Diplomarbeit

zum Thema

Entwicklung und Implementierung einer echtzeitfähigen Multimediakonverterschnittstelle

an der

Technischen Universität Dresden

Fakultät Informatik

Institut Systemarchitektur

Professur Datenbanken

eingereicht von:	Sven Schmidt ss54@inf.tu-dresden.de
geboren am:	06.03.1978
Betreuender Hochschullehrer:	Prof. Dr. Hermann Härtig
Betreuender Mitarbeiter:	Dipl.-Inf. Andreas Märcz
eingereicht am:	02.12.2002

Aufgabenstellung

Seite 1

Seite 2

Seite3

Danksagung

Für die vielen Anregungen und hilfreichen Diskussionen danke ich Prof. Klaus Meyer-Wegener und Andreas Märcz. Jork Löser gilt mein Dank für die Hinweise und Vorschläge zu DROPS und DSI. Ganz besonders danke ich Frank Mehnert und Lars Reuther für die Hilfe bei der Einrichtung der Programmierumgebung und die fortlaufende Unterstützung bei damit verbundenen Problemen.

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst zu haben.

Sven Schmidt

Dresden, 01.12.2002

Inhaltsverzeichnis

Abkürzungsverzeichnis

AC-3	Audio Codec 3
API	Application Programming Interface
AVI	Audio Video Interleave
CORBA	Common Object Request Broker Architecture
CSI	Component Streaming Interface
DCT	Discrete Cosine Transform
DROPS	Dresden Real Time Operating System
DSI	DROPS Streaming Interface
DVD	Digital Versatile Disc
FIFO	First In First Out
FLICK	Flexible IDL Compiler
GOP	Group of Pictures
I-, P-, B-, D - Frame	Intra-coded, Predictive-coded, Bi-directionally predictive-coded, DC-coded Frame
IDL	Interface Definition Language
IPC	Interprozess Kommunikation
JCP	Jitter-constrained Periodic
MPEG	Motion Pictures Expert Group
MUX, DeMUX	Multiplexer, De-Multiplexer
PCM	Pulse Code Modulation
RGB	Farbformat: Rot-, Grün-, Blau-Information
RIFF	Resource Interchange File Format
RT	Real Time
sbS	schwankungsbeschränkter Strom

TFTP .... Trivial File Transfer Protocol

TLV .... Tag Length Value

VOB .... Video Object

YUV .... Farbformat: Y..Helligkeitsinformation, U,V..Farbinformation

1. Einführung

Das Projekt memo.REAL beschäftigt sich mit der format- und geräteunabhängigen Speicherung von Medienobjekten.

Die Format-Unabhängigkeit stellt dabei eine besondere, bisher nicht befriedigend gelöste Herausforderung dar.

Medienobjekte wie Bilder, Videos oder Musikstücke werden in den verschiedensten Formaten abgespeichert. Bei einer darauffolgenden Anforderung der Daten durch den Benutzer soll dieser nicht durch das Speicherformat der Medienobjekte eingeschränkt werden, sondern im Zuge seiner Anforderung, auch über das zu liefernde Format bestimmen können.

Das *Format* eines Medienobjektes beschränkt sich dabei nicht auf das Kodierungsverfahren. Eigenschaften wie Bildgröße, Farbtiefe und Tonqualität gehören ebenso dazu wie Framerate, Komprimierungsparameter und Wiedergabegeschwindigkeit. Ziel dieser Format-Unabhängigkeit ist es, unabhängig vom internen Format des Medienobjektes eine externe Repräsentation bereitzustellen, welche optimal auf die Bedürfnisse der datenverarbeitenden Anwendung zugeschnitten ist. Dabei sollen die Daten, die der Anwendung übergeben werden, auf ein Minimum reduziert und die Anwendung von trivialen Operationen bzgl. der Medienobjekte entlastet werden. ([Mar])

Es ist die Aufgabe des Datenhaltungssystemes, diese Format-Unabhängigkeit von Speicherung und Benutzer-Anforderung zu gewährleisten. Zur Format-Anpassung stehen Konverter zur Verfügung.

Die Medienobjekte könnten in mehreren verschiedenen Formaten gespeichert werden und zur Zeit der Anforderung kann eines dieser Formate ausgewählt werden. Ganz offensichtlich können somit nicht alle möglichen Benutzeranforderungen abgedeckt werden. Die Speicherung wäre in hohem Maße redundant und im Hinblick auf die Größe mancher Medienobjekte (Audio, Video) wäre die Kapazität vieler Datenhaltungssysteme schnell erschöpft.

Eine andere Möglichkeit wäre, sämtliche Medienobjekte eines Medientypes bei der Speicherung in ein einziges, internes Format zu überführen und bei der darauffolgenden Benutzeranforderung in das gewünschte Zielformat zu konvertieren. Die Medienobjekte müssten dabei zweimal konvertiert werden, einmal bei der Speicherung, ein zweites Mal bei der Benutzeranforderung. Außerdem kann es durch das eine, feststehende interne Datenformat zu Informationsverlusten kommen, welche in Anbetracht mancher Benutzer-Quell- und Zielformate hätten vermieden werden können.

Besser ist es, die Medienobjekte in genau dem Format zu speichern, in dem diese vom Benutzer übergeben worden sind. Dann ist eine einzige Konvertierung unmittelbar vor der Benutzeranforderung notwendig (Abb. 1.1). Soll diese Konvertierung dann jeweils von einem einzigen Konverter erledigt werden, müsste in Anbetracht der großen Anzahl von möglichen Formaten eine noch größere Anzahl von Konvertern existieren.

Um dem zu begegnen, wird für die Aufgabe nicht mehr nur ein Konverter, sondern eine ganze *Konverterkette* herangezogen.

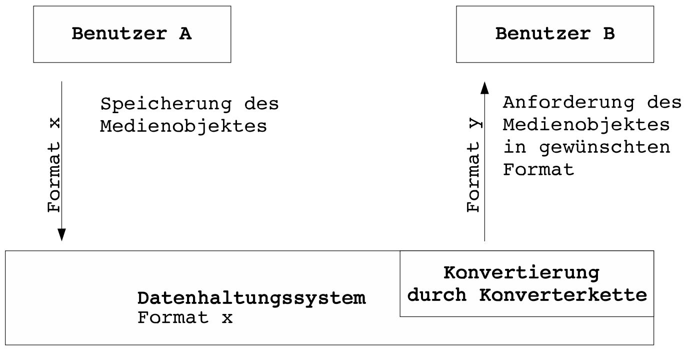

Abbildung 1.1.: Speicherung des Medienobjektes im Ursprungsformat, Konvertierung bei Anforderung

Dahinter steckt die Idee, die Konvertierung von dem Quell- in das Ziel-Format in kleineren Arbeitsetappen zu erledigen und dadurch die Anzahl der nötigen Konverter beträchtlich zu reduzieren.

Eine besondere Herausforderung stellen die zeitabhängigen Medienobjekte wie Audio und Video dar. Sie können nicht als ein Ganzes verarbeitet werden. Zum einen liegt das in ihrer teilweise enormen Größe begründet, zum anderen ist es aufgrund ihrer zeitlichen Eigenschaften sinnvoll, nur den Teil des Medienobjektes zu einem bestimmten Zeitpunkt bereitzustellen, der auch benötigt wird.

Daraus resultieren zeitliche Anforderungen an das Datenhaltungssystem und an die Konverterkette. Dem Benutzer muss eine gewisse Dienstgüte ([Ste99]) garantiert werden, um beispielsweise ein Video ruckelfrei betrachten oder ein Musikstück unterbrechungsfrei anhören zu können.

Aus diesem Grund werden die o.g. Aufgaben im Rahmen des Projektes memo.REAL durch ein *Echtzeitbetriebssystem* eingeplant und verwaltet.

Ziel dieser Arbeit ist es nun, ausgehend von einer Echtzeitumgebung und von Anforderungen an den Konvertierungs- und Datenübertragungsprozess die Möglichkeiten der Speicherung und Übertragung von Medienobjekten zu untersuchen und eine Vorlage für Konverter zu schaffen, die in einer Kette organisiert sind. Dadurch soll es dem Anwendungsentwickler ermöglicht werden, seine spezifischen Konverter in einfacher Art und Weise zu implementieren und in die Echtzeit-Umgebung einzupassen.

Gliederung dieses Dokumentes

Im folgenden Kapitel wird das Echtzeitbetriebssystem DROPS[1] im Hinblick auf die Implementierung einer Multimediakonverterumgebung beschrieben. Es wird die Grundlage für Konzeptionierung und Implementierung dieser Arbeit bilden.

Danach wird versucht, mit Hilfe zweier Konverterbeispiele die generelle Funktionsweise von Multimediakonvertern zu verallgemeinern und dies durch ein Modell zu beschreiben. Die Idee ist, eine einfachste Klassifizierung von Konverter herauszuarbeiten, auf deren Grundlage die Realisierung der zu entwickelnden Konverterschnittstelle erfolgen kann.

Kapitel 4 stellt zwei Multimedia-Dateiformate vor. An dem Dateiaufbau ist zu sehen, in welcher Art und Weise zeitabhängige Multimediadaten in einem Dateisystem abgelegt werden können. Außer-

[1] Dresden Realtime Operating System

dem wird ausgehend davon ein sehr einfaches Modell der Speicherung und der Datenübertragung zur Benutzung in der Beispielimplementierung gewählt.

In der letzten Vorbetrachtung in Kapitel 5 wird das Modell der schwankungsbeschränkten periodischen Ströme (sbS) erläutert. Dieses Modell bildet die Grundlage für den Datenaustausch in Echtzeit zwischen den einzelnen Konvertern.

