet, daß die Daten von Anfang bis Ende hintereinander ausgegeben werden. Dies entspricht dem in der Einleitung vorgestellten Streaming. Beim wahlfreien Datenzugriff besteht eine Möglichkeit zur Positionierung innerhalb eines Datenstroms. Am einfachsten ist dies mit dem Ändern der Lese- oder Schreibposition bei einem Dateisystem zu erklären. Dabei ist es im Gegensatz zum linearen Zugriff möglich, auch mittendrin liegende Dateiteile zu lesen.

3.7.2 Aufnahme

Bei der Aufnahme von Videos liefern die Quellen, z.B. Webcam und Soundkarte, die Daten linear hintereinander. Ein Positionieren im Datenstrom ist nicht möglich. Die gestreamten Audio- und Videodaten sind zu enkodieren und dann zu speichern. Das entspricht dem bereits Vorgestellten: Die Datenrate ist bis zur Kodierung konstant, da die Daten unkodiert vorliegen. Nach dem Enkodieren ist es abhängig vom Codec, ob CBR oder VBR vorliegt. Für die meisten Codecs ist im Falle von VBR eine mittlere Datenrate einstellbar. Geringe Schwankungen der Bitrate nach dem Enkodieren sind verkraftbar, da im vorgestellten Modell die Puffer vor den beiden Kernen und damit vor dem Enkodieren liegen. Diese Puffer müssen so dimensioniert sein, daß unterschiedliche Entnahme- und Füllraten, wie sie insbesondere bei der Kombination von CBR mit VBR vorkommen, ausgeglichen werden können und somit nie zu wenig Daten in den Puffern sind. Beispielsweise kann ein Echtzeit-Dateisystem nur mit konstanter Datenrate arbeiten. Ein VBR-kodiertes Video muß aber mit schwankender Datenrate verarbeitet werden. Lösungsansätze und Details zur Pufferdimensionierung sind in [39] zu finden.

3.7.3 Wiedergabe

Für die lineare Wiedergabe ist nach dem Dekodieren die Datenrate konstant, aber vorher in Abhängigkeit des beim Enkodieren verwendeten Codecs möglicherweise auch variabel. Die Verarbeitungsdauer beim Dekodieren ist aber im Gegensatz zur Aufnahme durchaus relevant. Es muß sichergestellt werden, daß die Daten schnell genug dekodiert werden, um die konstante Datenrate zu liefern, die zum störungsfreien Abspielen des Videos nötig ist. Dazu dient das bereits vorgestellte Konzept der Qualitätsstufen-Anpassung, welches versucht die Bearbeitungsdauer innerhalb der für das Scheduling einzuplanenden Zeit konstant zu halten. Auch hier kann es bei VBR-kodierten Videos zu den gleichen Problemen mit der Datenpufferung kommen. Auch hier sind die Puffer entsprechend groß zu dimensionieren.

Die bisherigen Ausführungen berücksichtigen lediglich das lineare Abspielen, aber nicht das Springen innerhalb des Videos (seek, seeking, engl.) oder die beschleunigte Vorwärts- bzw. Rückwärtswiedergabe. Mit der Realisierung dieser Funktionen ändern sich auch die Anforderungen an die beteiligten Verarbeitungskomponenten.

▷ **beschleunigte Vorwärtswiedergabe** Im einfachsten Fall der beschleunigten Vorwärtswiedergabe, erhöht sich die nötige Datenrate proportional zur Geschwindigkeit des Abspielens, sofern alle Frames und Samples wiedergegeben werden sollen. Dabei ist diese Funktion unabhängig von den aktuell üblichen Container-Formaten wie AVI oder OGM realisierbar.

Wird stattdessen versucht, die Datenrate unverändert gegenüber der normalen Abspielrate zu belassen, ist es nicht möglich, alle Frames bzw. Samples zu übertragen. Daher müssen Daten ausgelassen werden. Das Auslassen von Audio-Samples ist hierbei nicht sinnvoll, da dies bei der Wiedergabe als abgehackt und damit als sehr störend empfunden wird. Daher sollte auf Ton komplett verzichtet werden, womit die gesamte Datenrate den Videodaten zur Verfügung steht. Unter der Annahme, daß die Daten mit aktuellen Video-Codecs und damit via 'inter'-Kodierung kodiert wurden sind, kann nicht jedes beliebige Frame ausgelassen werden. Die Abhängigkeiten der Frames untereinander sind zu berücksichtigen. Daher ist es zumindest denkbar, nur I-Frames zu übertragen. Bei MPEG-4 kodiertem Material kommen Keyframes nicht regelmäßig vor, beispielsweise bei MPEG-2, was dazu

führt, daß die wahrgenommene Abspielgeschwindigkeit schwankt. Dies kann aber durch Pufferung am Ende der Verarbeitungskette kompensiert werden. Ein Problem von I-Frames bzgl. der Bitrate sollte nicht unerwähnt bleiben: I-Frames benötigen mehr Speicherplatz und damit eine höhere Bitrate als P- oder B-Frames.

Bezugnehmend auf die Quelle sind keine weiteren Anforderungen als das Senden eines Datenstromes nötig, sofern die Quelldatenrate erhöhbar ist. Ist dies nicht der Fall, ist ein wahlfreier Zugriff auf die Quelldaten notwendig, um zu den zu übertragenden Frames zu gelangen. Dazu muß allerdings auch die Zielposition des Sprunges einer Kontrollinstanz bekannt sein. Ein Möglichkeit ist das Erstellen und Nutzen eines Indexes, dem die Position aller benötigten Frames bekannt ist.

▷ **beschleunigte Rückwärtswiedergabe** Die Annahme, daß die Rückwärtswiedergabe die Daten lediglich rückwärts mit der gleichen Bitrate wie beim normalen Vorwärtsabspielen erfordert, ist nur begrenzt korrekt. Bei RAW-Daten, also unkodiertem Video- bzw. Tonmaterial ist es möglich. Allerdings sind hierbei einige Einschränkungen zu machen: Die Quelle muß den wahlfreien Zugriff innerhalb der gespeicherten Daten unterstützen und es muß eine Kommunikation zwischen einer Kontrollinstanz zur Datenstromsteuerung und der Quelle möglich sein. Dies ist bei Dateisystemen einfacher zu realisieren, als z.B. bei Internet-Streaming-Angeboten. Allerdings gibt es Quellen wie TV-Karten oder Webcams bei denen diese Form der Wiedergabe ohne zusätzliche technische Hilfsmittel unmöglich ist.

Bei zeitgemäßeren Codecs wird die problemlose Rückwärtswiedergabe weiter erschwert: Dort sind Abhängigkeiten zwischen Keyframes und folgenden Frames zu beachten. Daher müssen zuerst die Keyframes gelesen und dekodiert werden, danach die Differenzdaten. Ein Puffer muß die Daten dann in die gewünschte Reihenfolge bringen.

Eine andere Möglichkeit ist die Begrenzung auf I-Frames. Dabei treten die gleichen Problem wie bei der beschleunigten Vorwärtswiedergabe auf: Keyframes treten möglicherweise nicht regelmäßig auf, womit ihre exakte Position bekannt sein muß. Die Realisierung ist aufwendig. Daher ist bei aktuellen Videoplayern (Xine [40], Mplayer [41]) die Rückwärtswiedergabe nicht implementiert.

▷ **Springen im Video** Das Springen innerhalb eines Videos wird einerseits vom Benutzer gesteuert, andererseits kann es auch für die Implementation einer Synchronisationskomponente genutzt werden (siehe dazu auch Kapitel 4.5).

Im folgenden Abschnitt wird angenommen, daß das Video mit Bild und/oder Ton innerhalb eines Containers gespeichert ist, welcher wiederum als physische Datei vorliegt. Allerdings sei darauf verwiesen, daß es sich nicht zwingend um Dateien handeln muß, welche als Quelle dienen. Beispielsweise können die Daten auch im Hauptspeicher liegen oder über Netzwerke erreichbar sein. Dateien dienen daher in diesem Abschnitt nur der Veranschaulichung von Quellen.

Betrachtet man das oft verwendete Springen innerhalb von Videos genauer, so ist auch eine Abhängigkeit vom verwendeten Container ersichtlich. Durch die im Container vorhandenen Synchronisationsinformationen ist es nötig, an genau definierte Stellen im Video zu springen, z.B. an Kennungen für den Frameanfang oder an den Beginn einer OGG-Page. Dies setzt eine genaue Kenntnis der Sprungziele und somit des Containers voraus. Am Beispiel von AVI und OGM werden einige Möglichkeiten zum Springen in Videos kurz erläutert.

▷ **AVI** Das AVI-Format beruht auf der Verwendung eines Indexes, welcher sich am Dateiende befindet. Um alle nötigen Informationen zur Darstellung des Videos zu erhalten, muß der Dateianfang und das Dateiende zuerst gelesen werden. AVI speichert im übrigen keine Zeitinformationen. Diese müssen bei Bildern anhand der Framenummer und bei Audio anhand der Byteposition innerhalb des Audio-Tracks rekonstruiert werden. Diese Fakten erklären, daß der AVI-Container nicht zum Streaming geeignet ist. Zu erwähnen ist, daß es jedoch möglich ist, aus AVIs ohne gültigen Index diesen aufwendig zu rekonstruieren. Ein Springen innerhalb des Containers erfolgt durch Nachsehen im Index und den anschließenden Sprung an die entsprechende Byteposition innerhalb der Datei. Ein weiteres Problem entsteht bei der Verwendung von Audio mit variablen Bitraten (VBR). Die Zeitinformation kann bei einem Sprung nicht exakt anhand der Byteposition rekonstruiert werden, was zu einem oft beobachteten Verlust der Ton-Bild-Synchronisation führt, wenn beispielsweise in AVIs mit VBR-MP3 hin und her gesprungen wird. Die Datenrate läßt sich für Audio durch eine avilib-API-Funktion auslesen; für Bilder ist nur die maximale Framegröße zu erfahren, was zumindest eine Nutzung der maximalen Bitrate zur Dimensionierung der Übertragungskanäle ermöglicht.

Zusammenfassend läßt sich sagen, daß ...

- ... für Streaming des Videos die Quelle wahlfreien Zugriff zum Lesen des Indexes unterstützen muß.
- ... die Datenrate sich meist ermitteln läßt, was das Reservieren von Bandbreiten im System ermöglicht.

▷ **OGM** Wie bereits erwähnt, benutzt OGM keinen Index. Alle nötigen Information zur Darstellung sind in jeder OGG-Page bzw. den darin enthaltenen Paketen vorhanden (siehe Abschnitt 2.5.2). Durch API-Funktionen der libogg ist es möglich, sich auf den Bytestrom zu (re-)synchronisieren. Diese Faktoren ermöglichen den Einsatz von OGM beim Streaming von Videos.

Allerdings wird ein Springen innerhalb des Videos und damit des Containers erschwert, da nicht einfach in einem Index nachgesehen werden kann. Bei OGM sind daher andere Techniken einzusetzen, beispielsweise:

- Das Lesen einer Datei von Anfang an bis zu der gewünschten Stelle. Dies hat den großen Nachteil, daß es bei einer begrenzten Datenrate zu einer inakzeptabel langen Wartezeit kommen kann. Evtl. muß der Strom vorher geschlossen und neu geöffnet werden, um vor die gewünschte Position zu gelangen. Bei dieser Technik ist kein wahlfreier Quellzugriff nötig.
- Das Erstellen eines Indexes durch das Vorab-Lesen der gesamten Datei, was aber zu einer langen Wartezeit zu Beginn des Abspielens führen kann.
- Das Erstellen eines Indexes während dem Abspielen. Dies bringt den Nachteil, daß der benötigte Speicher mit längerer Abspieldauer wächst und daß es nicht möglich ist, vorwärts zu springen.

Eine pragmatischere Lösung ist eine Suche nach dem Prinzip eines binären Baumes: Zuerst wird in die Mitte der Datei gesprungen und via libogg-API der Zeitstempel ermittelt. Ein Vergleich zwischen

gewünschtem und gefundenem Zeitstempel zeigt, ob die gewünschte Zeit vor oder hinter der aktuellen Dateiposition zu finden ist. Ein weiterer Sprung entweder zu 1/4 bzw. 3/4 der Dateigröße mit anschließendem Zeitstempel-Vergleich zeigt den Quadranten. Dies Vorgänge werden wiederholt, bis der gewünschte Zeitstempel gefunden wurde. Eine mögliche Optimierung ist das Springen innerhalb der Datei anhand von Schätzungen der Dateigröße im Bezug zu der aktuellen Dateiposition und deren Zeitstempel mit dem anschließenden Vergleich nach dem eben geschilderten Prinzip.

Ein Problem von OGM in bezug auf die Datenrate und somit auf das Streamingverhalten muß ebenfalls erwähnt werden: Es ist nur durch Lesen und Analysieren der kompletten Datei möglich, die korrekte Bitrate zu ermitteln.

Zusammengefaßt muß bei reinem Streaming nur linearer Zugriff von der Quelle unterstützt werden. Für des Springen innerhalb des Videos ist der wahlfreie Zugriff nicht unbedingt notwendig, aber sehr vorteilhaft. Die Tabelle 6 zeigt eine Zusammenfassung der in diesem Abschnitt gewonnenen Erkenntnisse in bezug auf AVI und OGM.

Container	***Streaming***	***exakte Ermittlung der Datenrate***	***Springen im Container***
AVI	erschwert	einfach bei CBR, schwierig bei VBR	einfach bei CBR (Index), evtl. fehlerhaft bei VBR
OGM	sehr gut	komplex und aufwendig	aufwendig, da kein Index vorhanden

Tabelle 6: Zusammenfassung der OGM- und AVI-Eigenschaften

3.7.4 Zusammenfassung

Durch das Bestimmen der Bitrate der verwendeten Videoströme wird es überhaupt erst möglich, Echtzeitzusagen zu machen. Damit ist die Kenntnis der Datenrate eine Grundvoraussetzung für den Echtzeitbetrieb. In der Tabelle 7 werden die Anforderungen an die Quellen, die Senken und die Verarbeitungskanäle nochmals zusammengefaßt.

3.8 Synchronisationsmechanismus

▷ **Aufnahme** Bereits bei der Aufnahme muß dafür gesorgt werden, daß die zeitlichen Bezüge zwischen Bild und Ton später exakt rekonstruiert werden können, um eine lippensynchrone Wiedergabe zu gewährleisten. Dies kann realisiert werden, indem alle Quellen jedem Audio-Sample bzw. Videoframe einen Zeitstempel sowie eine Gültigkeitsdauer zuordnen. Bezugnehmend auf das im Abschnitt 3.5.2 vorgestellte Modell zur Aufnahme muß der Multiplexer dafür sorgen, daß die Daten im Container so gespeichert werden können, daß die zeitlichen Bezüge bei der Wiedergabe wiederhergestellt werden. Wie das Video gespeichert wird, ist dabei vom Container-Format abhängig. Framedrops bzw. das Auslassen von Audio-Samples wird anhand der Zeitinformation erkannt. Der Multiplexer muß auch hier Container-spezifisch reagieren. Im Fall von *Ogg Media* kann dies beispielsweise durch Änderung der *granulepos* kodiert werden.

▷ **Wiedergabe** Um Bild und Ton synchron wiedergeben zu können, ist es nötig, daß sowohl dekodierte Audio- als auch Videodaten nahezu zeitgleich von den entsprechenden Ausgabegeräte präsentiert werden. Die einfachste Möglichkeit ist es, für Bild und Ton eine einheitliche Zeitbasis auf

Funktionalität	***Anforderung an Quellen***	***an Übertragungskanäle***	***an Senken***
Aufnahme	linearer Zugriff: Lesen und Weitergabe mit konstanter Bitrate	Annahme der Daten mit konstanter Bitrate (unkodiert) und Weitergabe mit konstanter Datenrate, evtl. Durchschnittsbitrate (kodiert)	Annahme der Daten mit konstanter Bitrate, evtl. Durchschnittsbitrate (kodiert)
normale Wiedergabe	linearer Zugriff: Lesen und Weitergabe mit konstanter Bitrate, evtl. Durchschnittsbitrate und zusätzlich evtl. wahlfreier Zugriff nötig z.B. bei AVI	Annahme der Daten mit konstanter Bitrate (kodiert), evtl. Durchschnittsbitrate und Weitergabe mit konstanter Datenrate (unkodiert)	Annahme der Daten mit konstanter Bitrate (unkodiert)
beschleunigte Wiedergabe	bei Übertragung aller Daten: proportionale Erhöhung der Datenrate mit linearem Zugriff *oder* wahlfreier Zugriff für selektive Übertragung von Frames	bei Übertragung aller Daten: proportionale Erhöhung der Datenrate	bei Übertragung aller Daten: proportionale Erhöhung der Datenrate
Rückwärtswiedergabe	wie beschleunigte Wiedergabe, aber wahlfreier Zugriff *unbedingt* nötig	wie beschleunigte Wiedergabe	wie beschleunigte Wiedergabe
Springen im Video	wahlfreier Zugriff (nicht unbedingt nötig bei OGM, aber sehr empfohlen)	-	-

Tabelle 7: Zusammenfassung der Anforderungen an Quellen und Übertragungskanäle

Grundlage der Systenuhr zu verwenden: Die Audio- und Videodaten werden in Pakete gleicher Abspieldauer zerlegt. Beispielsweise ist bei einem Video mit 25 Bildern pro Sekunde die Anzeigedauer für ein Bild 40ms. Der Ton wird ebenfalls in Pakete à 40ms zerlegt. Nun wird periodisch alle 40ms sowohl das Bild als auch das zugehörige Audiostück ausgeben. Ein Vorteil dieser Methode ist, daß die hinreichend exakte Position von Ton und Bild bekannt ist, um auf Lippensynchronität zu prüfen. Bei zu hoher Systemlast würden evtl. einige Bilder oder einige Audio-Samples fehlen, da diese dann nicht vollständig innerhalb der Periode dekodiert werden konnten. Daher sollte das alte Bild wiederholt gezeigt werden bzw. kein Ton ausgegeben werden.