Kapitel 6 beschäftigt sich mit einigen Ideen zur Umsetzung der Datenübertragung und der Datenspeicherung sowie mit Möglichkeiten der Steuerung und Kontrolle der Konverterumgebung.

Danach folgt in Kapitel 7 die Beschreibung des Entwurfes der zu entwickelnden Schnittstelle. Es wird dabei sowohl auf die Interaktion mit DROPS als auch auf den Programmentwurf eingegangen.

Abschließend wird das entwickelte System bewertet und es werden Ansätze für weitere Arbeiten aufgezeigt.

2. Untersuchung vorhandener Zugriffsschnittstellen

2.1. Allgemeines

Grundlage des Zusammenspiels mehrerer Konverter ist die Produzenten-Konsumenten-Beziehung. Zwischen Produzent und Konsument müssen Mediendaten ausgetauscht werden. Mechanismen, die in dem zugrundeliegenden Betriebssystem für den Datenaustausch sorgen, sollten schnell, einfach und flexibel sein und die nötige Sicherheit bezüglich der Synchronisation von Produzent und Konsument gewährleisten.

Weiterhin sollte es sich um eine Betriebssystemumgebung handeln, die die Eigenschaften eines *offenen Systemes*[1] erfüllt. Optimal wäre zusätzlich eine breite Unterstützung von Hardware und eine reichhaltige Ausstattung mit Systemsoftware.

Im Folgenden wird nur das Echtzeitbetriebssystem DROPS vorgestellt, da die Entscheidung DROPS zu benutzen bereits vor Beginn dieser Arbeit gefällt wurde.

2.2. DROPS

Das Betriebssystem DROPS wird am Lehrstuhl Betriebssysteme der TU-Dresden entwickelt. Es baut auf einem Mikrokern auf. Als Echtzeit-Betriebssystem wird mit DROPS die Unterstützung von Anwendungen mit „weichen“ Echtzeitanforderungen realisiert.

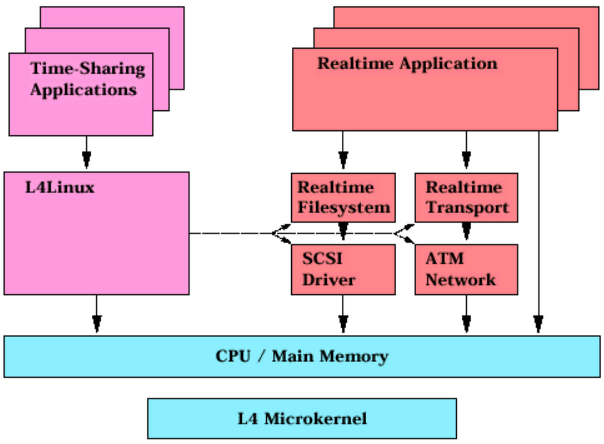

Abbildung 2.1.: DROPS Architektur (aus [HBB+])

[1] offengelegte Schnittstellen

Echtzeit- und Nicht-Echtzeitanwendungen können dabei parallel betrieben werden. Durch die Entwicklung von L4Linux ist es möglich, bestehende Nicht-Echtzeitanwendungen, die für Linux entwickelt worden, ohne erneute Programmübersetzung auch in der DROPS-Umgebung zum Einsatz zu bringen. L4Linux ist ein für die Ausführung auf dem L4-Mikrokern angepasstes Linux.

Prinzipiell könnten auch andere bestehende *offene* Betriebssysteme für den Einsatz auf dem L4-Mikrokern angepasst werden und somit die Grundlage für den Nicht-Echtzeitteil des Gesamtsystemes DROPS bilden.

Damit Echtzeit-Anwendungen auf die Peripherie des Systemes zugreifen können, muss für die gewünschten Ressourcen jeweils ein echtzeitfähiger Treiber entwickelt werden. Bereits möglich ist der Zugriff auf Netzwerk, Sound-, Video- und Grafikhardware.

Am echtzeitfähigen Dateisystem- bzw. Festplattenzugriff wird z.Z. noch gearbeitet.

DSI - DROPS Streaming Interface

Bezüglich des Datenaustausches für die Multimediakonverterumgebung stehen die Mechanismen des gemeinsam genutzten Speichers (Shared Memory) zur Verfügung. DROPS erweitert mit dem DSI die Benutzung von gemeinsamem Speicher um Möglichkeiten der Synchronisation des Zugriffes darauf. DSI stellt dabei eine Schnittstelle für die Kommunikation zwischen Produzent und Konsument in einer Echtzeit-Umgebung zur Verfügung. Es wurde berücksichtigt, dass die auszutauschenden Daten zeitbehaftet sind und unter Umständen (noch) nicht zur Verfügung stehen.

Die produzierten Daten sind nur eine begrenzte Zeit lang gültig. Um dies auszudrücken wird den produzierten Daten eine virtuelle Zeit zugewiesen, welche mit der Reihenfolge der Daten im Datenstrom in Zusammenhang steht. Die Verknüpfung von virtueller Zeit und echter Zeit liegt dabei im Verantwortungsbereich des Produzenten bzw. des Konsumenten.

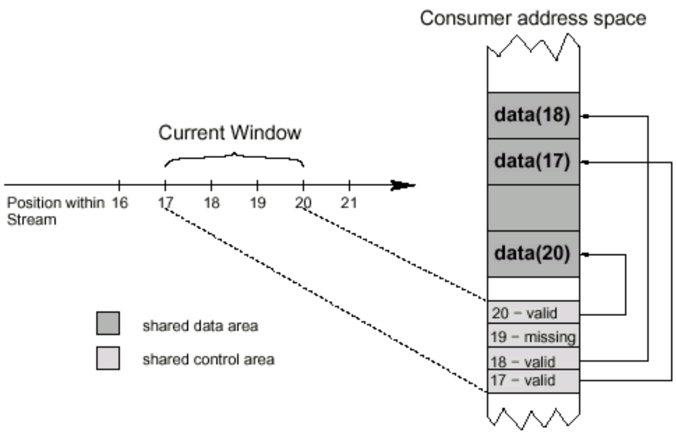

Abbildung 2.2.: DSI Daten- und Kontrollbereich (aus [LHR])

Für die Übertragung der Daten zwischen Produzent und Konsument wird ein Pufferbereich vor Beginn des Datenaustausches reserviert. Es wird ein Kontrollbereich und ein Datenbereich benötigt (Abb. 2.2). Der Kontrollbereich dient als eine Indirektion des Datenzugriffes und besteht aus sogenannten Paket-Deskriptoren [2]. Der Produzent schreibt nun die Daten in den reservierten Datenbereich

[2]Lt. [LHR] können durch diese Indirektion auch Speicheranordungen abgedeckt werden, bei denen die

und gibt im Paket-Deskriptor (Kontrollbereich) die Position und Größe der geschriebenen Daten zusammen mit der zugeordneten virtuellen Zeit bekannt. Der Konsument liest diesen Paketdeskriptor und damit die beschriebenen Daten und signalisiert im Kontrollbereich den Verbrauch eines Paketes. Es ist auch möglich, dass sich mehrere Produzenten und Konsumenten einen einzigen Datenbereich teilen.

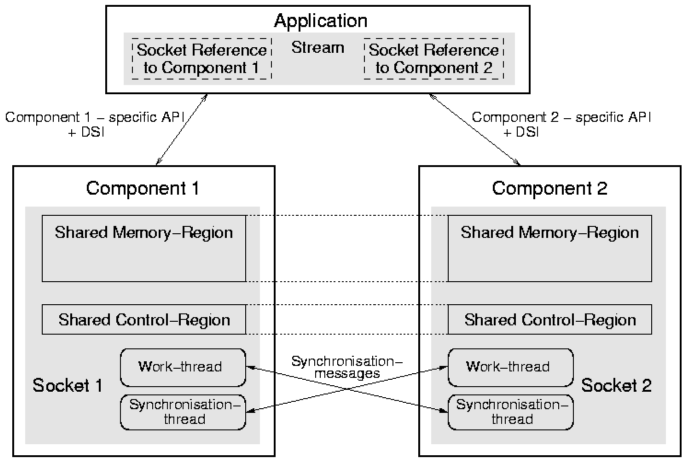

Abbildung 2.3.: DSI Komponenten (aus [LR01])

Die Kommunikation zwischen Konsument und Produzent kann blockierend oder nicht-blockierend erfolgen. Im ersten Fall wird der Produzent blockiert, falls er Daten senden will und der Puffer bereits voll ist. Ebenso wird der Konsument blockiert, wenn dieser bei leerem Puffer Daten zu empfangen versucht. Bei nicht-blockierender Kommunikation muss die Synchronisation durch Scheduling, Polling und Signalisierungen erfolgen.

Wie in Abb. 2.3 zu sehen ist, besitzt jede Komponente zusätzlich zum Work-Thread noch einen Synchronisations-Thread und es werden entsprechende Synchronisations-Nachrichten ausgetauscht.

Zusammenfassung

Da sowohl DROPS als auch diese Diplomarbeit an der TU-Dresden entwickelt wurden, lag es nahe, DROPS als Umgebung für die Multimediakonverter zu benutzen.

Die DROPS-Architektur und das DROPS Streaming Interface untermauern diese Entscheidung.

Daten nicht in zusammenhängendem Speicher abgelegt werden

3. Untersuchung vorhandener Multimediakonverter

Im Folgenden werden zwei existierende Konverter untersucht. Dabei soll eine für die Implementierung einer Konverterschnittstelle notwendige einfache Klassifizierung erarbeitet werden.

3.1. Beispiel: Framegrößenkonverter

Der Framegrößenkonverter wurde am Institut Datenbanken mit dem Ziel entwickelt, Zeitmessungen für bestimmte Videobearbeitungsschritte durchzuführen um Konverter später bezgl. ihres Ressourcenbedarfs (Rechenleistung) in einer Konverterkette einplanen zu können.