Bei der praktischen Umsetzung dieses Konzeptes gibt es aber Probleme: Sowohl die Soundkarte als auch die Treiber nutzen Puffer. Die Größe des Puffer muß daher bekannt sein, um die durch Pufferung entstehenden Verzögerungen bei der Wiedergabe einzukalkulieren. Das größte Problem ist aber ein anderes: Soundkarten benutzen ihren eigenen Taktgeber als Zeitbasis zur Tongenerierung und -ausgabe. Damit werden zwei Uhren im System genutzt: eine auf der Soundkarte und die Systemzeit. Diese Uhren können auseinander driften (vgl. [42]). Das führt dazu, daß entweder der Ton relativ zum Bild zu schnell oder zu langsam abgespielt wird: d.h., daß Video und Audio ebenfalls auseinander driften und nicht mehr synchron wiedergegeben werden. Messungen am Testsystem ergaben, daß die Systemuhr ca. 0,23% langsamer ist als der Taktgeber der Soundkarte. Die im Anhang F gezeigte Abbildung 33 verdeutlicht dies. Dieser Drift ist bei längeren Videos bereits deutlich spürbar. Die kritische Grenze für die Synchronität (80ms) ist bereits bei rund 35 Sekunden erreicht.

Die Nutzung einer einheitlichen Zeitbasis auf Grundlage der Systenuhr ist somit nicht sinnvoll. Daher bedarf es eines anderen Ansatzes. Da aufgrund der Eigenschaften der menschlichen Sinnesorgane Störungen beim Ton eher als beim Bild wahrgenommen werden, ist es sinnvoll, den Taktgeber der Soundkarte als Zeitbasis zu nutzen. Der Taktgeber der Soundkarte läßt sich nicht einstellen. Allerdings kann man die Systemuhr mit dem Taktgeber der Soundkarte synchronisieren. Die Ausgabe der Audio- und Videodaten erfolgt anhand von Zeitinformationen der synchronisierten Systemuhr. Für die Prüfung auf Synchronität ist es daher nötig, die Ausgabezeitpunkte des aktuellen angezeigten Bildes und die des aktuell ausgebenen Audio-Samples zu kennen. Durch einen Vergleich der beiden Zeitinformationen läßt sich auf Lippensynchronität prüfen. Die Verwendung von Puffern, z.B. auf der Soundkarte, erschwert die exakte Bestimmung. Bei Verwendung eines Sound-Servers sollte dieser die Ermittlung des exakten Ausgabezeitpunktes eines Audio-Samples unterstützen, da sonst keine vernünftige Möglichkeit gegeben ist, den Uhrendrift zu bestimmen und auszugleichen.

Wenn entdeckt wird, daß Bild und zugehöriger Ton nicht lippensynchron präsentiert werden, muß die Synchronität wiederhergestellt werden. Dies kann durch Verzögerung der Audio- oder Videoausgabe erfolgen, je nachdem ob Bilder dem Ton hinterherlaufen oder umgekehrt. Auch eine Beschleunigung der Verarbeitung des hinterherlaufenden Mediums ist denkbar. Zusammenfassend müssen bei der Implementierung des Synchronisationsmechanismus für die Wiedergabe von Videos folgende Probleme gelöst werden:

1. Die Synchronisation von PC- und Soundkartenuhr, da beide als Zeitbasis dienen.
2. Die Entdeckung von fehlender Lippensynchronität durch Ermittlung der aktuell präsentierten Video- und Audioposition,
3. sowie eine geeignete Reaktion bei Verlust der Synchronität.

3.9 Zusammenfassung

Das Konzept der Qualitätsstufen-Anpassung sowie das Quality-Assuring Scheduling bilden die Grundlagen sowohl für das entwickelte Aufnahme- als auch für das Wiedergabe-Modell. Dabei wurde das Problem der Synchronisation von Bild und Ton berücksichtigt und ein Synchronisationsmechanismus entworfen und vorgestellt.

Im folgenden Kapitel wird erörtert, wie die vorgestellten Modelle implementiert wurden. Desweiteren folgen Details zur Implementierung der Synchronisationskomponente sowie Hinweise zur verwendeten Software.

4 Implementierung

4.1 Verwendete Softwareumgebung

4.1.1 L4Env

L4Env dient als Umgebung (*Env*ironment, engl.) zum Programmieren auf Basis der L4-Mikrokernel-Familie [43]. Das L4Env ist als Teil des *Dresden Real-Time Operating Systems* entwickelt worden und dient als einheitliche, minimale Basis zum Erstellen von Bibliotheken und Programmen für DROPS. Zusätzlich abstrahiert es die verwendete L4-*API* bzw. die Hardware-Architektur und fördert damit das Portieren von Software. Das L4Env ist die zentrale Basis für die Implementation des Videorecorders und -players.

4.1.2 DOpE

Das *Desktop Operating Environment* (*DOpE*) ist eine fensterbasierente grafische Benutzeroberfläche für DROPS. DOpE zeichnet sich durch gute Erweiterbarkeit, hohe Performance und geringen Speicherbedarf aus. Die grundlegende Architektur wird in [38] vorgestellt. DOpE verwendet sogenannte *Widgets*, die Basiselemente des Benutzerinterfaces, aus denen beispielsweise komplexere Fenster hierarchisch aufgebaut werden können. Ein Auszug der aktuell implementierten Widgets wird in Tabelle 8 vorgestellt.

Widget	***Funktionen***
Window	Darstellung eines Fensters
Grid	Organisation und Ausrichtung von Kind-Widgets
Label	Anzeige eines Textes
Button	Schaltfläche
Scale	Schieberegler
VScreen	Darstellung eines Bild mit Echtzeitzusagen

Tabelle 8: Verfügbare DOpE-Widgets (Auszug)

In bezug auf Echtzeitanforderungen und der damit verbundenen regelmäßigen Darstellung von Videobildern ist insbesondere der VScreen-Widget interessant. Bei Nutzung dieses Widgets sorgt DOpE für eine regelmäßige Aktualisierung des Bildschirminhalts. Die Ausgabe der Bilder wird außerdem bevorzugt und damit garantiert durchgeführt. Weitere Details zum VScreen-Widget sind im Abschnitt 4.5.1 zu finden.

Aktuell wird für DOpE Hardwareunterstützung implementiert, welche die Darstellung von Videos erheblich beschleunigen wird. In der gegenwärtigen Version realisiert DOpE sämtliche Anpassungen der auszugebenden Daten via Software. Durch die Unterstützung der in Grafikkarten verbauten Hardware ist ein signifikanter Geschwindigkeitszuwachs erreichbar.

4.1.3 DROPS Driver Environment

DDE, das Kürzel für DROPS Driver Environment, dient als Umgebung zum Portieren von *Linux*-Treibern auf DROPS. Dazu bietet DDE verschiedene Nachbildungen von Teilen des Linux-Kernels

2.4, welche zum Kompilieren und Nutzen der Treiber notwendig sind. Außerdem bietet DDE einen *I/O*-Server für L4. Dieser dient der Interaktion der Hardware mit den portierten Treibern. Details von DDE sind in [44] zu finden. Praktischerweise wurden als Fallbeispiele für die Implementation von DDE einige Soundtreiber sowie Teile der *Open-Sound-System*-Architektur auf DROPS portiert. Diese Portierungen werden in der Implementierung genutzt (siehe dazu 4.5.1).

4.1.4 Portierte Bibliotheken und weitere Software

Basierend auf dem in der Vorarbeit [1] auf DROPS portierten XviD Codec (Version 0.91), wurde der Videoplayer sowie der Videorecorder implementiert. XviD dient jedoch nur der De- und Enkodierung von Videodaten. Um Audiodaten zu enkodieren, wurde auch LAME portiert. Das Dekodieren von MP3 übernimmt die in einer Bibliothek gekapselte Dekodierfunktionalität von mpg123, welche ebenfalls umgesetzt wurde. Für die Realisierung des im Abschnitt 3.4 angesprochenen Postprocessings, ist die portierte *libpostprocess* verantwortlich. Um zu testen, ob das implementierte System auch mit anderen Videoformaten als MPEG-4 zurechtkommt, wurde *libavcodec* (Teil von ffmpeg [45]) ebenfalls für DROPS verfügbar gemacht. Eine Tabelle mit allen unterstützten Video- und Audioformaten ist im Anhang D zu finden.

Als Container-Formate für den Videoplayer bzw. -recorder dienen AVI und OGM. Dafür wurden die *avilib* und die *libogg* verwendet. Beide kapseln ihre Funktionalität zum Zugriff auf AVI- bzw. OGM-Container in einer Bibliothek. Die avilib wurde bereits in der Vorarbeit auf DROPS umgesetzt. Zusätzlich wurde die libogg ebenfalls portiert. Zur einfacheren Nutzung bestehender Software, z.B Import-Plugins, wurde die *ogmlib* für DROPS verfügbar gemacht. Die ogmlib nutzt ein zur avilib nahezu identisches Interface, wobei jedoch auf Schreibfunktionalität für OGM verzichtet wurde. Für das Speichern wird direkt die libogg verwendet, da ihre Schnittstellen in dieser Hinsicht besser geeignet sind, als die der avi- bzw. ogmlib.

Die Portierungen verliefen weitestgehend problemlos, waren jedoch teilweise langwierig, da verschiedene Funktionen wie String- und Mathematik-Routinen nicht verfügbar waren und somit ebenfalls mit portiert werden mußten. Als Grundlage dafür wurde die *dietlibc* genutzt. Nur XviD konnte unverändert übernommen werden. Bei den meisten anderen Bibliotheken mußten mindestens einige Ausgaberoutinen umgebaut werden. LAME wurde zusätzlich noch so erweitert, daß die Qualitätsstufen auch während dem Enkodieren geändert werden können. Dies war bisher bei LAME nicht vorgesehen: Die eingestellte Qualität galt für den gesamten Enkodierprozess.

Als Basis für die Implementierung von Demultiplexer, Audio- und Videokern wurde *dsi_staticfs* verwendet. Diese RAM-Disk tauscht ihre Daten via DSI aus und hat bereits alle dazu notwendigen Funktionen implementiert. Daher wurde dsi_staticfs so modifiziert, daß bei der Anforderung der Daten, diese zuerst geholt, dann ggf. en- oder dekodiert und anschließend via DSI weitergegeben werden. Als Ersatz für ein fehlendes Dateisystem konnte dsi_staticfs nicht genutzt werden, da einige portierte Bibliotheken nur auf Basis der Dateizugriffsfunktionen von Unix funktionieren.

Mangels eines Dateisystemes, welches die aufgenommenen Videos persistent speichert, wurde ein ”Dummy-Dateisystem” implementiert. Es arbeitet als RAM-Disk und unterstützt einige ausgewählte *UNIX*-Befehle zum Dateizugriff wie *open()*, *close()*, *read()* und *write()*. Dateien werden via *grub* dem Adreßraum des Systems zugänglich gemacht. In diesem Speicherbereich werden dann die Daten abgelegt. Dies schränkt die praktische Nutzung des Videorecorders enorm ein, da die Videos

weder dauerhaft gespeichert werden können, noch ausreichender Speicherplatz zur Verfügung steht. Im übrigen wurde das Dummy-Dateisystem auch als Quelle für darzustellende Videos verwendet.

Damit stand die Basis für die Implementation des Gesamtsystems für das *Dresden Real-Time OPerating System* zur Verfügung. Eine Zusammenfassung der verwendeten Software ist im Anhang D zu finden.

4.2 Implementierung der Modelle

Als Infrastruktur zur Implementation des Videorecorder und -players wurden die bereits im Abschnitt 3.5.3 und 3.5.2 vorgestellten Modelle verwendet. Im Wesentlichen wurden diese so umgesetzt, wie es die genannten Abschnitte schildern. Einige Abweichungen davon gilt es hierzu dennoch zu erläutern. So wurden keine Export-Plugins implementiert. Die Synchronisationskomponente greift direkt auf die Senken zu, da nur DOpE und eine Soundkarte als mögliche Senken zur Verfügung standen. Vom Multiplexer werden die Video- und Audiodaten in einen OGM-Container auf dem Dummy-Dateisystem geschrieben. Auch hier wurde mangels Alternativen auf Export-Plugins verzichtet. Weiterhin wird im Gegensatz zu den vorgestellten Modellen der Multiplexer und die Synchronisationskomponente ebenfalls via Import-Plugin mit den Audio- bzw. Videokern gekoppelt. Das ermöglicht, die beiden Komponenten auch losgelöst vom Gesamtsystem beispielsweise in anderen Projekten zu nutzen. Der eigentliche Datentransfer wird im Abschnitt 4.3 detaillierter betrachtet. Die Abbildung 20 stellt die implementierte Infrastruktur schematisch dar.

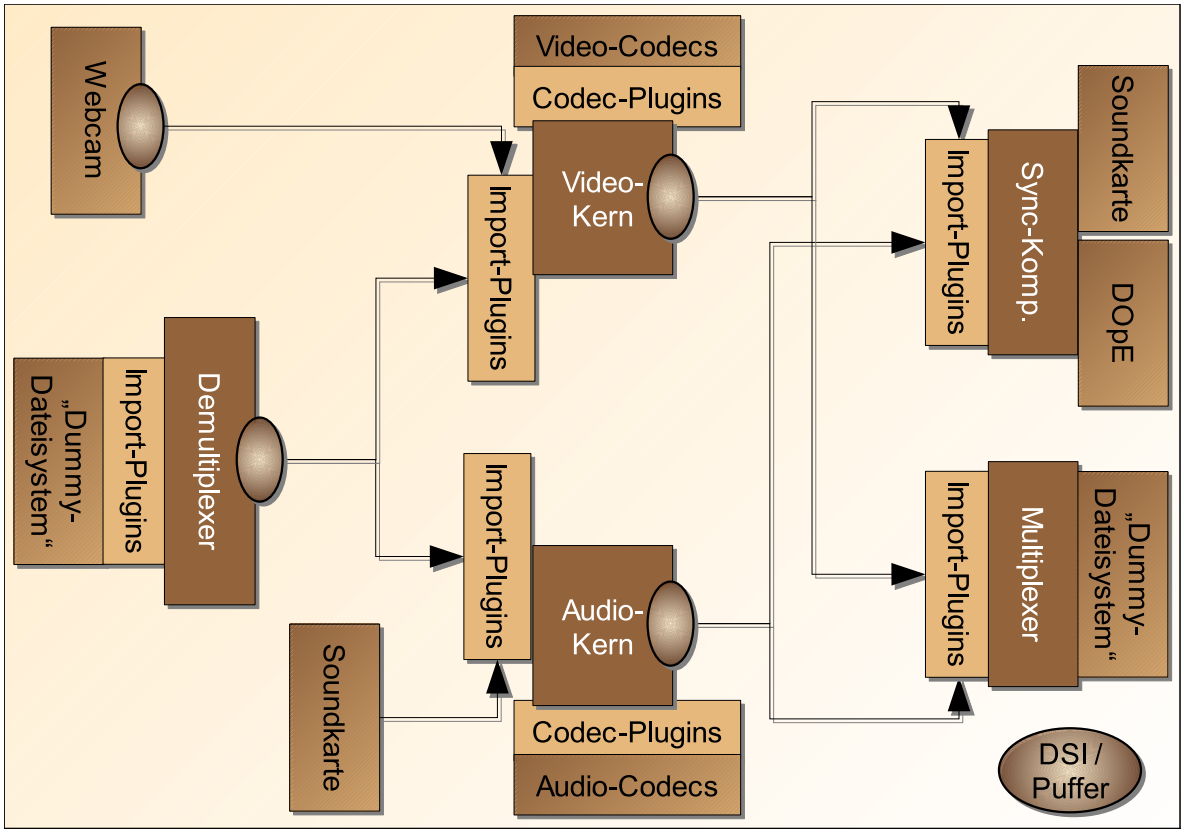

Abbildung 20: Implementierte Infrastruktur

Die Steuerung der Qualitätsstufen-Anpassung erfolgt nicht, wie im Entwurf beschrieben, ausschließlich durch den Controller (nicht dargestellt), sondern die Qualitätsstufe wird dynamisch anhand der Pufferfüllstände von den Kernen selbst verändert. Dazu werden die Pufferfüllstände via DSI-lib ausgelesen. Zweck ist es, den Kommunikationsoverhead einzusparen und nicht auf die Ergebnisse des Controllers warten zu müssen, da dies den Echtzeitbetrieb gefährden kann. Der Benutzer kann dennoch manuell die gewünschte Qualitätsstufe einstellen.

Wie im entworfenen Modell auch, wurden Import- und Codec-Plugins verwendet und implementiert. Auf Export-Plugins wurde, wie bereits erwähnt, verzichtet. Tabelle 9 zeigt die erstellten Import- und Codec-Plugins. Die Schnittstellen der Plugins sind in der Tabelle 10 beschrieben. Die grundlegenden Eigenschaften und Befehle der Schnittstellen haben sich gegenüber der Vorarbeit nur unwesentlich verändert, daher wird an dieser Stelle auf [1] verwiesen.

Die angedachte Integration der Modelle in das DROPS Scheduling Framework wurde vorbereitet, jedoch nicht vollständig implementiert. Der Grund dafür ist, daß zum Zeitpunkt der Erstellung dieser Arbeit der QAS-Scheduler nocht nicht fertiggestellt ist. Somit wurden im Quellecode entsprechende Scheduler-Aufrufe auskommentiert.

Codec-Plugin	***Plugin-Funktion***
aud_codec_lame	zur Enkodierung von PCM-Samples nach MP3 via LAME
aud_codec_mp3	zum Dekodieren von MP3 via mpg123-lib
aud_codec_pass	einfaches Weiterreichen ohne Veränderung der Daten nützlich bei PCM-zu-PCM-Konvertierung, z.B. Downmixing
vid_codec_ffmpeg	zur Dekodierung von ”nicht-MPEG-4-Video”
vid_codec_pass	einfaches Weiterreichen ohne Veränderung der Daten
vid_codec_xvid	zum De- und Enkodieren von MPEG-4 via XviD 0.91
Import-Plugin	***Plugin-Funktion***
import_avi	Video- und Audioimport aus AVI-Containern
import_mp3	Import von Audiodaten aus mp3-Dateien
import_ogm	Video- und Audioimport aus OGM-Containern
import_raw	Import von Video und Audio aus Dateien ohne Container
import_dsi_staticfs	Import von Video-RAW-Daten von dsi_staticfs
import_vcore	Importieren vom Audio- bzw. Videokern
import_vdemuxer	Importieren von Demultiplexer (genutzt von den Kernen)
import_soundcard	Audioimport von der Soundkarte
import_webcam	Videoimport von der Webcam

Tabelle 9: Implementierte Codec- und Import-Plugins

4.3 Datentransfer zwischen den Komponenten

Zum Datentransfer zwischen den Komponenten wird DSI verwendet. Daten werden beim *Drops Streaming Interface* immer paketweise übertragen. Dies führt dazu, daß der Datenstrom in Pakete zerlegt werden muß. Bei Video entspricht ein Paket einem Frame. Bei unkodiertem Audiomaterial wird eine gewisse Anzahl von PCM-Samples in ein Paket gepackt. MP3-Daten sind bereits frameweise kodiert, daher entspricht ein Frame auch hier einem Paket.

Als Beispiel für eine Übertragung wird ausgehend vom Demultiplexer der Weg eines Videoframes hin zur Synchronisationskomponente geschildert. Die Abbildung 21 stellt dies schematisch dar.

Schnittstelle der Codec-Plugins	***Beschreibung***
init()	Initialisierung des Plugins sowie des Codecs
step()	En- oder Dekodieren eines Frames
close()	Schließung von Plugin und Codec
Schnittstellen der Import-Plugins	***Beschreibung***
init()	Initialisierung des Plugins sowie der Quelle
step()	Import eines Frames bzw. Audio-Samples
commit()	Daten werden nicht mehr benötigt, Plugin kann Daten verwerfen
seek()	Kommando zum Springen innerhalb der Quelle, gegenwärtig nur vom Demultiplexer verwendet (ohne Funktion, wenn Quelle kein Seek unterstützt)
close()	Schließung von Plugin und Quelle

Tabelle 10: Schnittstellen der Plugins

Der OGM-Container, der die Videodaten enthält, wird mit grub in den RAM übertragen. Mit Hilfe des Dummy-Dateisystems und der ogmlib greift das OGM-Import-Plugin auf die Videoframes zu. Das Frame wird via DSI an den Videokern übergeben. Dieser sorgt dafür, daß das Frame dekodiert wird. Dazu übergibt er es über das Codec-Plugin dem entsprechenden Video-Codec, z.B. XviD. Der Videocodec versucht nun, das Frame zu dekodieren. Dazu bedarf es aber Informationen über das Frame, welche im kodierten Bild selbst nicht vorliegen. Unter anderem sind dies die Bildgröße sowie die Framerate. Diese Informationen sind im Container gespeichert. Daher ist es notwendig, diese Daten bereits im Demultiplexer auszulesen und mit an den Videokern zu übergeben.

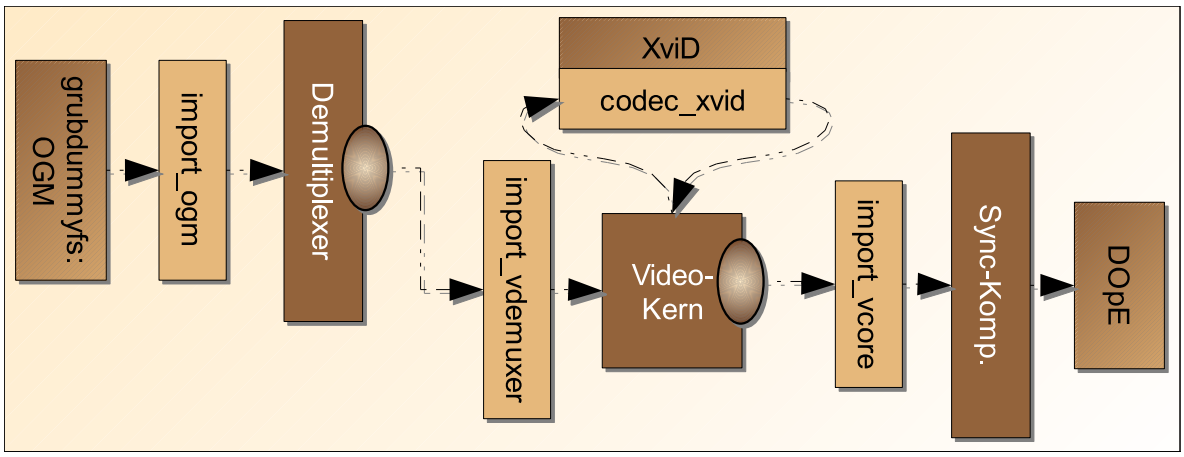

Abbildung 21: Weg eines Videoframes bei der Wiedergabe

Nach erfolgreicher Dekodierung wird das Bild nun ebenfalls über DSI an die Synchronisationskomponente geschickt. Die Synchronisationskomponente braucht wiederum andere Informationen über das Frame, um entscheiden zu können, wann dieses Bild darzustellen ist. Diese Zeitinformationen sind bereits im Container kodiert. Es ist auch hier nötig, daß diese Informationen vom Demultiplexer ausgewertet und durch den Kern bis zur Synchronisationskomponente mitgeliefert werden. Bei der Übertragung von Audio, sowie bei der Aufnahme von Videos erfolgt der Datenaustausch analog zum eben geschilderten Weg.

Aus diesem knappen Beispiel ist ersichtlich, daß verschiedene Komponenten jeweils andere Informationen über die Frames bzw. Samples brauchen, um zu funktionieren. Daher ist es notwendig, daß bereits die erste Komponente, z.B. der Demultiplexer, alle nötigen Informationen mitschickt. Dazu wird ein *Frameheader* definiert, welcher in jedem übertragenen Paket mitgesendet wird. Tabelle 11 zeigt auszugsweise, welche Daten notwendig sind und im Frameheader übertragen werden. Teilweise ändert z.B. der Kern bestimmte Daten im Frameheader auch im Nachhinein. So wird nach erfolgreicher Dekodierung z.B. das Format von MPEG-4 auf RAW geändert.

Notwendigkeit	***Eintrag***	***Funktion***
alle Pakete	format	Format des Pakets. z.B. MPEG-4, RAW, MP3 oder PCM
	frameID	fortlaufende Nummer zur Reihenfolgebestimmung
	frameSize	Anzahl der Bytes im eigentlichen Frame
	type	Typ des Pakets: Audio oder Video oder Text, zusätzlich als Flag bei notwendiger Resynchronisation (Reset-Sync-Flag) oder bei veränderten Attributen, z.B. Bildgröße (Reset-Attr)
Videodaten	vi.fourCC	*FourCC*, Kennung über verwendeten Codec
	vi.colorpspace	Farbraum, beispielsweise *YUV420* oder *RGB*
	vi.framerate	Framerate, Anzahl der Bilder pro Sekunde
	vi.xdim	Breite des Bildes
	vi.ydim	Höhe des Bildes
Audiodaten	ai.samplerate	Samplerate
	ai.channels	Tonkänale, z.B. Stereo (2) oder Mono (1)

Tabelle 11: Frameheader (Auszug)

4.4 Interaktion der Komponenten

Am Beispiel der einfachen Wiedergabe eines Videos soll die Interaktion der Komponenten untereinander näher beleuchtet werden. Die Vorgänge sind für die Aufnahme von Videos prinzipiell gleich. Die Abbildung 22 verdeutlicht die Vorgänge.

Der Controller erhält vom Nutzer das Kommando, ein bestimmtes Video, welches als Datei vorliegt, abzuspielen. Vom Controller wird überprüft, ob die für die Wiedergabe des Videos notwendigen Komponenten im System vorhanden sind (nicht dargestellt). Wenn alle Komponenten, nämlich Synchronisationskomponente, Demulitplexer sowie die beiden Kernen, erfolgreich gefunden wurden, wird als erstes dem Demultiplexer mitgeteilt, daß die gewünschte Datei geöffnet werden soll, und daß Bild und Ton angefordert wird. Danach wird vom Videokern ein Import-Plugin ausgewählt, in diesem Fall das spezifische für die Kommunikation mit dem Demultiplexer. Gleiches passiert auf Seiten des Audiokerns (nicht dargestellt). Als vierter Schritt folgt nun das Kommando an die Synchronisationskomponente, den Transfer von Audio- und Videodaten zu starten. Eigentlich sollte für die Synchronisationskomponente auch ein Import-Plugin auswählt werden, da aber momentan nur Plugins zur Kommunikation mit den beiden Kernen zur Verfügung stehen, entfällt dies. Nachträglich ist die Wahl des Plugins einfach zu implementieren. Damit hat der Controller alle nötigen Befehle zur Wiedergabe bereits gesendet.

Jetzt wird von der Synchronisationskomponente eine Kettenreaktion in Gang gesetzt: Dazu wird ein Befehl an den Videokern gesendet, daß Frames im RAW-Format erwartet werden (Schritt V1).

Anschließend wird die Übertragung gestartet. Der Videokern seinerseits sendet nun ebenfalls den Befehl an den Demultiplexer, mit dem Austausch der Frames zu beginnen. Nachdem der Datentransfer zwischen Demultiplexer und Videokern läuft, wird vom Kern festgestellt, daß beispielsweise MPEG-4-kodierte Frames empfangen werden. Das korrekte Codec-Plugin, in diesem Fall das XviD-Plugin, wird vom Kern ausgewählt, da von der Synchronisationskomponente RAW-Daten gefordert wurden. Nach dem anschließenden Dekodieren, werden die RAW-Frames an die Synchronisationskomponente weitergeleitet (Schritt V5).

Quasi parallel dazu läuft der Vorgang für Audio ab (nicht dargestellt): Diesmal erwartet die Synchronisationskomponente PCM-Samples. Dies wird dem Audiokern mitgeteilt, sowie der Transfer gestartet. Der Audiokern kommuniziert mit dem Demultiplexer, welcher dann beispielsweise MP3-Frames aus dem Container holt. Diese werden an den Audiokern gesendet, wo das passende Codec-Plugin auswählt wird und die Frames anschließend dekodiert werden.

Nachdem Bild und Ton bei der Synchronisationskomponente angekommen sind, werden diese im Zusammenspiel mit DOpE bzw. der Soundkarte präsentiert. Weitere Details zur Implementation der Synchronisationskomponente sind im Abschnitt 4.5 vorgestellt.

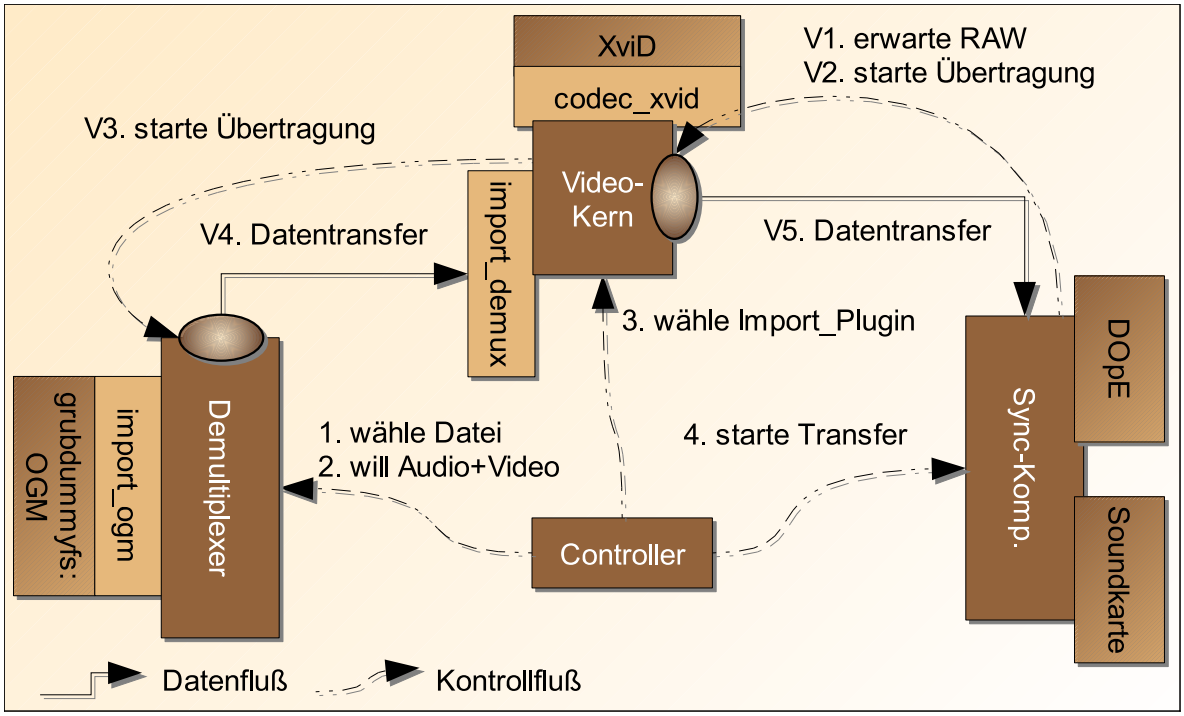

Abbildung 22: Komponenteninteraktion bei der Wiedergabe (Auszug)

Das Ende des Videos setzt eine Kettenreaktion in umgekehrter Reihenfolge in Gang (nicht dargestellt): Der Demultiplexer bemerkt das Ende der Video- bzw. Audiodaten und schließt den Datenkanal zu den Kernen. Ihrerseits schließen sie die Verbindung zur Synchronisationskomponente. Diese bemerkt dies und meldet dem Controller das Ende des Abspielens. Andererseits wird, wenn vom Benutzer gewünscht, die Videowiedergabe abgebrochen, indem die Kette von der Synchronisationskomponente aus geschlossen wird. Der Controller sendet das erforderliche Kommando daher nur zur Synchronisationskomponente.