Dieser spezielle Konverter ist eine Linux-Applikation und operiert auf zwei AVI-Dateien im lokalen Dateisystem. Die Eingabedatei wird sukzessiv ausgelesen. Unter Beachtung der Metainformationen werden die Videoquanten identifiziert, einer Framegrößenkonvertierung unterzogen und mit aktualisierten Metainformationen in die Ausgabedatei geschrieben.

Aus Abbildung 3.1 lässt sich eine Trennung von umgebungsabhängigen Vor- und Nachbereitearbeiten und der zentralen Steuerschleife erkennen. Den Kern der zentralen Steuerschleife bildet die Verarbeitungsfunktion, welche für jedes Frame durchlaufen wird. Für diese Verarbeitungsfunktion müssen zum einen die Eingabedaten bereitgestellt werden, zum anderen muss das Ergebnis der Verarbeitung in geordneter Art und Weise abgelegt werden. In diesem Fall stehen für die Ein- und Ausgabe jeweils eine Datei im lokalen Dateisystem zur Verfügung.

In der bisherigen Form bestehen an den Framegrößenkonverter keine Echtzeit-Anforderungen.

3.2. Beispiel: Smartmpeg

Smartmpeg ist eine Sammlung von Bibliotheken zum Dekodieren, Bearbeiten und Wiedergeben von MPEG-Bild- und Toninformationen.

3.2.1. Pufferverwaltung

Der MPEG-Datenstrom wird durch eine Sequenz von Puffern dargestellt, auf die über Iteratoren zugegriffen wird [Hoh00].

Die einzelnen Pufferelemente haben dabei eine feste Größe. Um ein Frame des Videostromes zu dekodieren, werden also u.U. mehrere aufeinanderfolgende Pufferelemente benötigt.

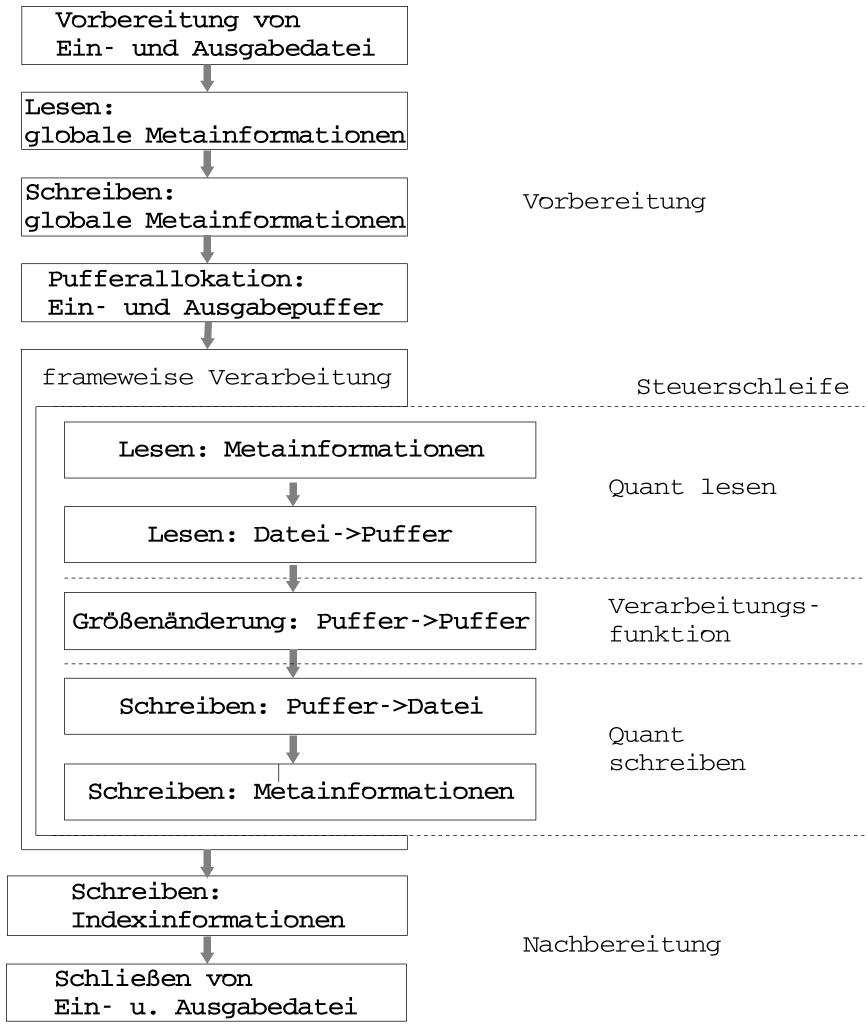

Abbildung 3.1.: Ablauf Framegrößenkonverter

3.2.2. Konverter dsi-mpegview

Im Gegensatz zu dem Framegrößenkonverter wurden Smartmpeg und dsi-mpegview bereits für DROPS entwickelt. Die Komponente dsi-mpegview nutzt die Smartmpeg-Bibliotheken und stellt die Verbindung zwischen dem Lesen des Eingabestromes, dem Dekodieren der einzelnen Frames und dem Anzeigen derer dar.

Zur Datenübertragung werden die o.g. Puffer auf das DSI abgebildet. Das Dateisystem (Ramdisk) stellt die MPEG-Datei über eine DSI-Schnittstelle in Form von einzelnen Blöcken bereit.

Der Dekoder greift über seine Puffer-Schnittstelle auf diese DSI-Pakete zu und dekodiert daraus die einzelnen Frames.

Dieser Zusammenhang ist in Abb. 3.2 dargestellt.

Das Dateisystem liefert die MPEG-Daten mit einer fest eingestellten Datenrate an dsi-mpegview, welcher für das Dekodieren der einzelnen Frames bestimmte Zeitschranken einhalten muss. Es gibt dabei einen Pflichtteil und einen optionalen Teil. Ist die eingeplante Zeit nach dem Dekodieren des Pflichtteils bereits erreicht oder überschritten, wird der optionale Teil nicht durchgeführt. Ist der Dekoder mit seiner Arbeit vor Erreichen des Zeitscheibenendes fertig, so wartet er die restliche Zeit. Der MPEG Dekoder bietet außerdem die Möglichkeit, Frames zu verwerfen. Dies muss ihm explizit mitgeteilt werden.

Die Anzeigekomponente synchronisiert sich mit dsi-mpegview, um die gewünschte Framerate einzuhalten. Sollten die Frames nicht rechtzeitig zur Verfügung stehen, kann die geplante Framerate der Anzeigekomponente nicht eingehalten werden.

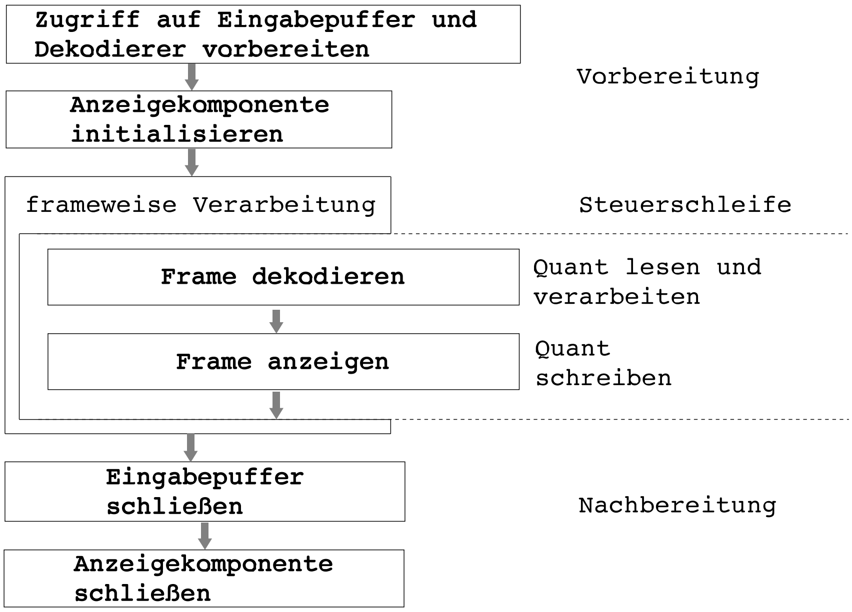

Abbildung 3.2.: Ablauf dsi-mpegview

3.3. Generelle Funktionsweise von Multimediaonvertern

Da bezüglich des Ablaufplanes des in Abb. 3.1 dargestellten Framegrößenkonverters und des in Abb. 3.2 dargestellten dsi-mpegview Gemeinsamkeiten erkennbar sind, soll nun versucht werden, den Arbeitsablauf in den Konvertern zu verallgemeinern.

Die vorbereitenden Arbeiten und die Nachbereitung laufen bei den zwei Beispielkonvertern unterschiedlich ab, da es sich um umgebungsspezifische Aufgaben handelt. Beim Framegrößenkonverter werden dabei beispielsweise Dateien geöffnet bzw. geschlossen, wogegen bei dsi-mpegview mit dem DSI kommuniziert werden muss.

Die Steuerschleife ist beim dsi-mpegview als extra Thread implementiert, in dem die eingehenden Daten dekodiert und dann der Anzeigekomponente zugeführt werden, welche die Frames letztendlich in der DROPS-Konsolenumgebung darstellt. Beim Framegrößenkonverter werden fortlaufend aus der Eingabedatei die Frames des Videos gelesen, der Verarbeitungsfunktion zur Verfügung gestellt und in die Ausgabedatei geschrieben.