4.5 Implementierungsdetails zur Synchronisationskomponente

Ausgehend von den im Abschnitt 3.8 entwickelten Ideen zur Implementierung der Synchronisationskomponente zur Wiedergabe von Videos müssen hauptsächlich folgende Probleme gelöst werden:

1. die Bestimmung der Bildposition und -ausgabe anhand der Systemuhr und Bestimmung der exakten Audioposition auf Soundkartenbasis
2. geeignete Reaktionen auf fehlende Synchronität
3. Synchronisation der Soundkarten- und Systemuhr (vgl. Abschnitt 4.5.5)

Die Realisierung einer solchen Wiedergabekomponente ist sehr aufwendig, was die folgenden Abschnitte beschreiben. Es sei darauf hingewiesen, daß vorausgesetzt wird, daß Bild und zugehöriger Ton bereits dekodiert wurden. In der vorliegenden Implementation des Videoplayers sorgen der Audio- bzw. der Videokern für das Dekodieren.

4.5.1 Ausgabegeräte

Zunächst folgt eine kurze Betrachtung, wie die dekodierten Daten DOpE bzw. der Soundkarte übergeben werden. Diese Ausführungen sind für das weitere Verständnis der Funktionsweise der Synchronisationskomponente notwendig.

▷ **Video** Zur visuellen Darstellung wird der VScreen-Widget von DOpE genutzt. Dabei wird ein in den Adreßraum der Synchronisationskomponente gemappter Speicherbereich mit den RAW-Bilddaten gefüllt. DOpE sorgt in Abhängigkeit der eingestellten Framerate für die regelmäßige Aktualisierung des Videospeichers und damit der Darstellung am Ausgabegerät. Desweiteren ist der VScreen-Widget ein Realtime-Widget (vgl. [38]), dessen Inhalt damit garantiert dargestellt wird. Ein Warten auf das von DOpE generierte Sync-Signal, z.B. alle 40ms bei 25fps, vermeidet Zugriffskonflikte auf den shared-memory-Bereich und sorgt automatisch für ein richtiges Timing beim Darstellen.

In der gegenwärtigen Implementierung von DOpE ist dieser Mechanismus nur eingeschränkt nutzbar, da er nur für exakt 25 Bilder pro Sekunde ausgelegt ist. Bei einer anderen Bildwiederholfrequenz wird der VScreen-Widget nicht als Echtzeit-Widget behandelt, was diesen momentan für die Wiedergabe von Videos mit anderen Frameraten unbrauchbar macht. Die realisierte Lösung besteht daher aus der Nutzung des VScreen-Widgets mit 25fps für alle Videos, auch mit anderer Framerate, und der Einführung eines eigenen Zeitgebers - dem *Metronome* (dt. das Metronom, der Zeitmesser).

Das Metronom ist einfach aufgebaut: Er berechnet aus der vom Video vorgegebenen Framerate die Dauer, für die ein Bild dargestellt werden soll. Er mißt dann pro Bild die Zeit mit Hilfe der Systemuhr über Kernelroutinen, die zur Darstellung dieses Bildes benötigt wird und legt sich sowie die Bildaktualisierung für den Rest der berechneten Zeit schlafen.

Ein besonderer Vorteil dieser Methode ist, daß die berechnete Zeit auch unabhängig von der Videoframerate einstellbar ist und damit die Bilddarstellung beschleunigen oder verlangsamen kann. Von dieser Möglichkeit macht im übrigen das schnelle Abspielen gebrauch. Ein Nachteil sei aber nicht verschwiegen: Aufgrund des nicht synchronisierten Zugriffs auf den shared-memory-Bereich ist es möglich, daß Teile des alten und des neuen Bildes gleichzeitig dargestellt werden. Bei den verwendeten Testsystemen und sämtlichen Testvideos wurde dieser Effekt jedoch nicht sichtbar.

Das Ergebnis zur Darstellung des Bildmaterials ist eine Kombination eines DOpE-Realtime-Widget mit garantierter Darstellung und einer flexiblen Zeitgebung zur Aktualisierung des Bildschirminhaltes.

▷ **Audio** Im Verhältnis zur Bilddarstellung ist die Audioausgabe einfach realisiert. Über den mit DDE portierten OSS-Treiber wird der PCM-Sound via *write*-Kommando an die Soundkarte übergeben. Einstellungen wie beispielsweise die Samplerate, Kanal-Anzahl und Audioformat werden über *ioctl*-Befehle vorgenommen. Die vorliegenden PCM-Samples werden solange geschrieben, bis der write-Befehl blockiert. Dieser Fall tritt ein, wenn der vom Treiber reservierte Speicher gefüllt ist. Ein Timing geschieht daher nur indirekt dadurch, daß bei gleichbleibenden Einstellungen für Samplerate, Format und Kanalanzahl die Puffer mit einer konstanten Geschwindigkeit geleert werden, so daß der blockierte write-Befehl fortgesetzt wird.

In bezug auf Echtzeitanforderungen sei vermerkt, daß angenommen wird, daß die Daten ohne Verzögerung dem Treiber bzw. der Soundkarte übergeben werden. Die Bearbeitung erfolgt innerhalb einer konstanten Zeit. Diese Annahmen ermöglichen es, den Soundkartentreiber als echtzeitfähig zu betrachten.

4.5.2 Ermittlung der nötigen Zeitinformationen

Für die Ermittlung des Versatzes zwischen Bild und Ton ist es nötig, sowohl die exakte Audioposition als auch die Videoposition zu bestimmen. Die Implementierung berechnet die Positionen in Millisekunden.

Für das Bildmaterial wird in der aktuellen Implementierung der Zeitstempel anhand der Framenummer ermittelt. Jedem Frame wird eine fortlaufende Framenummer zugeordnet. Die folgende Formel beschreibt die Berechnung des Zeitstempels in ms:

$t_{video} = Framenummer * 1000/Framerate$

Wie aus dieser Formel hervorgeht, funktioniert dieser Ansatz nur für konstante Frameraten und unter der Bedingung, daß jedes Bild dabei für die Dauer von 1/Framerate lang angezeigt werden soll. Für die Nutzung von MPEG-4-Videos verpackt in AVI oder OGM ist dies zutreffend. Bei MPEG-1- und MPEG-2-Videos wird die Darstellungszeit durch eine Präsentationszeit bestimmt, welche von der Framenummer unabhängig ist.

Die Nutzung dieser Präsentationszeit anstelle der oben verwendeten Formel würde es ermöglichen, die Synchronisationskomponente auch für MPEG-1/-2-Video zu nutzen. Anzuraten ist i. allg. die Nutzung der Präsentationszeit innerhalb der Synchronisationskomponente. Die oben erwähnte Formel sollte bereits im Videokern für entsprechendes Videomaterial Verwendung finden. Gegenwärtig ist dies jedoch nicht implementiert, da es für die verfügbaren Codecs und Container-Formate unnötig ist.

Komplizierter erwies sich die Ermittlung der aktuellen Audioposition. Es genügt nicht, die Position anhand der Anzahl der zum Treiber gesendeten PCM-Samples zu ermitteln. Die dadurch ermittelten Werte berücksichtigen nicht die Größe der Puffer einerseits auf der Soundkarte andererseits auf Treiberseite. Die Folge wäre, daß bei großen Puffern unter Umständen die gemessene Ausgabezeit der wirklichen erheblich voran schreitet. Eine Ermittlung der Puffergrößen ist zwar möglich, aber zu

treiber- bzw. gerätespezifisch.

Die implementierte Komponente setzt eine Ebene höher an: Sie nutzt die Funktionalität des *Open Sound System-Layers* (OSS). Via *ioctl* wird die Anzahl der bereits von der Soundkarte ausgegebenen Bytes ermittelt. Die Berechnung der Audioposition in Millisekunden mit Hilfe der Byteanzahl erfolgt gemäß der folgenden Formel:

$$t_{audio} = Bytes_{gespielt} * 1000/Tonkanaele * Bytes_{sample} * Samplerate$$

Auch hier wird vorausgesetzt, daß bestimmte Werte während der Wiedergabe konstant bleiben: die Samplerate, die Anzahl der Tonkanäle und das Audioformat, welches die Bytes pro Sample bestimmt. Im allgemeinen ist dies jedoch der Fall. Problematischer ist ein Überlauf des Bytezählers nach rund einer Stunde Tonmaterial. Dieser führt zu falschen Positionen und wird daher von der Synchronisationskomponente kompensiert.

Aktuelle OSS-Treiber unterstützen den verwendeten ioctl-Befehl, jedoch ältere Treiber manchmal nicht korrekt [46]. Daher wird z.B. von xine ein anderer Mechanismus auf OSS-Basis genutzt [47]. In [46] werden Lösungen für andere Abstraktionsebenen wie ALSA (Advanced Linux Sound Architecture) vorgestellt. Die gegenwärtige Implementierung unterstützt nur die bereits vorgestellte Methode via ioctl. Aktuell ist dies nicht von Nachteil, da DDE nur OSS unterstützt und nur ein Treiber portiert ist.

Die Ermittlung der Audio- und Videoposition ist recht aufwendig und wird daher nur im Intervall von 75 Frames (3 Sekunden bei 25fps) durchgeführt. Die Prüfung auf Synchronität erfolgt ebenfalls im gleichen Intervall.

4.5.3 Auswertung der Zeitinformationen und Reaktionen

Die gewonnenen Zeitpositionen für Video und Audio werden zur Prüfung auf Synchronität subtrahiert. Die Differenz beschreibt folglich die Abweichung von Bild und zugehörigem Ton. Ist der Betrag dieser Differenz größer als 80ms, so sind Maßnahmen zu ergreifen um Bild und Ton zu resynchronisieren. Die Korrektur der Synchronität erfolgt prinzipiell auf Seiten der Video-Komponenten, da ein Auslassen von Audio-Samples bei der Präsentation wesentlich auffälliger ist (vgl. Abschnitt 2.1). Aufgrund dessen wird die Differenz in eine Frameanzahl umgerechnet.

Folgende Reaktionen werden in Abhängigkeit der Größe der Abweichung realisiert:

- lokales Auslassen oder Verzögern innerhalb der Synchronisationskomponente bei kleinen Abweichungen
- bei größeren Abweichungen eine Verständigung mit dem Controller

▷ **Lokales Auslassen oder Verzögern** Bei einer geringen Abweichung von Bild und Ton wird die Synchronisationskomponente selbst die Synchronität wiederherstellen. Dies geschieht im Falle, daß das Bild dem Ton vorweg läuft, durch Verzögern von einzelnen Videoframes. Dies ist einfach implementiert, indem einfach kein Bild importiert wird und stattdessen das letzte gültige Bild im shared-Memory des VScreen-Widgets verbleibt. Dann wird entsprechend gewartet (vgl. Metronome).

Das Auslassen von Frames, auch *Dropen* genannt, erfolgt, wenn das Bild dem Ton nachläuft.

Dazu werden einfach hintereinander Frames importiert, aber nicht dargestellt. Dies wird solange wiederholt, bis das gewünschte Frame vorliegt. Danach erfolgt wieder eine normale Wiedergabe.

Der Vorteil des lokalen Auslassen bzw. Verzögerns besteht darin, daß einerseits der Kommunikations-Overhead wegfällt und andererseits keine Rücksicht auf Frametypen (I-,P- oder B-Frame) genommen werden muß, da diese bereits dekodiert im RAW-Format vorliegt. Ebenfalls werden andere Komponenten nicht beeinflußt - diese können ungestört weiter arbeiten.

▷ **Smoothing** Beim Dropen bzw. dem Verzögern von mehreren Frames kann es vorkommen, daß beim Betrachten des Videos harte Sprünge zwischen Einzelbildern erkennbar sind. Diese wirken stellenweise sehr störend. Aus dem Wunsch heraus, dies zu vermeiden, wurde experimentell ein Glätten (*Smoothing*, engl.) implementiert. Dabei wird für den Fall, daß Frames auszulassen sind, die Abspielgeschwindigkeit der Bilddaten temporär beschleunigt. Wenn die Bilder dem Ton vorweglaufen, wird die Geschwindigkeit temporär entsprechend gesenkt. Damit nähert sich das Bild dem Ton wieder an, ohne Videodaten nicht zu präsentieren. Vorteilhaft ist, das dem Anwender nicht der Eindruck entsteht, daß Bilder fehlen. Durch Tests wurde ermittelt, daß es störend wirkt, wenn die Abspielgeschwindigkeit kurzzeitig um mehr als 40% verändert wird. Für diese Fälle ist es günstiger, ein wirkliches Weglassen bzw. Verzögern von Frames zu nutzen, wie es im vorigen Abschnitt geschildert wurde.

▷ **Verständigung mit dem Controller** Größere Differenzen von Video- und Audiopositionen führen dazu, daß die Synchronisationskomponente eine Mitteilung an den Controller schickt. Diese enthält die gewünschte Videoposition, welche zur Resynchronisierung führen würde. Der Controller entscheidet anhand der ihm vorliegenden Daten, ob dieser Wunsch erfüllbar ist. Beispielsweise ist zu prüfen, ob die Quelle überhaupt ein Springen innerhalb des Videos unterstützt. Wird das Springen von den beteiligten Komponenten nicht unterstützt, dann wird auf lokales Auslassen bzw. Verzögern zurückgegriffen.

Unterstützen alle nötigen Komponenten ein Springen innerhalb des Videos, wird der Controller dies veranlassen, z.B. dem Demultiplexer mitteilen, daß an die entsprechende Position gesprungen werden soll. Im Idealfall kommt irgendwann das Frame mit der gewünschten Zeit an, was die Synchronität von Bild und Ton wiederherstellt. Das Springen innerhalb des Videos bringt aber einige Probleme mit sich, die die Umsetzung dieses Konzeptes erschweren. Auf diese Probleme wird im nächsten Abschnitt eingegangen.

4.5.4 Probleme beim Springen im Video

Wurde ein Springen innerhalb des Videos veranlaßt, sei es aufgrund des Synchronisationsmechanismus oder durch den Wunsch des Benutzers, treten Probleme auf, die die Synchronität zwischen Bild und Ton gefährden. Das Hauptproblem resultiert daraus, daß nicht beliebig genau innerhalb eines Videos gesprungen werden kann.

Beim Springen ist es im Falle von 'inter'-kodiertem Videomaterial nötig, zu einem Keyframe zu springen, damit der nachgeschaltete Videokern die Daten auch richtig dekodieren kann. Auch bei kodiertem Audiomaterial tritt das Problem auf, da beispielsweise MP3-Daten auch in einer Gruppe

von 1152 Samples zu einem Frame kodiert werden. Daher können nur die Positionen zwischen zwei Frames angesprungen werden. Diese Ungenauigkeiten müssen korrigiert werden.

Zur Lösung des Problems wird die Differenz zwischen gewollter und tatsächlich erreichter Video- bzw. Audioposition mit dem aktuellen Frame mitgeschickt. Die Synchronisationskomponente korrigiert die von ihr bestimmte Video- und/oder Audioposition ggf. um diese Differenzen. Um die Synchronisationskomponente zu veranlassen, diese Korrektur vorzunehmen, wird vom Demultiplexer das angesprungene Frame mit der gewünschten Zeit, der Abweichung und aktiven *Reset-Sync-Flag* (vgl. Tabelle 11) versehen. Bei gesetztem Reset-Sync-Flag übernimmt die Synchronisationskomponente diese Daten zur weiteren Einbeziehung in die Berechnungen der Audio- bzw. Videoposition.

Ein Vorteil dieser Methode ist, daß nicht nur der Demultiplexer dieses Flag nutzen kann sondern auch beispielsweise der Videokern. Relevant ist dies z.B, wenn der Videokern eine Lücke im dekodierten Material entdeckt, die der Demultiplexer nicht wahrnehmen konnte, weil dazu die Daten erst dekodiert sein müssen. Dieser Fall kann beim Streaming von Videos auftreten.

4.5.5 Ausgleich des Uhrendrifts

Wie bereits erwähnt, driften die Uhr auf der Soundkarte und die Systemuhr möglicherweise auseinander. Da zur sinnvollen Messung der Audioposition nur die Soundkartenuhr und zur Bestimmung der Videoposition nur die Systemuhr genutzt werden kann, gibt es das Problem, daß Bild und Ton auseinanderdriften können. Daher gilt es, entweder die beiden Uhren zu synchronisieren oder den Drift bei Ausgabe mit einzukalkulieren. Letzteres ist implementiert wurden.

Wie bereits im Abschnitt 4.5.2 beschrieben, wird die Audioposition nur alle paar Sekunden ermittelt. Diese wird gespeichert. Beim erneuten Bestimmen wird die letzte Audioposition von der aktuell gemessenen subtrahiert. Die Differenz repräsentiert folglich die Soundkartenuhr.

Jedes Mal, wenn die Audioposition bestimmt wird, wird auch die Systemuhr abgefragt und gespeichert. Daher kann für die Systemuhr auch die Dauer zwischen zwei Messungen bestimmt werden. Das Verhältnis zwischen den gemessenen Audiopositionen und der Differenz von zwei Systemuhrmessungen entspricht dem Drift zwischen der Soundkartenuhr und der Systemuhr. Dieses Verhältnis geht als Faktor für die Berechnung der Videoposition im Metronome ein.

Es sei darauf hingewiesen, daß diese Messung der beiden Zeiten nicht exakt sein müssen, da es zwischen dem Bestimmen der Audioposition und dem Auslesen der Systemuhr zu unvorhersagbaren Verzögerungen kommen kann, z.B. aufgrund einer Prozeßumschaltung. Da diese Messungen kontinuierlich über einen gewissen Zeitraum, nämlich der Abspieldauer des Videos, vorgenommen werden, haben diese Verzögerungen keinen oder einen unmerklich kleinen Einfluß. Das im Anhang F dargestellte Diagramm 33 verdeutlicht dies anhand von aufgenommenen Meßwerten. Eine Optimierung des Algorithmus bezüglich der Fehlertoleranz ist durch die Verwendung eines gleitenden Durchschnitts möglich.

4.6 Benutzerinterface

Das in Abbildung 23 zu sehende Benutzerinterface wurde mit DOpE-Widgets implementiert. Dargestellt sind der Controller, der sowohl als Interface für den Player als auch für den Recorder dient, und der für die Videoausgabe relevante Teil der Synchronisationskomponente. Diese ist, wie im Abschnitt 4.5 beschrieben, mit Hilfe des VScreen-Widgets implementiert worden.

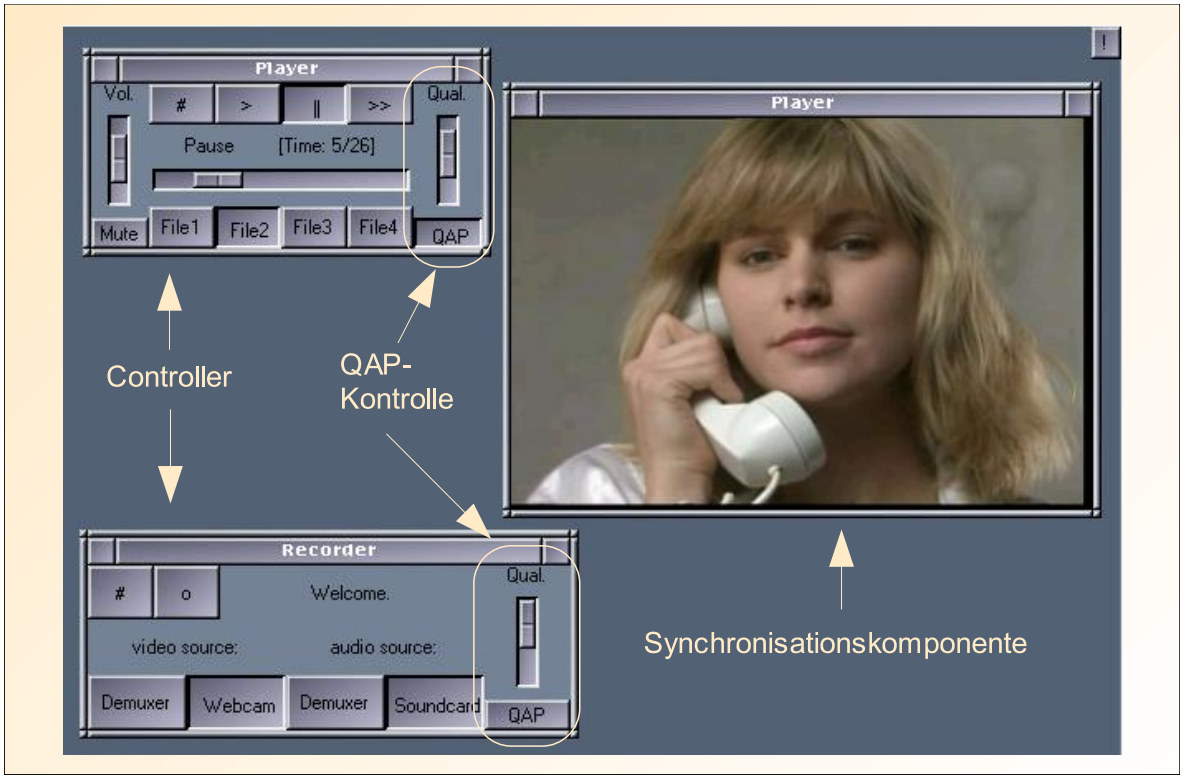

Abbildung 23: Implementiertes Benutzerinterface

Die Steuerung des Recorders erfolgt durch den Stop- und den Record-Button sowie durch Auswahl der Video- und Audioquelle. Dabei wird dem Video- bzw. Audiokern mitgeteilt, welches Import-Plugin benutzt werden soll.

Der implementierte Player unterstützt neben dem normalen Abspielen sowie Stop und Pause ebenfalls die Möglichkeit zum schnellen Vorspulen. Die Audiowiedergabe wird dabei ausgeschalten, jedoch alle Videoframes dekodiert und wiedergegeben. Über den horizontalen Scale-Widget kann zu beliebigen Positionen im Video gesprungen werden, sofern die Quelle das unterstützt. Zur Auswahl stehen maximal vier Videos oder Audiodateien, auf die gegenwärtig über das Dummy-Dateisystem zugegriffen wird. Außerdem ist es möglich, die Lautstärke für die Wiedergabe zu regeln.

Die gewünschte Qualitätsstufe kann sowohl für die Aufnahme als auch für die Wiedergabe manuell über den jeweiligen Scale-Widget eingestellt werden, oder aber die Reglung erfolgt dynamisch mit dem Konzept der Qualitätsstufen-Anpassung (QAP-Button).

4.7 Zusammenfassung

Die im Kapitel 3 vorgestellten Modelle zur Aufnahme sowie Wiedergabe wurden nur unwesentlich verändert implementiert. Dabei flossen die in [1] erstellte Software sowie die Erkenntnisse des Beleges mit ein. Im folgenden Kapitel gilt es die Leistung der Implementierung sowie der vorgestellten Modelle zu erörtern. Auf die Bewertung der implementierten Synchronisationskomponente wird ebenfalls im nächsten Kapitel eingegangen.

5 Leistungsbewertung

5.1 Bewertung der Synchronisationskomponente

Eine objektive Bewertung der Synchronität anhand der Präsentation von Bild und Ton ist ohne entsprechende Testvideos und Meßgeräte nicht möglich. Um den subjektiven Eindruck, daß Video und Audio synchron wiedergegeben werden, zu untermauern, wurden Messungen der Audio- und Videopositionsabweichungen für zwei Testvideos durchgeführt. Dabei wird die Differenz zwischen der Ton- und Bildposition anhand der im Abschnitt 4.5.2 beschriebenen Formeln ermittelt und grafisch dargestellt. Die Abbildung 24 zeigt die aufgenommenen Meßwerte in Form von Diagrammen.

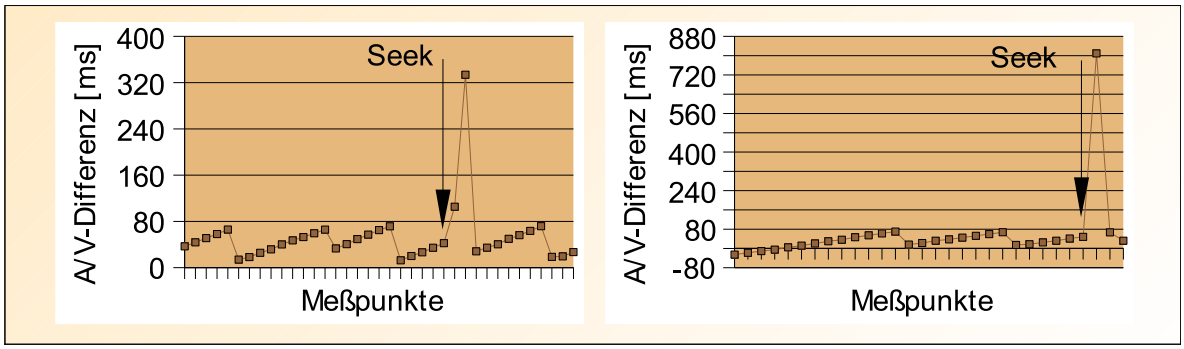

Abbildung 24: Audio-Video-Abweichung bei der Wiedergabe für zwei Videos

Dabei ist zu sehen, daß der für die lippensynchrone Wiedergabe kritische Grenzwert von 80ms nur nach dem Springen in den Videos überschritten wird. Nach den Sprüngen sorgt die Synchronisationskomponente wieder für den Ausgleich der beiden Positionen und für anschließende Lippensynchronität. Um die Synchronität bei der Wiedergabe zu erhalten, wurden Frames ausgelassen, wenn die Audio-Video-Differenz größer als 65ms war. Damit wird versucht, nicht erst bis zum bemerkbaren Verlust der Synchronität zu warten, sondern schon vorher zu reagieren und die Synchronität zu erhalten.

Das stetige Steigen der Differenz von Bild und Ton ohne Eingriff der Synchronisationskomponente liegt daran, daß zur Bewertung der Komponente die Korrektur von System- und Soundkartenuhr ausgeschalten wurde. Dies beeinflußt die Ausagefähigkeit der Meßergebnisse jedoch nicht. Ansonsten wäre der mögliche Verlust der Lippensynchronität erst später bei Wiedergabe relevant geworden, was die Messungen unnötig erschwert hätte. Die unterschiedlichen Startdifferenzen sind mit der Nebenläufigkeit der Bild- bzw. Tonverarbeitung und damit unterschiedlichen Verarbeitungsbeginnen zu begründen. Die Synchronisationskomponente korrigiert auch dies.

5.2 Simulation der Qualitätsstufen-Anpassung

Es ist nicht möglich, das implementierte Gesamtsystem in bezug auf das Konzept der Qualitätsstufen-Anpassung sowie die damit verbundene Integration in das DROPS Scheduling Framework vollständig zu beurteilen. Der Grund ist, daß zum gegenwärtigen Zeitpunkt der Scheduler des Quality-Assuring Schedulings nicht verfügbar ist. Wie bereits erwähnt, ist die Einbindung in das Scheduling-Schema damit nicht möglich und somit auch keine abschließende Bewertung. Um Aussagen über die zu erwartende Leistungsfähigkeit von Qualitätsstufen-Anpassung in Kombination mit Quality-Assuring

Scheduling machen zu können, wurden Simulationen durchgeführt. Diese arbeiteten nach folgendem Prinzip: Ein Prozeß füllt einen Puffer von vorgegebener Größe mit Videoframes mit einer konstanten Frequenz, wobei ein Pufferelement einem Frame entspricht. Dieses Vorgehen entspricht z.B. dem bei der Aufnahme von einer Webcam. Ist der Puffer voll, werden Frames verworfen. Der Verbraucher beginnt mit der Entnahme aus dem Puffer, sobald dieser einen gewissen Füllstand (Preload) erreicht hat. Jedem entnommenen Frame wird eine Verarbeitungszeit zugeordnet, die aus real gemessenen Werten stammen (vgl. Anhang F). Dabei werden die Meßdaten der gewünschten Qualitätsstufe genutzt. Zuerst wurden Messungen ohne Qualitätsstufen-Anpassung durchgeführt. Dabei wird in der gesamten Simulation mit derselben wählbaren Qualitätsstufe gearbeitet. Demgegenüber wurde für die weiteren Messungen die Qualitätsstufen-Anpassung aktiviert. Bei der Simulation wurde zur QAP-Steuerung der Pufferfüllstand als Maß für die Systemlast verwendet. Ab einem oberen Schwellwert wird die Qualität gesenkt und bei Unterschreiten eines unteren Schwellwertes wird die Qualität erhöht.

▷ **Erkenntnisse bzgl. QAS** Wie im Abschnitt 3.6 bereits vorgestellt, ist es angedacht, die En- bzw. Dekodierung der Video- und Audiodaten innerhalb eines optionalen Teils auszuführen. Eine einfache Einbindung in das DROPS Scheduling-Schema könnte daher folgende sein: Wird die Bearbeitung des Frames vor dem Ablauf der Reservierungszeit fertig, wird mit next_period() auf den nächsten Periodenbeginn gewartet. Wird dagegen ein Frame innerhalb der Reservierungszeit nicht vollständig verarbeitet, dann wird in der folgenden zuerst dieses und dann das nächste bearbeitet. Pro Periode wird maximal ein Frame aus dem Puffer entnommen und bearbeitet. Dies kann nicht funktionieren, da der Pufferfüllstand bei dieser Methode ausgehend vom Preload nur monoton steigen kann, da mindestens ein neues Frame in den Puffer gelangt und maximal eines verbraucht wird. Ausgenommen sind Framedrops und das Ende des Videostroms. Damit liefe mit steigender Aufnahmedauer die visuelle Qualität gegen null, da der Puffer zunehmend voller würde und damit irgendwann über den oberen Schwellwert läge.

Ein anderer Vorschlag ist, den Verbraucher während der kompletten Reservierungszeit arbeiten zu lassen und nicht nach einem bearbeiteten Frame auf den Beginn der nächsten Periode zu warten. Dies würde zu sehr guten Ergebnissen bzgl. Framedrops führen, da die komplette reservierte Zeit zur Verarbeitung verfügbar wäre. Allerdings werden die optionalen Teile im QAS-Modell verwendet, um mit Überlastsituationen umgehen zu können. Dabei wird davon ausgegangen, daß optionale Teile aufgrund ihrer Verteilungsfunktion auch manchmal kürzere Verarbeitungszeiten als die jeweilige Reservierungszeit in ihrer Periode in Anspruch nehmen und dann mit next_period() ihre Verarbeitung bis zum nächsten Periodenbeginn einstellen. Ein anderer Job kann die nicht verbrauchte Zeit nutzen. Damit verstößt aber der Vorschlag, die komplette Reservierungszeit arbeiten zu lassen, gegen diese Voraussetzung für das Funktionieren des Quality-Assuring Schedulings.

Ein Kompromiß der zwei vorgestellten Methoden ist, daß im Normalfall analog zum ersten Vorschlag nur maximal ein Frame pro Periode entnommen werden kann. Wird aber festgestellt, daß innerhalb einer vorherigen Periode kein Frame fertig kodiert wurde, dann können mehrere Frames entnommen und verarbeitet werden, solange die Framenummer die Periodennummer nicht übersteigt. Beispielsweise wird in der 4. Periode maximal bis zum 4. Frame gearbeitet. Ist aber nach der Verarbeitung des 4. Frames in der 4. Periode die Reservierungszeit noch nicht aufgebraucht, dann wird gewartet und nicht mit dem 5. Frame angefangen. Damit gelingt es zumindest, daß nicht in jeder Periode die komplette Reservierungszeit voll ausgenutzt wird. Allerdings wird die Chance gegeben, den

Pufferfüllstand zu reduzieren. Hierbei ist zu beachten, daß der Pufferfüllstand, außer beim Ende des Videostroms, nur maximal ein Frame unter den Preload sinkt. Daher wurde er bei der Simulation auf ein Frame eingestellt.

Die Schwellwerte für die Steuerung der Qualitätsstufen-Anpassung wurden wie folgt gewählt: Bei 10% Pufferfüllstand wurde die Qualität erhöht, ab 50% gesenkt. Diese Schwellwerte erwiesen sich als recht günstiger Kompromiß zwischen hoher Qualität und Rechenzeit pro Periode. Die Überprüfung des Pufferfüllstandes sowie die Veränderung der aktuellen Qualitätsstufe wurden jeweils nach einer festgelegten Frameanzahl (Checkpunkt) durchgeführt. Besonders relevant für die Auswertung ist die Anzahl der Framedrops. Dabei wird erwartet, daß bei aktivierter Qualitätsstufen-Anpassung die Anzahl der Framedrops sinkt.