Ein charakteristischer Teil jedes Konverters ist seine Verarbeitungsfunktion. Die zentrale Steuerschleife kontrolliert die Verarbeitungsfunktion und stellt Schnittstellen für die Übergabe der ein- und ausgehenden Daten bereit („Quant lesen“ und „Quant schreiben“).

Obwohl die Teilabläufe der Konverter natürlich jeweils unterschiedlich sind, lässt sich ein allgemeiner Arbeitsablauf erkennen, der in Abb. 3.3 dargestellt ist und die Grundlage für die Ausführungen in Kapitel 6 sowie für die Implementierung bilden wird.

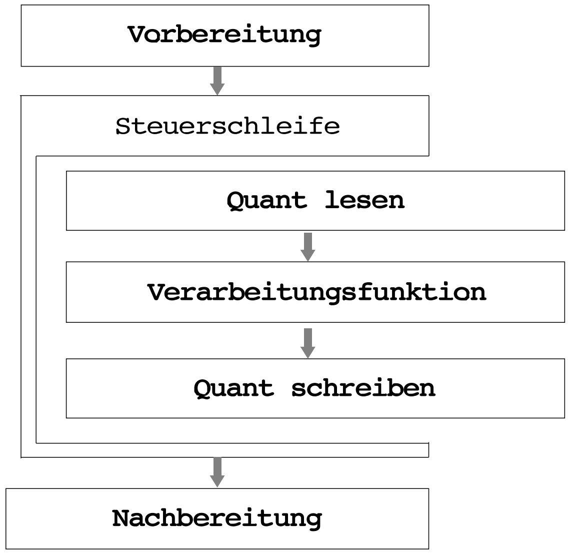

Abbildung 3.3.: allgemeiner Arbeitsablauf eines Konverters

3.3.1. Taktverhalten

Der allgemeine Arbeitsablauf lässt sich dabei noch genauer betrachten.

Gleicher Ein- u. Ausgangstakt

Diese Art der Konverter verarbeiten jeweils einen Eingangsquanten zu einem Ausgangsquanten (Abb. 3.4), Größenänderung der Quanten möglich). Die Verarbeitungsfunktion des Konverters ist also unabhängig von früher oder später einlaufenden Mediendaten. Die Einhaltung dieses Kriteriums ist außerdem von der gewählten Quantengröße abhängig.

Wenn die Quantengröße jeweils ein Frame (nur intraframe-komprimiert) ist, sind derartige Konverter z.B. Framegrößenkonverter und Farbraumkonverter. Auch der in Kapitel 3.1 vorgestellte Framegrößenkonverter fällt in diese Kategorie.

Unterschiedlicher Ein- und Ausgangstakt

Bei diesen Konvertern werden entweder mehrere Eingangsquanten benötigt, um einen Ausgangsquanten zu erzeugen oder aus einem Eingangsquant entstehen mehrere Ausgangsquanten. Letzteres ist in

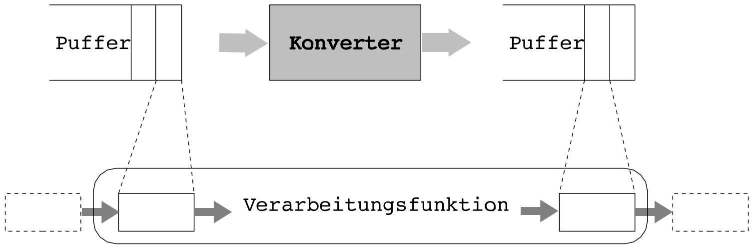

Abbildung 3.4.: Konverterfunktion, Eingangstakt gleich Ausgangstakt

Abb. 3.5 dargestellt.
Ein Beispiel für diese Konverterklasse ist ein Frameratenkonverter.

Soll die Framerate beispielsweise genau halbiert werden und ein Quant beinhaltet genau ein Frame, benötigt der Konverter zwei aufeinanderfolgende Eingangsquanten und bildet daraus ein neues Frame, welches er im Ausgabepuffer ablegt. Eine Frameratenverdopplung wäre ebenso denkbar: aus einem ankommenden Frame enstehen durch Verdopplung oder Interpolation zwei neue Frames.
Der in Kapitel 3.2 vorgestellte dsi-mpegview Konverter verhält sich ähnlich: Da er Pakete fester Größe vom Dateisystem liest, muss er teilweise auf die Ankunft mehrerer solcher Pakete warten, um daraus die Audio- und Videodaten dekodieren zu können.

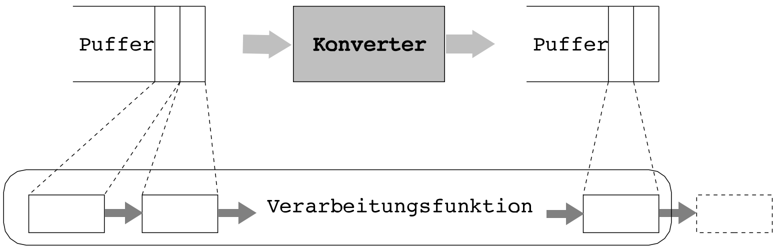

Abbildung 3.5.: Konverterfunktion, Eingangstakt ungleich Ausgangstakt

Bei dieser Art von Konvertern gibt es nun verschiedene Verfahrensmöglichkeiten. Angenommen ein Konverter benötigt mehrere Eingabequanten, um einen Ausgabequant zu erzeugen: Einerseits könnte der Konverter alle bis auf den letzten benötigten Eingabequanten lokal puffern und, wenn der letzte Quant eintrifft seine Verarbeitung starten. Dazu müsste lokal Speicher belegt werden und es müssten Daten aus dem DSI-Puffer in einen lokalen Programmpuffer kopiert werden.

Diese Variante bietet bezüglich Speicherverbrauch, Verarbeitungsgeschwindigkeit und zeitlicher Verzögerung der Verarbeitung keinerlei Vorteile und wird daher nicht weiter betrachtet.

Die bessere Möglichkeit ist abzuwarten, bis sämtliche benötigte Quanten im DSI-Eingabepuffer verfügbar sind und dann die Verarbeitung zu starten. Der DSI-Puffer müsste in diesem Fall größer eingeplant werden (unter Berücksichtigung der Menge an Quanten, die benötigt wird). Der Vorteil besteht hierbei in der Einsparung einer Kopieroperation.

Einschränkung

Zu welcher Klasse ein Konverter gehört, ist von seiner Funktion und von seiner Implementierung abhängig. Eine Verallgemeinerung der hier angedeuteten Klassifizierung wird im weiteren Verlauf des Dokumentes vorgestellt.

Für das in Abschnitt 6 vorgestellte Modell soll vorerst die Einschränkung „Eingangstakt gleich Ausgangstakt“ gelten. Es wird allerdings auch gezeigt werden, dass das Modell auf eine sehr einfache Art und Weise auf den allgemeineren Fall „Eingangstakt ungleich Ausgangstakt“ erweitert werden kann.

3.4. Zusammenfassung

Selbst bei gleicher Konverterumgebung (DROPS, DSI) reicht es nicht in jedem Fall aus, die Verarbeitungsfunktion herauszulösen. Es muss zusätzlich eine Möglichkeit existieren, die gesamte zentrale Steuerschleife konverterspezifisch zu implementieren.

4. Vorbetrachtungen Multimedia Formate

Obwohl nicht explizit durch die Aufgabenstellung gefordert, erscheint mir an dieser Stelle ein Blick auf zwei weit verbreitete Multimedia-Dateiformate sinnvoll. Mein Ziel dabei ist, Informationen über die Struktur dieser Dateien für die zu realisierende Multimediakonverterschnittstelle zu nutzen.

4.1. Das Tag-Length-Value Prinzip

Vereinfacht gesagt ist Tag-Length-Value (TLV) eine Möglichkeit, Datensätze variabler Länge zu verwalten ([MW]).Bezogen auf Speicherformate kann über eine hinzugefügte Indirektion (Index) direkt auf einzelne Sätze in einer Datei zugegriffen werden. Viele Daten sind in einem Tag-Length-Value - Format abgelegt oder werden auf diese Art transportiert.

Im einzelnen bedeuten:

- Tag (feste Größe): z.B. Identifikation des Datensatzes [1]
- Length (feste oder variable Größe): Länge des Datensatzes
- Value (variable Größe): die Daten selbst

Auch viele Multimedia-Daten sind derartig strukturiert. Zwei Beispiele sollen hier erläutert werden:

4.2. AVI

(nach [Wei])

AVI steht für Audio Video Interleave. Es ist eine Form des RIFF Formates (Resource Interchange File Format) und wurde von Microsoft spezifiziert. Es dient dazu, auf rekonstruierbare Art und Weise Audio- und Videoinformationen zu speichern. Eine AVI-Datei besteht dabei aus mehreren ineinander verschachtelten Datenstrukturen, welche wiederum unterteilt werden können. Es ist weiterhin möglich, auch andere Datenströme außer Audio und Video zu speichern. Ein Beispiel hierfür sind Kontrollströme für Geräte, die dann von der Abspielsoftware entsprechend interpretiert werden.