▷ **Ergebnisse** Die Tabelle 12 zeigt eine Übersicht der durch die Simulation gewonnen Erkenntnisse. Die Reservierungszeiten wurden dabei aus der Verteilungsfunktion der entsprechenden Qualitätsstufe

QAS-Qualität für opt. Teil	*Qualitäts-stufe*	*Reservie-rungszeit*	*Framedrops*		*⌀-Qualität (visuelle)*	
			ohne QAP	*mit QAP*	*ohne QAP*	*mit QAP*
65%	Q0	13,5ms	4	7	0	0,09
80%	Q1	20ms	0	1	1	1,28
50%	Q2	24ms	15	0	2	2,32
40%	Q3	27ms	15	0	3	3,15
65%	Q4	30ms	10	0	4	3,74
Periode = 1/29.97fps , Checkpunkt = 5, Puffergröße = 10 Preload = 1, Schwellwerte: 10% und 50%, 780 Frames						

Tabelle 12: Übersicht der Simulationsergebnisse

(Q0-Q4) anhand der vorgestellten QAS-Methodik ermittelt. Wie erwartet, sank die Anzahl der Framedrops bei aktivierter Qualitätsstufen-Anpassung mit Ausnahme der zwei niedrigsten Qualitätsstufen. Dies ist damit zu begründen, daß mit aktiver Qualitätsstufen-Anpassung am Anfang die Qualität erhöht wurde, da anfangs der Pufferfüllstand sehr niedrig ist. Dies begründet auch die leicht höhere Duchschnittsqualität bei Q0. Zu beachten ist bei Q0 außerdem, daß die Qualität nicht mehr gesenkt werden kann, um Rechenzeit zu sparen. Die durchschnittliche visuelle Qualität blieb bei deaktivierten QAP pro Meßreihe konstant. Mit Qualitätsstufen-Anpassung lag diese mit Ausnahme von Q4 jeweils über der visuellen Qualität ohne QAP. Bei Q4 kann dies nicht passieren, da die vierte Qualitätsstufe zugleich die höchste ist. Das temporäre Senken der Qualitätsstufe zur Vermeidung von Framedrops führt damit zwangsläufig zu einer niedrigeren Duchschnittsqualität. Bemerkenswert ist, daß für Q2 und Q3 weniger Framedrops mit QAP auftreten und trotzdem die durchschnittliche visuelle Qualität höher als ohne QAP ist.

▷ **Einfluß der Puffergröße** Um den Einfluß der Puffergröße auf die Anzahl der Framedrops sowie die mittlere visuelle Qualität zu ermitteln, wurden mit der dritten Qualitätsstufe Q3 und mit 27ms Reservierungszeit pro Periode sowie einer Puffergröße von 10 Frames Messungen mit unterschiedlichen Puffergrößen durchgeführt. Die Diagramme in Abbildung 25 zeigen die Ergebnisse. Die durchschnittliche visuelle Qualität ist mit QAP bei allen Messungen höher als ohne Qualitätsstufen-Anpassung, da temporär die Qualitätsstufe erhöht werden konnte, wenn der Pufferfüllstand unter 10% war. Mit

Ausnahme einer Puffergröße von zwei Frames ist festzustellen, daß die Anzahl der Framedrops sinkt, wenn QAP aktiv ist - und dies trotz höherer mittlerer Qualität. Daß bei maximal zwei Frames im Puffer mehr Framedrops auftreten, ist damit erklärbar, daß die gewählten Schwellwerte in diesem speziellen Fall unsinnig sind: Wenn ein Frame im Puffer ist, dann bleibt die Qualitätsstufe unverändert, bei zwei wird sie gesenkt, bei keinem Frame wird sie erhöht. Der Wert von zwei Frames pro Puffer tritt in der Simulation nur zu 5% auf, allerdings der Wert von null Frames im Puffer zu 29%. Somit wird häufiger gesteigert als gesenkt. Erwähnenswert ist, daß es ohne aktiver Qualitätsstufen-Anpassung mehr als der doppelten Puffergröße im Gegensatz zu aktiver QAP bedarf, um ohne Framedrops auszukommen.

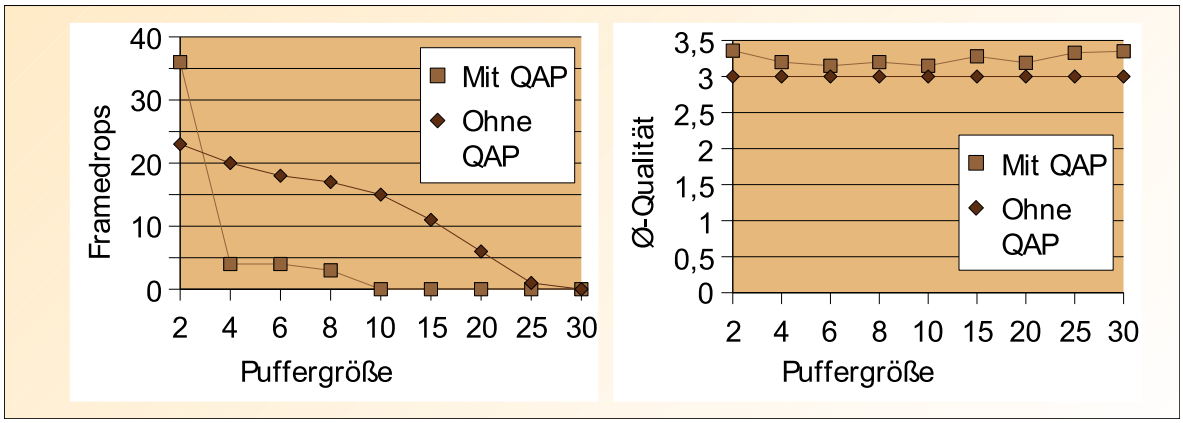

Abbildung 25: Einfluß der Puffergröße bei der Simulation

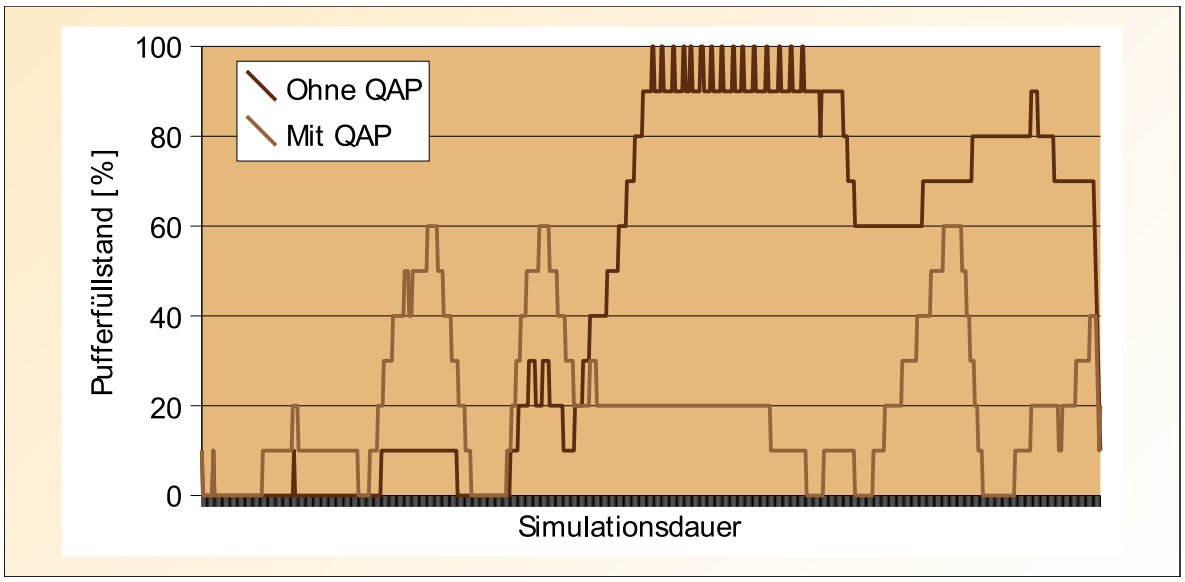

Abbildung 26: Pufferverlauf bei der Simulation

Die Abbildung 26 zeigt den Verlauf des Pufferfüllstandes. Dabei ist erkennbar, daß ohne QAP der Pufferfüllstand temporär steigt und es zu Framedrops kommt. Im Testvideo entspricht dies der Stelle mit erhöhter Bewegung (Football-Spiel, siehe Anhang F). Mit Qualitätsstufen-Anpassung ist der maximale Pufferfüllstand 60%, da bei 50% Füllstand die Qualitätsstufe gesenkt wird und dies

die gewünschte Wirkung zeigt. Bei beiden Messungen war im übrigen die durchschnittlich genutzte Rechenzeit pro Periode bei über 99%.

5.3 Zusammenfassung

Die Simulation des Konzeptes der Qualitätsstufen-Anpassung sowie die angedachte Integration in das DROPS Scheduling Framework zeigen durchaus die erhoffte Wirkung: Die Wahrscheinlichkeit für Framedrops ist mit Qualitätsstufen-Anpassung i. allg. geringer als ohne QAP. Daß im Vergleich zu den Messungen ohne Qualitätsstufen-Anpassung die Puffergröße kleiner gewählt werden kann, ohne daß Framedrops auftreten, ist positiv zu bewerten.

Ebenfalls positiv zu beurteilen ist, daß sich auch die durchschnittlichen visuellen Qualitäten auf ähnlichem Niveau bewegen. Dies sollte jedoch nicht überbewertet werden, da gerade bei schwierigem Videomaterial, nämlich den bewegten Szenen, die visuelle Qualität bei Verwendung der Qualitätsstufen-Anpassung gesenkt wird. Bei leichter zu kodierendem Material wird jedoch die Qualität erhöht. Ob dies vom Anwender vorteilhaft oder negativ empfunden wird, hängt vom verwendeten Videomaterial ab. Bei einigen Filmen sind Detailreichtum und qualitativ höherwertige Bilder auch in unbewegten Szenen positiv, dabei ist beispielsweise an ruhige Landschaftsaufnahmen zu denken.

Eine weitere Optimierung der Parameter z.B. von Schwellwerten und Checkpunkten könnte durchaus noch bessere Ergebnisse bzgl. der Anzahl der Framedrops und der mittleren visuellen Qualität liefern. Diese würde aber die Qualitätsstufen-Anpassung weiter auf das spezifische Testvideo optimieren. Angesichts der Tatsache, daß die Verteilungsfunktionen und damit die Meßwerte für andere Videos im geplanten Einsatz als Videorecorder videospezifisch und meist im voraus unbekannt sind, ist der Erfolg der Optimierung zu bezweifeln.

6 Fazit und Ausblick

Die Implementierung der entworfenen Modelle bedeutet einen wesentlichen Fortschritt zur Erreichung des Projektzieles Videorecorder. Die Zusammenarbeit des Konzeptes der Qualitätsstufen-Anpassung mit dem Quality-Assuring Scheduling konnte zumindest in der Simulation überzeugen und die erwartete Tendenz zu weniger Framedrops gegenüber herkömmlichen Ansätzen bestätigen. Der für einen Videorecorder wichtige Teil zur Wiedergabe der aufgezeichneten Videos - der Videoplayer - wurde ebenfalls entworfen und implementiert. Die Synchronität von Bild und Ton wird dabei hergestellt und bleibt bei der Präsentation erhalten.

Zur endgültigen Fertigstellung des echtzeitfähigen Videorecorders und -players bedarf es noch weiterer Schritte. Dazu zählen insbesondere

- die Einbindung des Systems in das DROPS Scheduling-Schema bei Verfügbarkeit des QAS-Schedulers sowie
- die Nutzung eines echtzeitfähigen Dateisystems, sobald dies vorhanden ist.

Außerdem ist die Nutzung weiterer Videoquellen wie beispielsweise TV-Karten erstrebenswert, um den möglichen Einsatzbereich des Videorecorders zu vergrößern. Eine Überprüfung des USB-Stacks sowie des Import-Plugins für die Webcam ist anzuraten, da auf mehreren Testsystemen die Kamera größtenteils nicht richtig erkannt und initialisiert wurde und somit die korrekte Funktionsweise nicht überprüft werden konnte. Desweiteren empfiehlt sich die Erstellung und Verwendung eines Sound-Servers, da damit Zugriffskonflikte beim Benutzen der Soundkarte vermieden werden könnten.

Einstellungsmöglichkeiten zur Erhöhung bzw. Senkung der visuellen oder akustischen Qualität bei Video- und Audio-Codecs sind nur dann sinnvoll, wenn die Verarbeitungsgeschwindigkeit ebenfalls fällt bzw. steigt. Es wäre unsinnig, mit langsamer Verarbeitung eine niedrige Qualität zu erhalten, wenn die schnellere Verarbeitung höhere visuelle oder akustische Qualität verspricht. Dadurch behält das entwickelte Konzept der Qualitätsstufen-Anpassung nach derzeitigem Kenntnisstand auch für zukünftige Entwicklungen seine Gültigkeit.

A Glossar

AAC	Advanced Audio Codeing, Audio-Codec
Admission	Entscheidung, ob ein Job zugelassen wird
API	Application Programming Interface
ASF	Advanced Streaming Format, Container-Format
AVI	Audio Video Interleave, Container-Format
Bitrate	Anzahl der Bits pro Zeit
Bitratengewinn	Einsparungen der Bitrate meist durch höhere Kompressionsrate
CBR	constant bitrate, engl. konstante Bitrate
Codec	Algorithmus zum en- und dekodieren
Container	enthält Bild und Ton unabhängig von Codecs
CVS	Concurrent Versions System, Versionskontrolle zur Quellcode-Verwaltung
Deblocking	glätten von Blockartefakten
Deringing	kaschieren und glätten von Ringartefakten
DVB	Digital Video Broadcasting
Downmixing	Reduktion der Audiokanäle (z.B. 5 Kanäle zu stereo) oder der Samplerate
EPG	elektronischer Programmführer
FourCC	vier Buchstaben als Codec-Kennung
FPS	Frames pro Sekunde
Framedrops	Auslassen von Frames
Framerate	FPS
Frames	Einzelbilder eines Videos
GCC	GNU project C und C++ Compiler
GMC	Global Motion Compensation, Bewegungsabschätzung für gesamte Bildteile und nicht nur auf Macroblockbasis
GOP	Group of Pictures, Gruppe von Frames zwischen zwei keyframes
GPL	GNU General Public License
I/O	Input/Output, engl. Eingabe/Ausgabe
'inter'-Kodierung	Kodierung der Unterschiede zu den Referenzbildern (P-/B-Frames)
ioctl	Funktion zur Manipulation von Treiber- bzw. Hardwareparametern
ISO/IEC	International Standardization Organization / International Electrotechnical Commission
Job	ein Aufgabenteil eines Tasks
keyframe	vollständiges Einzelbild, dient meist als Referenz für andere Bilder
LAME	"LAME Ain't an Mp3 Encoder"
LGPL	GNU Library General Public License
Matroska	offenes und streamingtaugliches Container-Format
Motion Estimation	engl. Bewegungsschätzung, Abschätzung der Bewegungen innerhalb von Bildfolgen auf Makroblockbasis
mpg123	kommandozeilenbasierte MPEG-Audio-Player für MPEG-1 und -2
Ogg Vorbis	komplett freier, nicht patentierter Audio-Codec
Ogg Media	freies und streamingtaugliches Container-Format

OSS	Open-Sound-System, API zum Zugriff auf Soundkarten
PCM	Pulse Code Modulation, PCM-Samples: digitalisierte Audiodaten
Plugin	Softwarekomponente zur Ergänzung der Funktionalität, wird dem Programm hinzugefügt
Processing	verarbeiten von Daten z.B. bei Bildern: Spieglung, Drehen etc.
Postprocessing	Nachbearbeitung von Daten z.B. von Frames
QAP	Qualitätsstufen-Anpassung
QoS	Quality of Service, engl. Güte eines Dienstes
QPel	genauere Motion Estimation bis zu Viertel-Pixel
RAW	unkodierte unkomprimierte Daten
RGB	Red Green Blue, engl. Rot Grün Blau. Farbmodell, bei dem Farben aus der Rot-, Grün- und Blauintensität additiv zusammengesetzt sind
RQF	Rechenzeit-Qualitäts-Funktion, Abhängigkeit der Verarbeitungszeit von der gewünschten akustischen oder visuellen Qualität eines Codecs
Samplerate	Anzahl der Abtastungen des Audiosignals pro Zeiteinheit (meist in 1/s)
Schedule	Ablaufplan, Zuordnung von Jobs zu Betriebsmitteln
Scheduling	Einplanung, Konkretes Binden von Jobs an Betriebsmitteln
Slices	bei H.264/AVC: Bildbereiche, unabhängig von anderen kodierbar
Smoothing	temporäre Erhöhung oder Senkung der Abspielgeschwindigkeit zur Vermeidung von Framedrops
sRQF	spezifische Rechenzeit-Qualitäts-Funktion eines Codecs
Streaming	kontinuierliche Übertragung ohne Zwischenspeicherung
Task	durch Computerverarbeitung berechenbare Aufgabe
Transkodierer	Umwandler von kodierten Daten in anderes, meist kodiertes Datenformat
UYUV	wie YUV, Chrominanzabtastung jedoch nur in jeder 2. Zeile
VCEQ	Video Coding Experts Group
VBR	variable bitrate, engl. veränderlicher Bitrate
Video4Linux	Video for Linux, API zum Zugriff auf bildverarbeitende Hardware
Videos	enstpr. Film, also Kombination von Bild und Ton
XML	Extensible Markup Language, Metaauszeichnungssprache
YUV	Farbmodell, bei dem die Farben eines Pixels aus Luminanz(Y) und Chrominanz(U,V) zusammengesetzt sind
YUV420	wie YUV, Chrominanzabtastung jedoch nur in jeder 2. Zeile und 2. Spalte

B Definition der Qualitätsstufen für XviD und LAME

▷ **Definition der Qualitätsstufen für XviD beim Enkodieren:** Details und Beschreibungen der Flags sind in [48] zu finden.