Die Datenstrukturen aus denen eine AVI- bzw. RIFF-Datei besteht, werden als Chunks (siehe Abb. 4.1) bezeichnet. Diese können dann ineinander verschachtelt beliebig viele sogenannte Sub-Chunks enthalten. Der Aufbau eines (Sub-)Chunks ist dabei immer gleich, nämlich:

- Identifizierer (feste Länge, 4 Byte)

[1] *Tag* ist optional. Auch *Length* und *Value* genügen, um Datensätze variabler Größe abzulegen

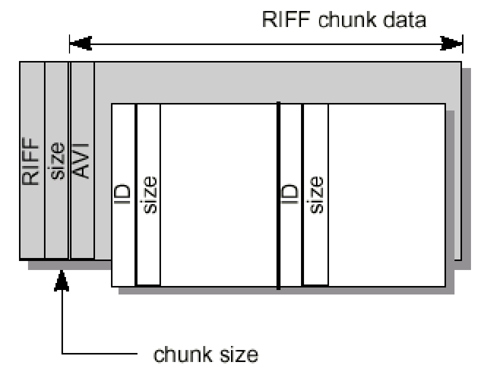

Abbildung 4.1.: Verschachtelung der AVI Chunks (aus [avi97])

- Längenangabe (feste Länge, 4 Byte)
- Daten (variable Länge, auf gerade Byteanzahl aufgefüllt)

Innerhalb einer AVI-Datei werden immer mindestens ein RIFF-Chunk und LIST Sub-Chunk vorhanden sein. Optional existiert ein Index für den LIST Sub-Chunk (Abb. 4.2).

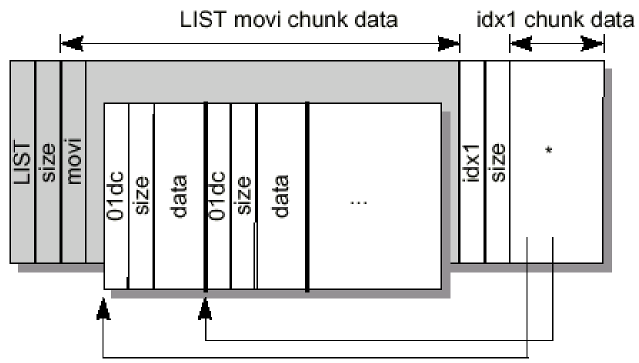

Abbildung 4.2.: LIST Sub-Chunk mit Index (aus [avi97])

4.3. MPEG

(nach [SN95], [Ste99])

Die *Motion Pictures Expert Group* wurde Ende der 80er Jahre zur Festlegung eines digitalen Standards für Bewegtbilddarstellung ins Leben gerufen. MPEG spezifiziert dabei eine Syntax für ineinander verschachtelte Audio- und Video-Datenströme.

Audiostrom

Ein Audiostrom besteht aus Frames, die in Zugriffseinheiten unterteilt sind. Jede dieser Zugriffseinheiten besteht ihrerseits aus Slots.
Die Zugriffseinheit ist die kleinste Audio-Sequenz (mit komprimierten Daten), welche unabhängig von allen anderen Daten dekodiert werden kann.

Die Zugriffseinheit eines Frames repräsentiert beispielsweise eine Abspielzeit von 8 Millisekunden bei einer Samplefrequenz von 48 KHz.

Videostrom

Wie in Abb. 4.3 zu sehen ist, besteht der Videostrom aus 6 Ebenen:

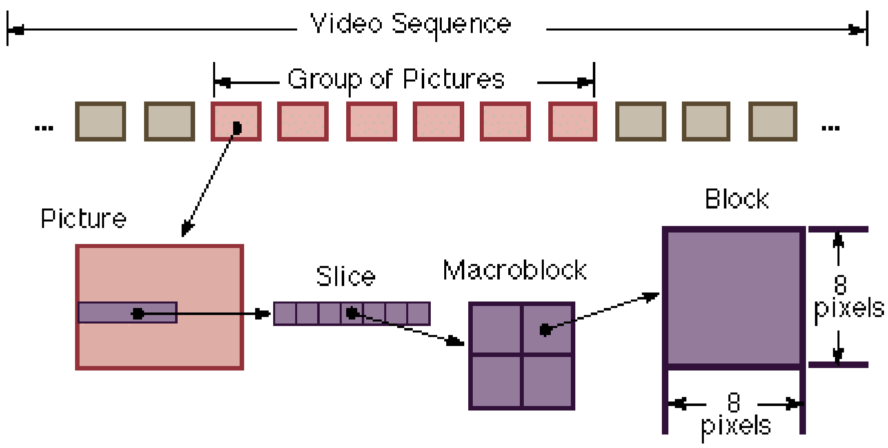

Abbildung 4.3.: Ebenen des MPEG Videostromes

- Sequenz-Ebene
  Dies ist die höchste Ebene des Videostromes.

  Hier sind Parameter wie Bildgröße, Framerate und die für das Dekodieren notwendige Speicherkapazität für diese Viedeosequenz hinterlegt.

- Group-of-Pictures - Ebene
  Diese Ebene repräsentiert die GOPs und jeder GOP enthält mindestens ein I-Frame[2] genau am Anfang.

  Es wird zwischen der Reihenfolge der Speicherung und der Reihenfolge der Darstellung der zu einem GOB gehörenden Frames unterschieden, d.h. das I-Frame am Anfang muss in jedem Fall zuerst dekodiert werden, in der Darstellung kann es jedoch hinter einem oder mehreren B-Frames erscheinen.

[2] I-, P-, B-, D - Frame: Intra-coded, Predictive-coded, Bi-directionally predictive-coded, DC-coded Frame

- Bild-Ebene
  Diese Ebene enthält die einzelnen Bilder. Ihre zeitliche Reihenfolge ist durch eine Bildnummer festgelegt. Zusätzlich wird der Typ eines jeden Bildes (I, B, P, D) vermerkt.
- Slice-Ebene
  Ein Slice ist eine Gruppe von Makroblöcken. Die Größe eines Slice ist dabei im MPEG-Standard nicht festgelegt. Die Bewegungsvorhersage ist auf den Bereich eines Slices begrenzt. In jedem Slice wird die benutzte DCT-Quantisierungsmatrix hinterlegt.
- Makroblock-Ebene
  In einem Makroblock sind 4 Blöcke zusammengefasst. Für sie wird der Bewegungsvektor gespeichert.
- Block-Ebene
  Hier werden für einen 8 x 8 -Pixel Block die Bilddaten gespeichert.

Systemschicht

In der Systemschicht (Abb.4.4) werden die Audio- und Videodaten zu einem Bitstrom zusammengefasst.

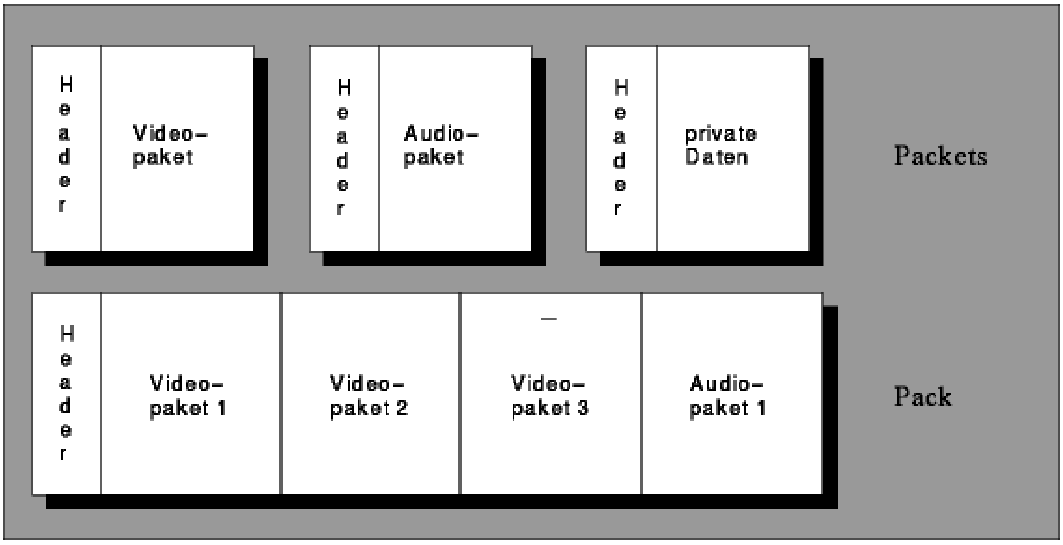

Abbildung 4.4.: Systemschicht MPEG Strom (aus [Buc94])

Sie hat ihrerseits wiederum Teilschichten, die *Pack*-Schicht und die *Packets*-Schicht. Die Packets enthalten neben den Headerinformationen nur Daten einer einzigen Art, also Audio oder Video oder private Daten. Mehrere solcher Packets werden dann zu Packs aneinandergereiht. Die Header der einzelnen Packets bleiben erhalten. Zusätzlich existieren Zeitstempel für die Synchronisation (*Presentation Time Stamps*), welche einen maximalen zeitlichen Abstand von 0,7 Sekunden aufweisen dürfen. Sowohl die Packets als auch die Packs sind nach dem TLV-Prinzip organisiert.

4.4. Zusammenfassung

Wie an den Beispielen deutlich wurde, besitzen Multimedia-Formate bezogen auf die Verwaltung der Medienquanten verschiedenartigster Medienströme durchaus eine sehr komplexe Struktur.
Trotzdem kann man den Aufbau von Teilen dieser Multimedia-Formate immer wieder auf das TLV-Prinzip zurückführen. Deshalb werde ich bei den Überlegungen zur Verarbeitung und zum Transport der Medienquanten dieses vereinfachte Modell als Grundlage benutzen.

5. Das Modell der schwankungsbeschränkten periodischen Ströme

Um die geforderte Datenunabhängigkeit zu erreichen, werden Konverter eingesetzt. Je nach Medienstrom und den zugrundeliegenden Ein- und Ausgabeformaten werden mehrere solcher Konverter benötigt, die eine Konverterkette bilden und mit dem Modell der schwankungsbeschränkten periodischen Ströme (sbS) beschrieben werden.