Qualitätsstufe	*Codec-Flags für Bewegungsanalyse*	*generelle Codec-Flags*
Q0	PMV_EARLYSTOP16	XVID_H263QUANT |
Q1	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16	XVID_H263QUANT | XVID_HALFPEL
Q2	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16	XVID_H263QUANT | XVID_HALFPEL | XVID_INTER4V
Q3	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16 PMV_EARLYSTOP8 | PMV_HALFPELREFINE8	XVID_H263QUANT | XVID_HALFPEL | XVID_INTER4V
Q4	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16 PMV_EXTSEARCH16 | PMV_USESQUARES16 | PMV_EARLYSTOP8 | PMV_HALFPELREFINE8	XVID_H263QUANT | XVID_HALFPEL | XVID_INTER4V

▷ **Definition der Qualitätsstufen für XviD beim Dekodieren:** Genauere Erklärungen sind in der Dokumentation zu MPlayer [41] zu finden.

Qualitätsstufe	*Flags für libpostprocess*	*kurze Erklärung*
Q0	-	-
Q1	h1:a,v1:a	schnelles horizontales und vertiakales Deblocking
Q2	fa	wie 1, dazu: Deringing, automatische Kontrast- und Helligkeitskorrektur, sowie Reduktion von Störungen
Q3	de	wie 2, jedoch normales (genaueres) Deblocking

▷ **Definition der Qualitätsstufen für LAME beim Enkodieren:** Details über den Qualitätsparameter sind im Quellcode von LAME zu entnehmen. Teilweise sind Qualitätsparameter identisch (z.B. 8=7).

Qualitätsstufe	*entspr. LAME 3.70 Qualitätsparameter*
Q0	8
Q1	5
Q2	3
Q3	1

C Verwendete Hard- und Software

▷ **verwendete Entwicklungs- und Testhardware:**
CPU: AMD Athlon XP 1800+ unterstützt MMX, MMX2, SSE, 3DNow und 3DNow-ext
RAM: 256MB DDR-RAM
Grafikkarte: Nvidia Geforce 2
Soundkare: Soundblaster 16 PCI (ES1371)

▷ **verwendete Software:**

- Entwicklungsumgebung: GNU gcc 2.96, L4Env, DOpE, DDE, DICE
- die implementierte Software nutzt nach Möglichkeit spezielle CPU-Befehlssätze aus
- Auszug der verwendeten und portierten Bibliotheken
  - für Video: XviD Version 0.91, libpostprocess aus MPlayer 0.90, libavcodec 0.4.6
  - für Audio: mpg123 vom Okt. 2002, LAME 3.70
  - für Container: libogg 1.0, avilib 20030518, ogmlib
  - sonstige: dietlibc 0.21
- Teile folgender Software wurden modifiziert und verwendet:
  - dsi_staticfs, DOpE (alle Teil des DROPS-Projekts)
  - Transcode 0.6.4, MPlayer 0.90, xine-lib1-beta8, ogmtools 0.98 bis 1.03 (alle freie Software, GPL)
- zur Erstellung der Testvideos: Avisynth und diverse Filter, Virtualdub, XviD

D Unterstüzte Audio-, Video- und Dateiformate

▷ **unterstütze Dateiformate:**

Dateiformat	*Bemerkung*
OGM	realisiert durch ogmlib und libogg
AVI	realisiert durch avilib, nur Lesen unterstützt
MP3	einfache Aneinandereihung von MP3-Frames, nur Lesen unterstützt

▷ **unterstützte Videocodecs:** Auszugsweise auch aus [45] entnommen. Alle durch libavcodec implemtierten Codecs außer MSMPEG4 sind ungetestet. Ein Enkodieren ist momentan nur bei XviD möglich, da für libavcodec nicht das passende Codec-Plugin implementiert wurde.

Videocodec	*Codec*	*Enkodieren*	*Dekodieren*
MPEG-4	XviD 0.91	x	x
MPEG-1	libavcodec	(*)	(x)
MPEG-2	libavcodec	-	(x)
MSMPEG4 V1/V2/V3 aka DivX 3 aka Divx:)	libavcodec	(*)	x
WMV7	libavcodec	(*)	(x)
WMV8	libavcodec	(*)	(x)
H263(+)	libavcodec	(*)	(x)
MJPEG	libavcodec	(*)	(x)
DV	libavcodec	-	(x)
sowie einige nicht aufgeführte durch libavcodec.			

x - funktioniert ; (x) - funktioniert wahrscheinlich, aber kein Import-Plugin
(*) - ungetestet mangels Codec-Plugin ; - nicht vom Codec untersützt

▷ **unterstützte Audiocodecs:** Auszugsweise auch aus [45] entnommen. Alle durch libavcodec implemtierten Codecs sind ungetestet.

Audiocodec	*Codec*	*Enkodieren*	*Dekodieren*
MP3	mpg123	-	x
MP3	LAME	x	-
WMA V1/V2	libavcodec	-	(*)
WMV8	libavcodec	-	(*)
sowie einige nicht aufgeführte durch libavcodec.			

x - funktioniert ; (*) - ungetestet mangels Codec-Plugin
; - nicht vom Codec untersützt

E Meßanordnung

▷ **Allgemeines:**

- Verwendung der im Anhang C genannten Hard- und Software
- alle Messungen erfolgten unter DROPS mit der portierten und implementierten Software

▷ **Testvideos:**

- von [49] zusammengestellt, wie in [11] beschrieben, jedoch beim Enkodieren der Testvideos mit XviD keine B-Frames und keine GMC verwendet.
- Format: XviD kodiert bzw. RAW UYUV

- Messungen mit drei Testvideos: 176x120/260 , 352x240/780 und 720x486/780 [Auflösung/Frameanzahl]
- die Testvideos bilden einen vernüftigen Schnitt aus ruhigen, bewegten und detailreichen Szenen [11]:

Abbildung 27: Ausschnitte aus den Testvideos

▷ **Audiotests:**

- Format: MP3 bzw. PCM, Stereo, 16Bit, 44100hz
- Messungen mit drei Teststücken:
  - ”Hardsequenzer - The Sound Transformation” (MP3: joint stereo, 44100hz, 160kBit/s)
  - ”U2 - Pride” (MP3: joint stereo, 44100hz, 128kBit/s)
  - ”Die Fantastischen Vier - Thomas und die Frauen” (MP3: joint stereo, 44100hz, 160kBit/s)