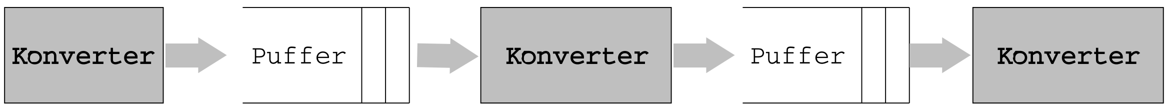

Abbildung 5.1.: Konverterkette

Bezüglich des Aufwandes der Abschätzung und bezüglich der Ressourcenausnutzung zur Laufzeit bewegt sich dieses Modell zwischen der Worst-Case-Annahme und einer vollständigen quantenweise Analyse des Medienstromes. Die Worst-Case-Annahme ist bzgl. der benötigten Ressourcen zur Laufzeit sehr teuer. Eine komplette Analyse jedes Medienstromes verursacht ebenfalls hohe Kosten und ist bei fortlaufend andauernden Strömen garnicht möglich.

Grundlage des Modelles ist die Beziehung zwischen Produzenten und Konsumenten von Medienquanten.

Dafür werden jeweils ein **Volumenstrom**(Abb. 5.3) und ein **Zeitstrom**(Abb. 5.2) beschrieben. Merkmale des Zeitstromes $(T, D, \tau, t_{0,P})$ sind:

T	Periode	
D	minimaler Abstand	minimaler zeitlicher Abstand zwischen zwei Quanten
τ	maximale Abweichung	maximale zeitliche Schwankung bei der Verarbeitung
$t_{0,P}$	Startzeitpunkt	

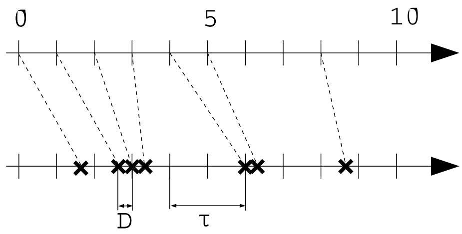

Abbildung 5.2.: Zeitstrom (nach [HMMW01])

und Merkmale eines Größenstromes $(S, M, \sigma, s_{0,P})$ sind:

S	durchschnittl. Quantengröße	
M	minimale Quantengröße	
σ	maximale Abweichung	maximale Abweichung von der kumulierten Quantengröße
$s_{0,P}$	Startwert	

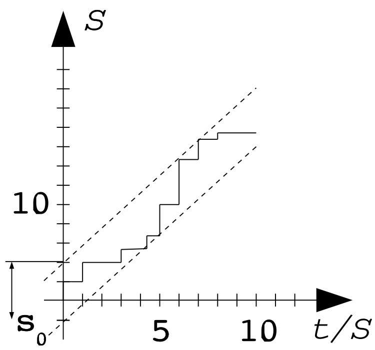

Abbildung 5.3.: Volumenstrom (nach [HMMW01])

Bezogen auf die Konverterkette sind alle bis auf den ersten und den letzten Konverter sowohl Produzent als auch Konsument. Für jede Produzent-Konsument-Beziehung erfolgt eine separate Betrachtung mit dem o.g. Modell. Die durchschnittliche Datenrate von Produzent und dazugehörigem

Konsument muss natürlich gleich sein.
Die Ergebnisse der Berechnung nach [HMMW01] sind die zwischen zwei Konvertern benötigte *Puffergröße* und die *Vorlaufzeit* des Produzenten vor dem Konsumenten. Wie diese zwei Parameter in die Planung der gesamten Konverterkette eingehen, wird in Kapitel 6.5.1 beschrieben.

6. Abstraktion von Datenübergabe zur Quantenübergabe

6.1. Einordnung dieser Arbeit

Aufbauend auf dem sbS-Modell und auf den Vorbetrachtungen zu Konvertern und Multimediaformaten soll nun eine Schnittstelle geschaffen werden, auf deren Basis Konverter in die Echtzeitumgebung von DROPS integriert werden können. Abb. 6.1 verdeutlicht den Zusammenhang zu anderen Teilbereichen des Projektes memo.REAL .

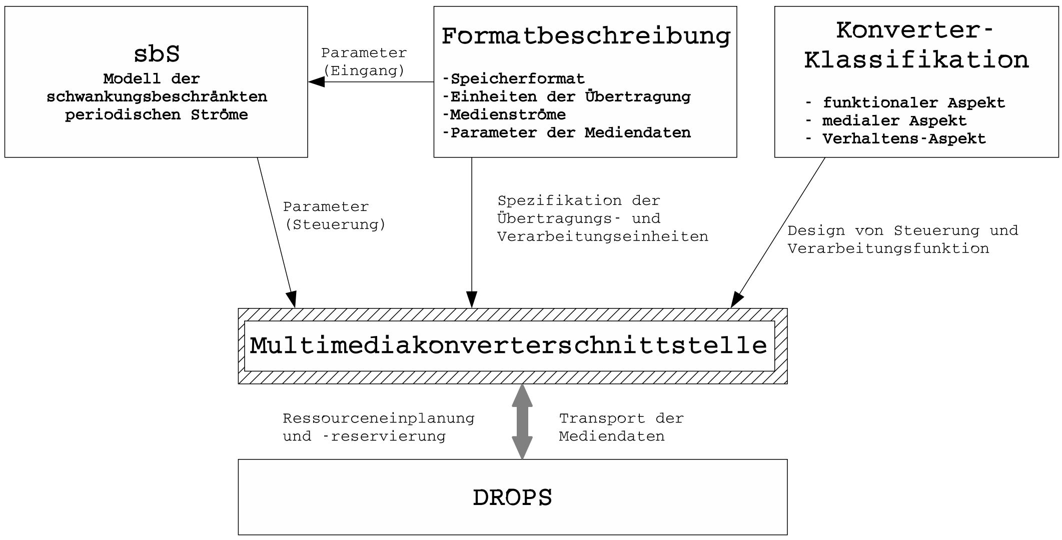

Abbildung 6.1.: Einordnung dieser Arbeit in das Projekt memo.REAL

Wie schon in Kapitel 2.2 erwähnt, wird DROPS die Grundlage für diese Schnittstelle bilden. Hinsichtlich des Verarbeitungsablaufes sollen vorerst Konverter mit der Eigenschaft „Eingangstakt gleich Ausgangstakt" betrachtet werden, ausgehend davon kann aber auf beliebige Konverter geschlossen werden.

Bezüglich der Formatbeschreibung wird mit der Vereinfachung gearbeitet, dass die Basis der Medienformate bzw. der Medien- und Multimediadateien das Tag-Length-Value - Prinzip (TLV) ist. Es werden mehrere Ansätze vorgestellt, welche die Verarbeitung komplexerer Strukturen inklusive mehrerer Medienströme auf Basis von TLV zulassen.

6.2. Anforderungen an die Multimediakonverterschnittstelle

Folgende Anforderungen an die zu entwickelnde Schnittstelle erscheinen nach den Vorbetrachtungen der vergangenen Kapitel sinnvoll:

- einfache Basis für die Implementierung verschiedenartigster Konverter in gemeinsamer Umgebung
- Herauslösen von zentraler Steuerschleife und/oder Verarbeitungsfunktion: Schaffung einer Konvertervorlage
- Betriebssystemumgebung DROPS
- Benutzung von DSI zum Datenaustausch zwischen den Konvertern
- entstehende Datenströme sollen mit dem sbS-Modell beschrieben werden

6.3. Datenfluss

6.3.1. Verarbeitung in den Konvertern

In den einzelnen Konvertern werden die Mediendaten nicht byteweise, sondern in Einheiten von Quanten verarbeitet. Ziel der Formatbeschreibung ist es, diese Quanten zu identifizieren.

Wenn der Eingangstakt eines Konverters gleich dem Ausgangstakt ist, sind die einzelnen Quanten also die kleinsten Einheiten, die ein Konverter unanhängig von anderen Mediendaten verarbeiten kann.

Die Zerlegung des Medienstromes in derartige Quanten erscheint aufgrund des Modelles der schwankungsbeschränkten Ströme sinnvoll: Wenn die Größe eines Datenpaketes nicht der eines Quanten entspricht, müssen die Daten entweder im Konverter gepuffert werden oder es werden mehr Daten gelesen, als der Konverter im aktuellen Verarbeitungsschritt benötigt.

Mit diesem Ansatz ist der Umfang eines Quanten also stark von dem jeweiligen Kodierungsformat abhängig:

Bezogen auf Videoströme können Quanten durchaus einzelne Frames sein, vorausgesetzt das Video ist unkomprimiert bzw. nur intra-frame-kodiert. Bei MPEG-Video mit P- und B-Frames kann der Quant sinnvoller Weise nur ein Group-of-Pictures (GOP) sein, da der GOP eine in sich geschlossene, unabhängig dekodierbare/umkodierbare Einheit darstellt.

Es müssen also verschiedene zu unterstützende Formate benannt werden und dazu jeweils die als Quant in Frage kommenden Einheiten spezifiziert werden.

6.3.2. Transport zwischen den Konvertern

Aufgrund der oben genannten Verarbeitungseinheiten liegt es nahe, genau diese auch als Einheiten für den Datentransport zwischen den Konvertern zu benutzen. Bezogen auf dieser Quanten (und deren Schwankung in Größe und Ankunftszeitabständen) werden die zwischen zwei Konvertern benötigte Puffergröße sowie die Vorlaufzeit eines Konverters nach [HMMW01] ermittelt.

Wie dieses Vorgehen auf das DSI abgebildet wird, ist ausführlich in Kapitel 7 beschrieben.

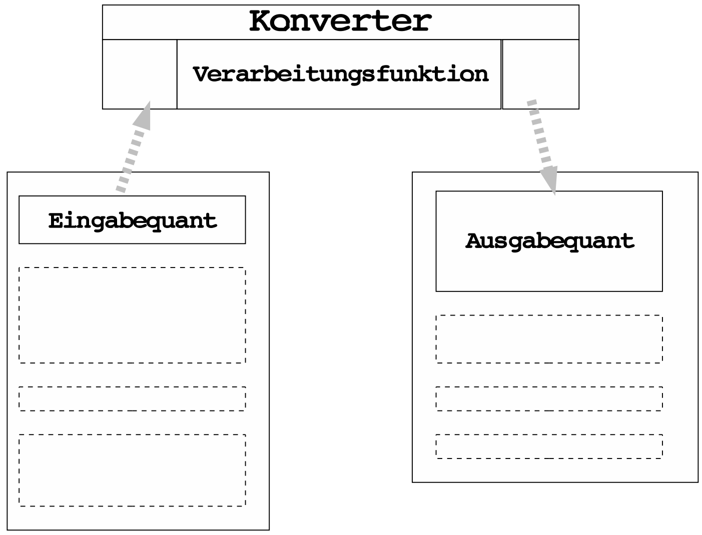

Abbildung 6.2.: Verarbeitung in den Konvertern

6.3.3. Speicherung der Mediendaten

Ausgehend von den Überlegungen zur Verarbeitung und zum Transport der Quanten sollen nun Ansätze der Datenspeicherung im Dateisystem vorgestellt werden.

Einfacher Ansatz

Der nächstliegendste Gedanke wäre, die Einheiten von Verarbeitung und Transport auch im Dateisystem abzulegen. Den Grundgedanken bildet dabei wie bereits angedeutet das Tag-Length-Value - Prinzip.

Dazu müsste für die verschiedensten Multimediaformate eine passende Format-/Quantenbeschreibung für die Speicherung entwickelt werden. Es ware außerdem nötig, jede Multimedia-Datei einer entsprechenden Konvertierung[1] zu unterziehen, bevor sie im Dateisystem abgelegt wird.

Dies bringt den offensichtlichen Nachteil mit sich, dass eine Multimedia-Datei, wenn sie in der Konverterumgebung mit zwei verschiedenen Quantengrößenbeschreibungen verarbeitet werden soll, auch zweimal im Dateisystem abgelegt werden müsste.

Benutzung einer Indexdatei für mehr Flexibilität

Ein anderer Ansatz wäre, die Multimedia-Datei in ihrem ursprünglichen Format in dem Dateisystem abzulegen und zugehörig zu dieser Multimedia-Datei eine oder mehrere Spezifikationen (Indexdateien) zu verwalten, die eine Quantenzerlegung der entsprechenden Datei beschreiben.

Der CSI-Adaptor - die Aufgabe des ersten Konverters: Ein erster Konverter der Kette (CSI-Adaptor) hätte dann die Aufgabe, aus dem vom Dateisystem gelieferten Byte-/Blockstrom der Medien-

[1] Konvertierung von dem (proprietären) Dateiformat in die gewünschten Quantenzerlegung

Datei die für den Transport nötigen Quanten bereitzustellen. (Abb. 6.3)

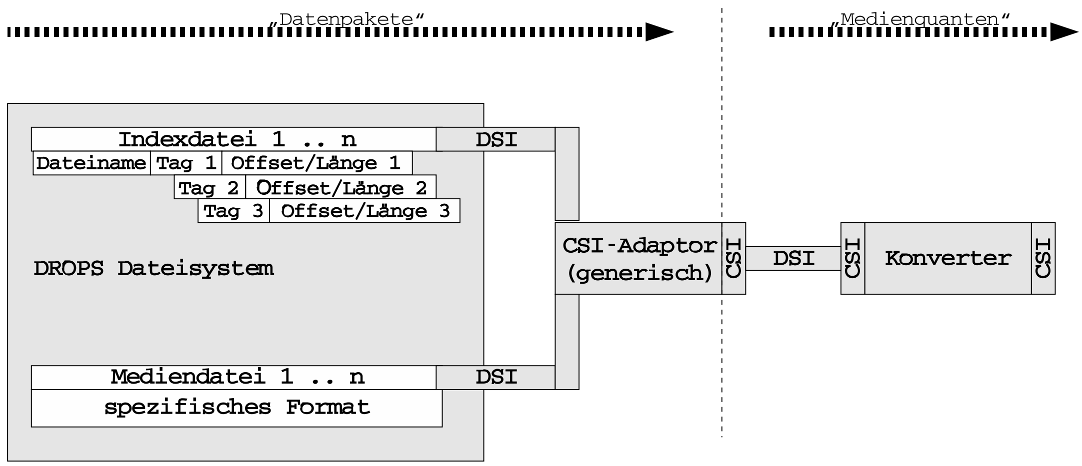

Abbildung 6.3.: Multimedia-Datei, Indexdatei und generischer CSI-Adaptor

Der Vorteil liegt in der Flexibilität, zu einer Multimedia-Datei mehrere sogenannte Zerlegungsbeschreibungen zu verwalten und bei Bedarf die gewünschte Zerlegung anzuwenden.

Eine ähnliche Funktionalität kann erreicht werden, wenn für die Multimedia-Datei zwar keine Indexdatei zur Verfügung steht, jedoch ein speziell für ihr Speicherformat und für die gewünschte Zerlegung entwickelter CSI-Adaptor. (Abb. 6.4)

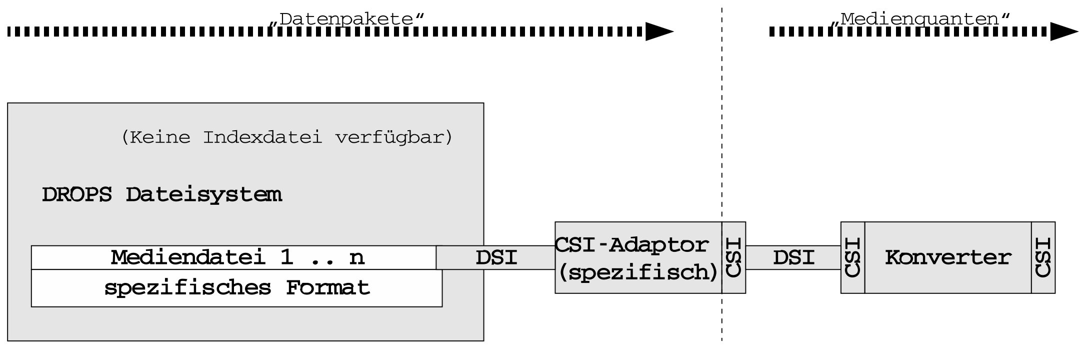

Abbildung 6.4.: Multimedia-Datei, spezifischer CSI-Adaptor

Vorteile für alle folgenden Konverter: Unabhängig davon, ob es sich um einen generischen oder um einen spezifischen CSI-Adaptor handelt, bleibt allen folgenden Konvertern die Sicht auf die im Dateisystem abgelegte Mediendatei verborgen.

Es zählt nicht zu den Aufgaben des DROPS Dateisystems, die in ihm abgelegten Multimediadaten zu interpretieren bzw. diese in einer für die Wiedergabe optimalen Form abzulegen. Um dennoch eine Servicequalität in einer Echtzeit-Umgebung realisieren zu können, besteht die Möglichkeit, mit einer vorher vereinbarten Bandbreite auf die Multimediadatei im Dateisystem zuzugreifen. Dieser Zugriff geschieht in Einheiten ebenfalls vorher festgelegter Größe, d.h. die Datei wird Block für Block[2] aus dem Dateisystem übertragen.

Der CSI-Adaptor als erster Konverter der Konverterkette hat nun die Aufgabe, aus den Blöcken des Dateisystems die Medienquanten zu identifizieren und herzustellen. Damit kommt zu der zeitlichen Schwankung der Blöcke vom Dateisystem[3] noch die Schwankung in der Blockgröße. Beides wird durch das sbS-Modell (Kapitel 5) beschrieben.

Der Vorteil für nachfolgende Konverter ist die Datenübergabe in Form von Medienquanten. Die Konverter können bezogen auf ihre Eingabe- und Ausgabepuffer direkt auf einem oder mehreren dieser Quanten operieren.

Der in Kapitel 3.2 vorgestellte dsi-mpegview Konverter kann die Rolle eines solchen spezifischen Konverters übernehmen. Er liest eine MPEG-Datei aus dem Dateisystem und dekodiert diese zu Einheiten von Frames bzw. GOPs, welche dann von anderen Konvertern weiterverarbeitet werden können.

Verallgemeinerung: Umgang mit Strömen, Multiplexer und De-Multiplexer

Im Folgenden soll zwischen den Begriffen „Mediendatei“ und „Multimediadatei“ unterschieden werden. Während eine Mediendatei lediglich aus einem Strom von Quanten besteht, setzt sich eine Multimediadatei aus mehreren Strömen zusammen. In den obigen Ausarbeitungen wurden also Multimedia-Dateien vernachlässigt bzw. nicht mit der nötigen Aufmerksamkeit betrachtet.

Wie man in Abb. 6.3 sehen kann, hat der generische CSI-Adaptor eine besondere Rolle in der Hinsicht, dass er zwei Eingabeströme und nur einen Ausgabestrom verwaltet. Die Daten des zweiten Eingabestromes (von der Indexdatei) fließen indirekt in den Ausgabestrom ein; sie beschreiben die „Quantisierung“ des Stromes der Mediendatei.
Abstrahiert man nun von dieser Gegebenheit, entstehen völlig neue Möglichkeiten der Bildung von Konverterketten und des Umgangs mit Medienströmen:
Der CSI-Adaptor kann als Multiplexer zweier Ströme betrachtet werden. Verallgemeinert heißt das, dass es sowohl Multiplexer mit mehr als zwei Eingabeströmen als auch De-Multiplexer geben soll, die einen Eingabestrom nach einer bestimmten Spezifikation in mehrere Ausgabeströme zerlegen.
Die Spezifikation für das Multiplexen / De-Multiplexen muss dabei nicht aus einer Datei (Indexdatei) kommen. Vielmehr kann sich diese Spezifikation in der Verarbeitungsfunktion des Konverters (des Multiplexers /De-Multiplexers) wiederfinden.