▷ **Qualitätsstufen:**

- wie im Anhang B angegeben

F Meßergebnisse für XviD, LAME und mpg123

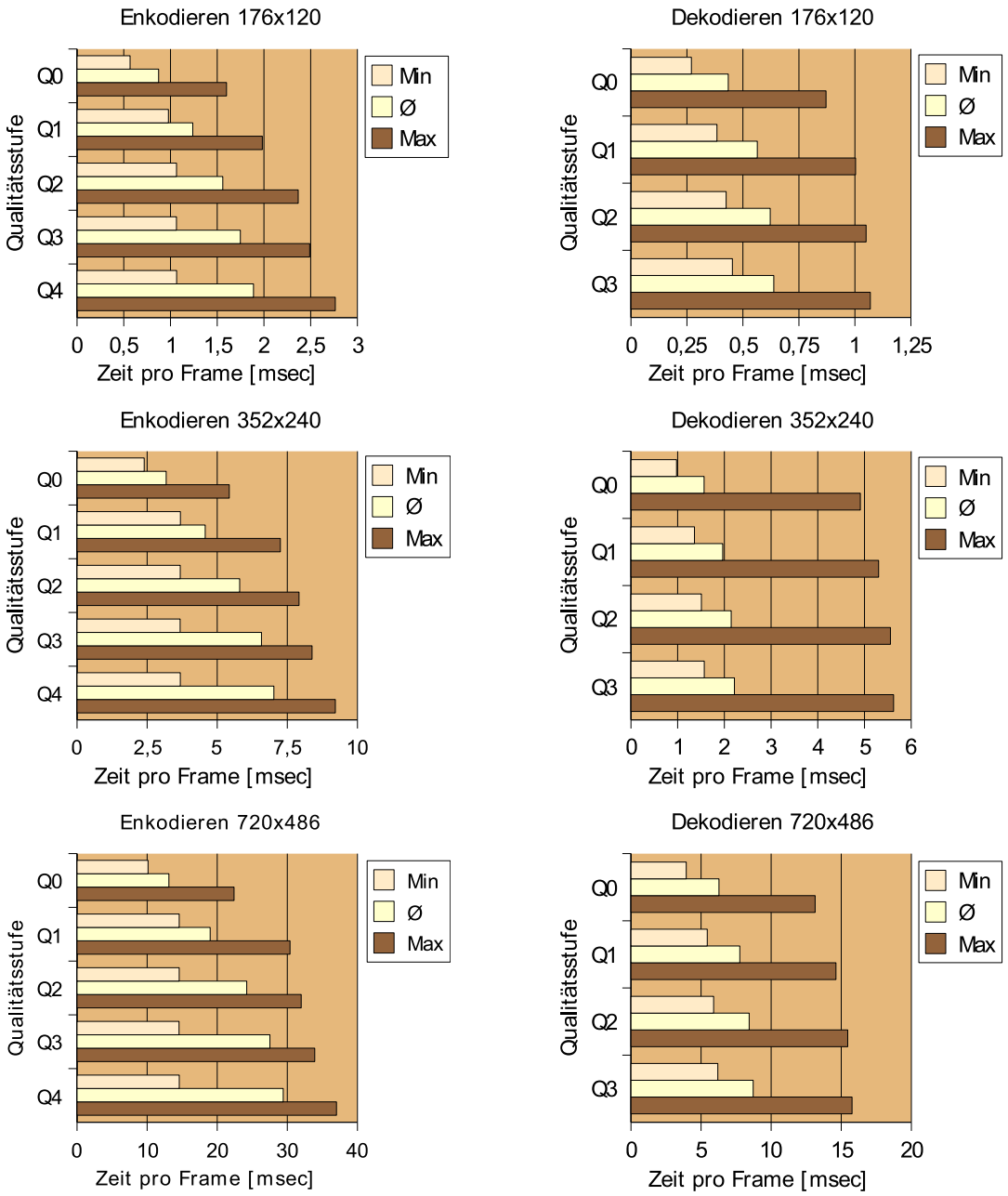

Abbildung 28: Meßwerte für XviD: Übersicht für das Enkodieren und Dekodieren

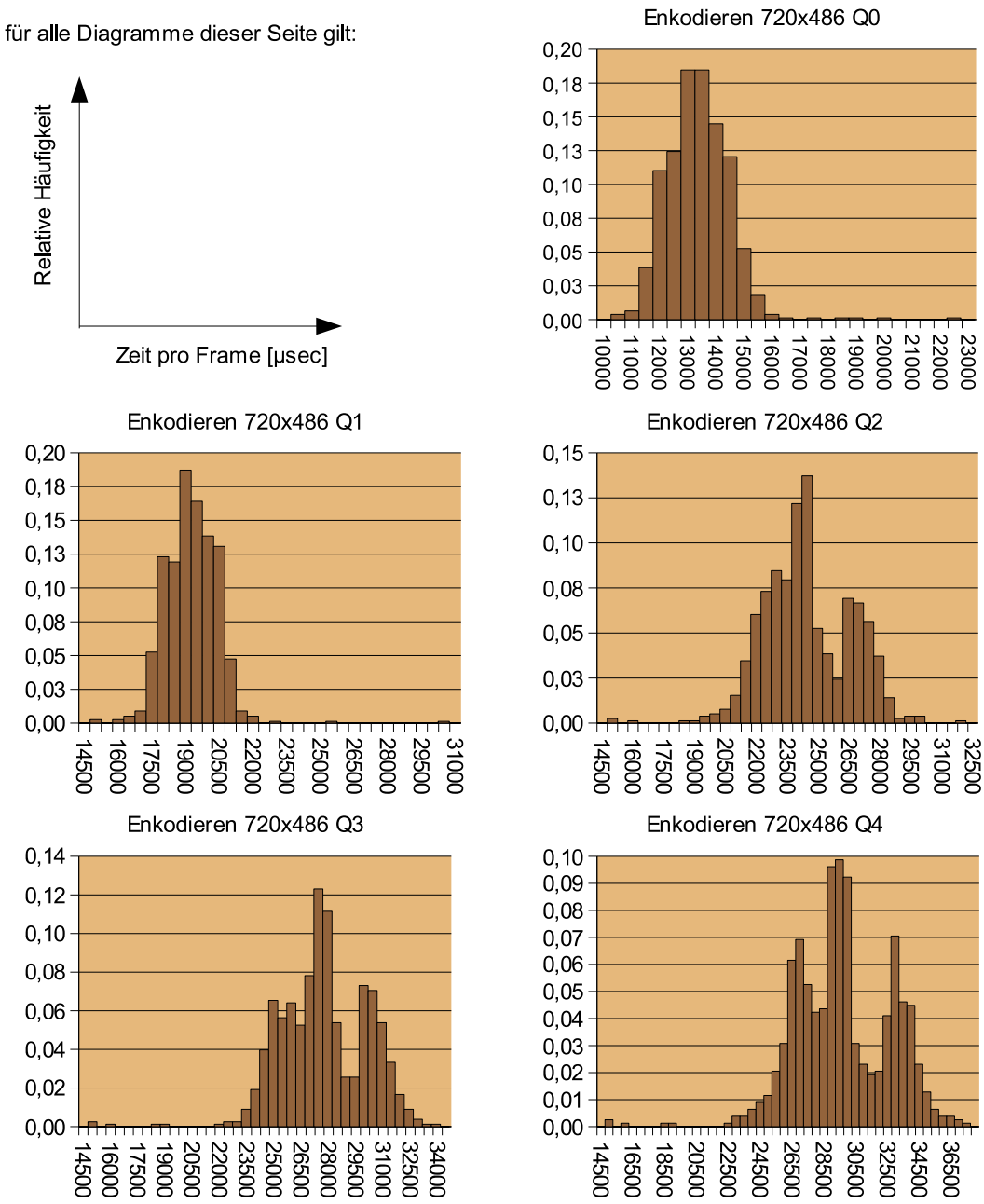

Abbildung 29: Meßwerte für XviD: Häufigkeitsverteilungen für das Enkodieren

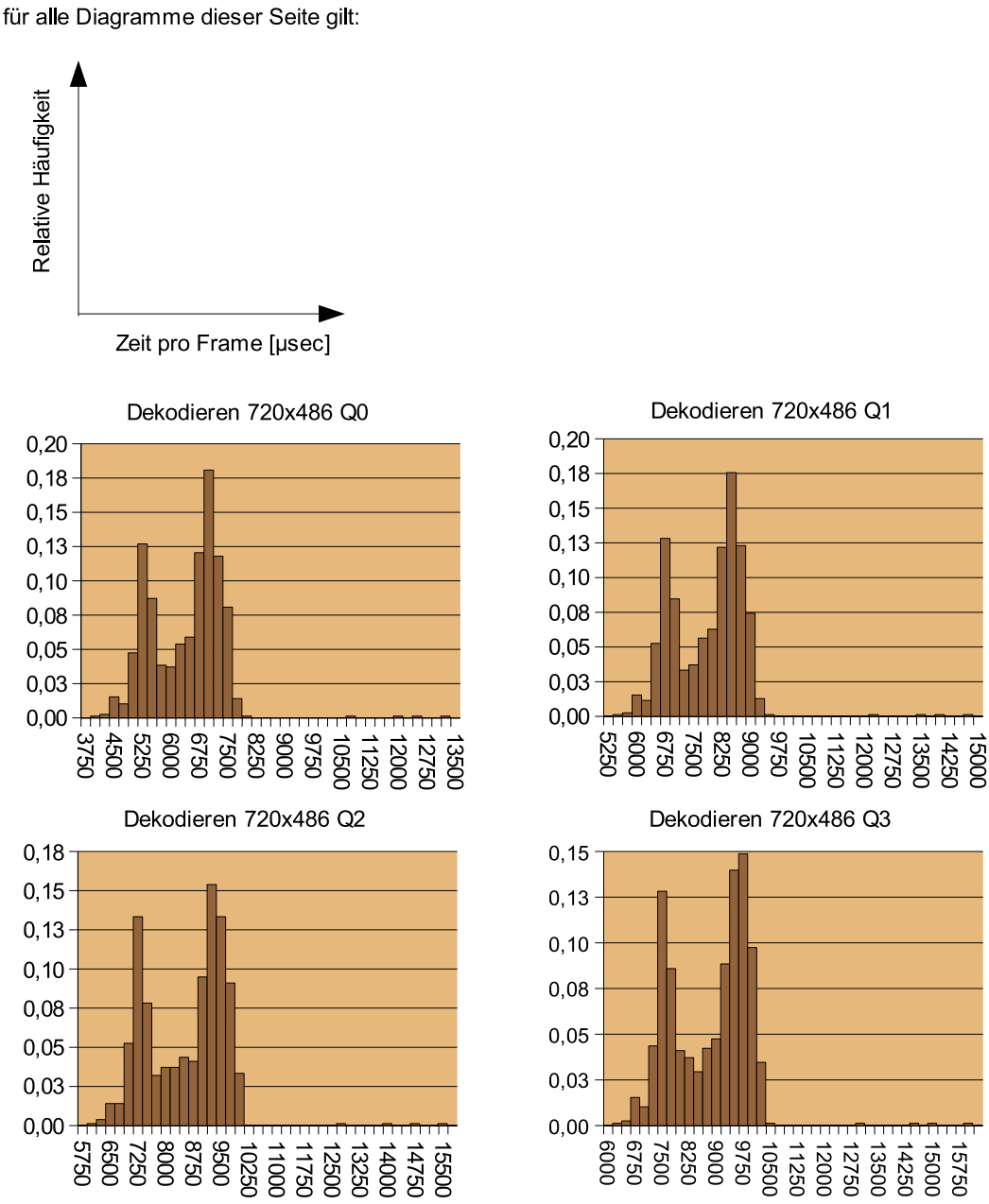

Abbildung 30: Meßwerte für XviD: Häufigkeitsverteilungen für das Dekodieren

für alle Diagramme dieser Seite gilt:

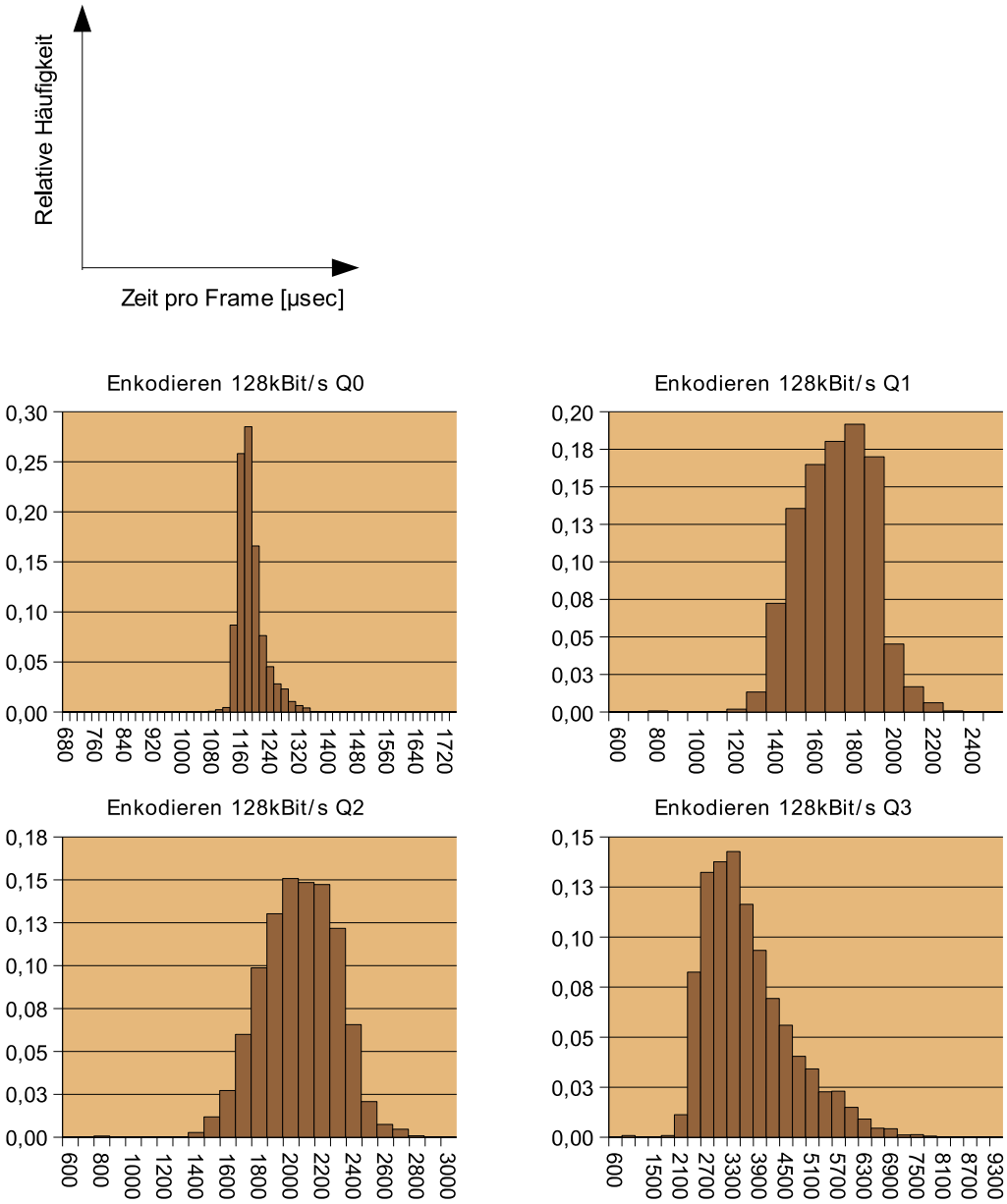

Abbildung 31: Meßwerte für LAME: Häufigkeitsverteilungen für das Enkodieren

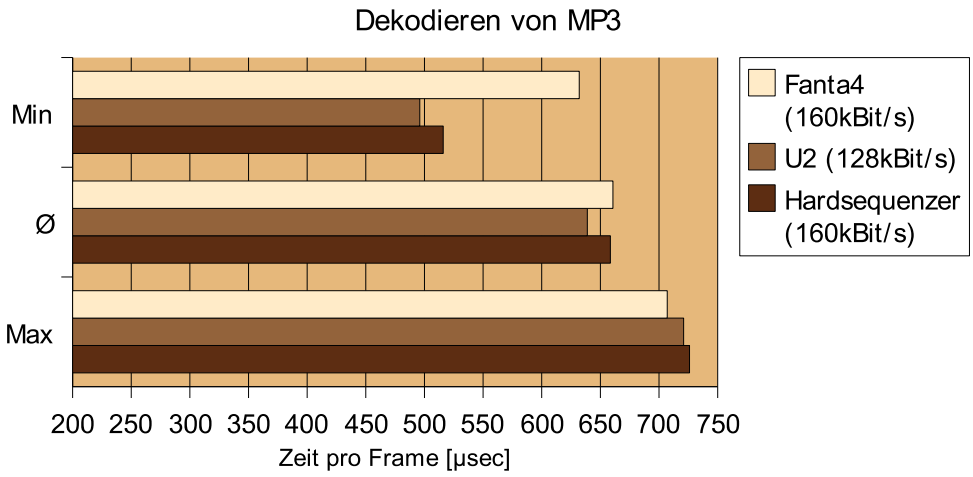

Abbildung 32: Meßwerte für mpg123: Dekodieren

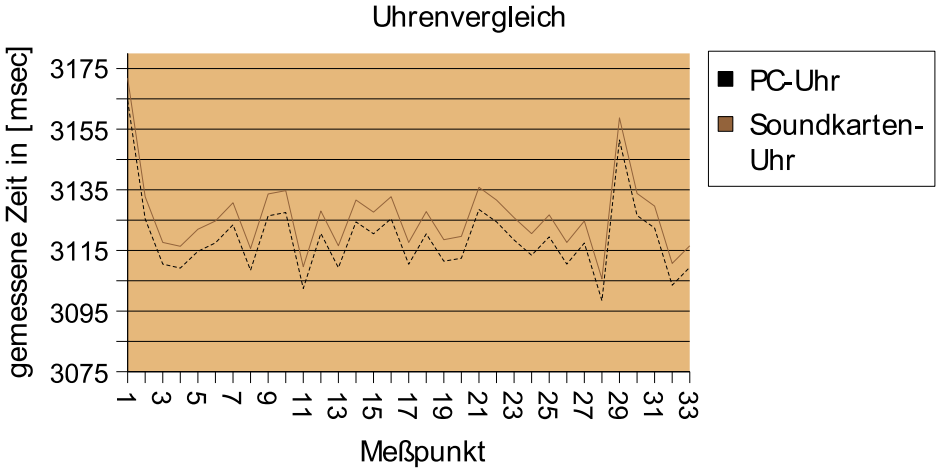

Abbildung 33: Meßwerte für Uhrenvergleich von System- und Soundkartenuhr

Literaturverzeichnis

[1] Carsten Rietzschel. Portierung eines Video-Codecs auf DROPS. TU Dresden, 2002

[2] Claude-Joachim Hamann. Schedulingtheorie - Echtzeit-Scheduling 7-1. 2001
http://i30www.ira.uka.de/research/talks/guesttalks/hamann/sched_sum2001/sched7-01.pdf

[3] Hermann Härtig. Folien zur Vorlesung Echtzeitsysteme. TU Dresden. 2001
http://os.inf.tu-dresden.de/Studium/Echtzeitsysteme/folien

[4] Konrad Froitzheim. Multimedia-Systeme. 2002
http://www-vs.informatik.uni-ulm.de/Lehre/MM-HTML/Kapitel5.doc

[5] Michael Hohmuth. DROPS - Overview. TU Dresden, 2001
http://os.inf.tu-dresden.de/drops/overview.html

[6] Christian Helmuth. Ein Konsolensystem für DROPS. TU Dresden, 2001

[7] Claude-Joachim Hamann, Jork Löser, Lars Reuther, Sebastian Schönberg, Jean Wolter, Hermann Härtig. Quality-Assuring Scheduling – Using Stochastic Behavior to Improve Resource Utilization. TU Dresden, 2001

[8] Karsten Sühring, Dr. Thomas Wiegand, Dr. Heiko Schwarz. Effizienter kodieren - Details zum kommenden Videostandard H.264/AVC. c't-Magazin, Heft 6, 2003

[9] Uwe Schneider, Dieter Werner. Taschenbuch der Informatik. 4.Auflage, 2001

[10] MPEG Home Page.
http://mpeg.telecomitalialab.com

[11] Volker Zota. Kompressionisten - Aktuelle Video-Codecs im Vergleich. c't-Magazin, Heft 10, 2003

[12] Glossary. 2003
http://www.doom9.org/glossary.htm

[13] MP3'Tech - www.mp3-tech.org
http://www.mp3-tech.org

[14] Rob Koenen. Overview of the MPEG-4 Standard. 2001
http://www.m4if.org/resources/Overview.pdf

[15] XviD.org :: Home of the XviD codec
http://www.xvid.org

[16] H.264/MPEG-4 AVC Video Compression Tutorial. LSI Logic Corporation, 2003
http://www.lsilogic.com/products/islands/h264/H.264_MPEG4_Tutorial.pdf

[17] Gabriel Bouvigne. MP3'Tech - Overview of the MPEG committee. 1997-2001
http://www.mp3-tech.org/layer3.html

[18] Gabriel Bouvigne. MP3'Tech - Overview of the MP3 techniques. 1997-2001
http://www.mp3-tech.org/tech.htm

[19] LAME Ain't an MP3 Encoder
http://lame.sourceforge.net

[20] Michael Hipp. mpg123, Fast MP3 Player for Linux and UNIX systems. 2001
phttp://www.mpg123.de/

[21] xiph.org: the Ogg project. 2003
http://www.xiph.org/ogg

[22] Volker Zota, Andree Buschmann. Konkurrierende Klangkonserven - Verlustbehaftete Audioformate im Vergleich. c't-Magazin, Heft 23, 2000

[23] Dolby Laboratories, Inc. Dolby Digital - General. 2002
http://www.dolby.com/digital/diggenl.html

[24] Bjarne Lundgren. Audio/Video Interleave Files AVI - MovieCodec.com. 1998-2002
http://www.moviecodec.com/filetypes/avi.shtml

[25] ASF Specification - Windows Media Technologies.
http://www.microsoft.com/windows/windowsmedia/WM7/format/asfspec11300e.asp

[26] Matroska. 2003
http://matroska.sourceforge.net

[27] Michael Hohmuth. Smart_MPEG: MPEG decoder library specification and manual. TU Dresden, 2000

[28] Jörg Nothnagel. Entwurf und Implementierung von Sound- und Synchronisationskomponenten für das Smart-MPEG-Projekt. TU Dresden, 2001

[29] Bram Avontuur. vcr, a text-console video recorder for Linux. 2001
http://www.stack.nl/~brama/vcr/

[30] nvrec - High quality video capture for Linux
http://nvrec.sourceforge.net/

[31] Klaus Schmidinger. Video Disk Recorder. 2003
http://www.cadsoft.de/vdr/

[32] Jean Wolter. L4-Scheduling: Presentation Slides. Internal Presentation, TU-Dresden, 2003

[33] lame/lame/libmp3lame
http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/lame/lame/libmp3lame/

[34] Lars Reuther, Martin Pohlack. Rotational-Position-Aware Real-Time Disk Scheduling Using a Dynamic Active Subset (DAS). TU Dresden, 2003

[35] Maciej Suchomski. Project RETAVIC. 2002
http://www6.informatik.uni-erlangen.de/retavic/

[36] Andreas Märcz. Homepage memo.REAL. 2002
http://wwwdb.inf.tu-dresden.de/research/memo.REAL/

[37] Sylvia Schulze. Evaluation des Datenaufkommens audiovisueller Datenströme verschiedener Themen (Überarbeitete Version). 2002
http://6bone.informatik.uni-leipzig.de/sylvia/mpeg.pdf

[38] Norman Feske. DOpE - a graphical user interface for DROPS. TU Dresden, 2002
http://os.inf.tu-dresden.de/~nf2/files/DOpE/documents/DOpE-diploma.pdf

[39] C.-J. Hamann, L. Reuther. Pufferdimensionierung für schwankungsbeschränkte Ströme in DROPS. 10. GI/ITG Fachtagung MMB'99 Trier, 1999
http://os.inf.tu-dresden.de/%7Ereuther/publications/trier99.pdf

[40] The xine-Project. xine - A Free Video Player. 2002
http://xinehq.de/

[41] MPLAYERHQ:HU _ THE MOVIE PLAYER FOR LINUX. 2003
http://www.mplayerhq.hu/homepage/

[42] Andrew S. Tanenbaum, Maarten van Steen. Distributed Systems: Principles and Paradigms. Chapter 5
http://www.cs.vu.nl/~ast/books/ds1/powerpoint.html

[43] L4 Environment. TU-Dresden, 2003
http://os.inf.tu-dresden.de/l4env/about.xml

[44] Christian Helmuth. Generische Portierung von Linux-Gerätetreibern auf auf die DROPS-Architektur. TU Dresden, 2001

[45] FFmpeg Documentation. 2003
http://ffmpeg.sourceforge.net/ffmpeg-doc.html

[46] PortAudio - portable cross-platform Audio API
http://www.portaudio.com/

[47] SourceForge.net: xine-user
http://sourceforge.net/mailarchive/forum.php?forum_id=3438

[48] XviD Team. XviD API 2.1 Reference (for 0.9.x series). 2003

[49] Welcome to the VQEG Home Page
http://www.its.bldrdoc.gov/vqeg/tsdownload.htmlvqeq
http://www.its.bldrdoc.gov/vqeg/thumbnails/thumbnails.html