Hier wird deutlich, weshalb man die Überlegungen bzgl. des Umgangs mit Strömen und bzgl. des Designs der Konverter auf den allgemeineren Begriff der Multimedia-Datei ausdehnen muss.

Vorteile: Die Konverter müssen nicht mehr spezifisch für das Multimedia-Datei-Format sein, sie sind nur noch spezifisch für die aus dem De-Multiplexen entstehenden „Elementarströme“ . Der Begriff des „Elementarstromes“ muss natürlich noch genauer spezifiziert werden. Zum einen wäre hier eine

[2] jeder Block hat dieselbe Größe
[3] je nach vereinbarter Servicequalität

Unterscheidung der einzelnen Medien(-typen) [4] sinnvoll. Zum anderen ist eine weitere Untergliederung möglich, die es erlaubt, aus „Elementarströmen“ auch auf „Elementarkonverter“ zu schließen. Dies sollen Konverter sein, die den „Elementarstrom“ verarbeiten können, ohne zu wissen aus welchem Mediendatei-Format dieser „Elementarstrom“ entstanden ist.
Ein Beispiel hierfür ist ein Konverter, der aus AC-3 Ton einen PCM Stereo Mix erzeugt. Die Quelle dafür könnte eine DVD gewesen sein (VOB-Datei) oder auch ein DivX-Video.

Ein Beispiel für Multiplexer und De-Multiplexer: In Abb. 6.5 ist ein Anwendungsbeispiel für Multiplexer und De-Multiplexer dargestellt. Die im Dateisystem abgelegte Datei ist eine von einer DVD kopierte VOB-Datei. Diese enthält die Bildinformationen im MPEG2-Format und die Toninformationen im AC-3 und oft auch im PCM Format in mehreren Sprachen. Die Untertitel sind in einer weiteren Datei ebenfalls mehrsprachig abgelegt [5]. Ein erster formatspezifischer De-Multiplexer soll nun die Aufgabe haben, den Bytestrom/Paketstrom der VOB-Datei in einen Videostrom und in einen Audiostrom (englisch, AC-3) zu zerlegen. Der Textstrom wird parallel dazu aus der Untertitel-Datei gelesen. Diese drei Ströme werden nun von verschiedenen Konvertern verarbeitet: Für den Videostrom wird eine Framegrößenkonvertierung durchgeführt, der Audiostrom wird zu PCM-Stereo umkodiert und zusätzlich noch in der Bitrate verändert, die Sprache der Untertitel wird gewählt und die Schriftgröße derselben festgelegt.
Danach werden der Videostrom und der Textstrom der Untertitel gemultiplext und der Ausgabehardware (z.B. Display) zugeführt. Der bearbeitete Audiostrom wird direkt an die Soundkarte geleitet.

Die Synchronisation der verschiedenen Ströme wird hier vorerst außer Acht gelassen.

Inwieweit der Datenfluß durch Konverter mit mehreren Ein- und Ausgabeströmen mit dem Modell der schwankungsbeschränkten Ströme beschreibbar ist, muss noch untersucht werden.

6.4. weitere Betrachtungen

In Kapitel 6.3 wurden die Verarbeitung und der Transport der Mediendaten diskutiert. Davon ausgehend wurden Ansätze für die Speicherung dieser vorgestellt und bewertet.

Alle Ansätze haben gemeinsam, dass die Einplanung der Daten auf Basis von Medienquanten erfolgt, welche eine strombedingt variable Größe haben und in nicht äquidistanten Zeitabständen eintreffen; diese Verhaltensweise wird durch das sbS-Modell beschrieben.

6.4.1. Verallgemeinerung des Konvertermodelles

Wie schon in der Vorbetrachtung von Kapitel 3 angedeutet, muss auch der allgemeine Fall betrachtet werden, dass der Eingangstakt ungleich dem Ausgangstakt ist und eine herausgelöste Verarbeitungsfunktion nicht für alle Konverter ausreichend ist, sondern auch die zentrale Steuerschleife konverterspezifisch sein muss.
Abbildung 6.6 soll die komplexen Anforderungen an die Verarbeitung von mehr als einem Ein- und Ausgabequanten in einem Konverter verdeutlichen.

[4] z.B. Text, Audio, Video, Image
[5] In einer VOB-Datei selbst sind die Untertitel nicht als Text-Strom abgelegt. Deshalb geht dieses fiktive Beispiel von einer zusätzlichen Untertitel-Datei aus.

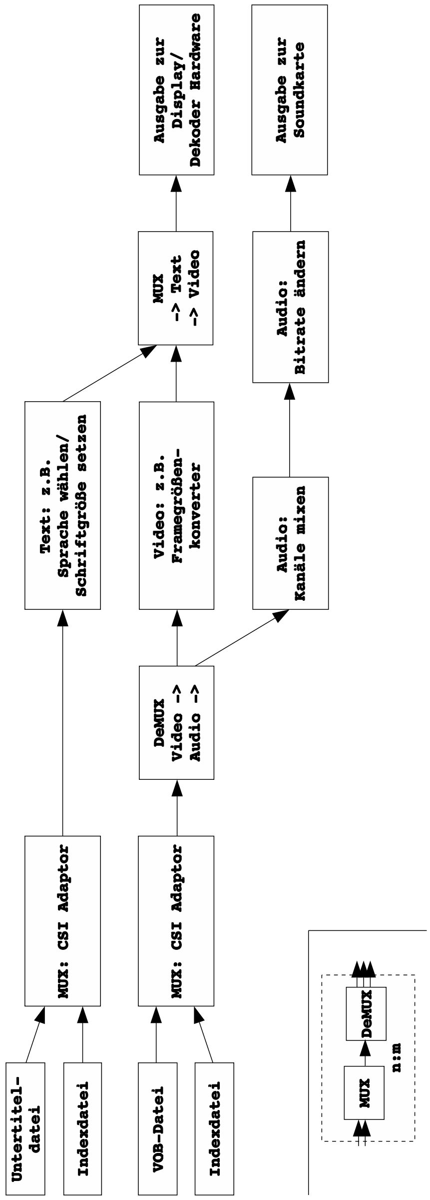

Abbildung 6.5.: Beispiel für den Einsatz von Multiplexern / De-Multiplexern

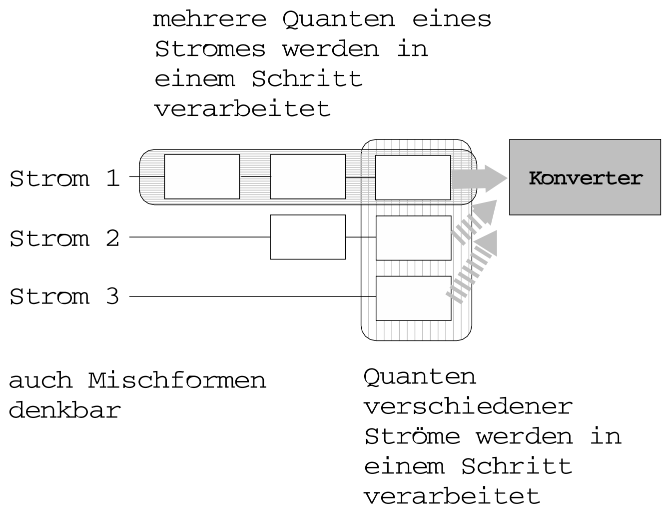

Abbildung 6.6.: Verarbeitung mehrerer Quanten in einem Takt

Es kann dabei vorkommen, dass sowohl mehrere Quanten *eines* Stromes als auch Quanten *mehrerer* Ströme zur Verarbeitung herangezogen werden müssen. Letzteres findet sich in dem beschriebenen CSI-Adaptor bzw. in den Multiplexern wieder, ist in der Beschreibung des sbS-Modelles aber aufwendiger. Ersteres kann dagegen auf eine einfache Art und Weise auf schwankungsbeschränkten Strömen abgebildet werden:

Es wird angenommen, die vom Konverter benötigten Quanten (mehr als einer) werden nicht lokal im Konverter gepuffert, sondern verweilen bis zur Verarbeitung im DSI-Puffer. Offensichtlich muss der Puffer dann größer geplant werden. Dies wird dadurch erreicht, dass als Parameter der Puffergrößenberechnung die Schwankung der Verarbeitungsperiode (zeitlich) sowie die Schwankung des Größenstromes (Volumen) mit erhöhten Werten eingehen.

Während der Laufzeit dieser Diplomarbeit wurde am Datenbankinstitut der TU-Dresden und der Universität Erlangen an verschiedenen Möglichkeiten der Klassifizierung gearbeitet.

Nachfolgend soll eine dieser Klassifizierungen vorgestellt werden und der Zusammenhang zum verwendeten Modell in dieser Arbeit deutlich gemacht werden.

6.4.2. Konverterklassifizierung

In [MMI02] wird dieses Verhalten der Konverter durch eine I/O-Klassifikation beschrieben. Ziele der I/O-Klassifikation sind die Vereinfachung der Implementierung bzw. Portierung von Konvertern sowie die Kapselung von Eingabe, Verarbeitung und Ausgabe mit dem Ziel der höheren Wiederverwendbarkeit von Modulen und der Schaffung von Elementarkonvertern.

Bezüglich der I/O-Eigenschaften sind Konverter nach [MMI02] wie folgt unterteilt:

- **fixed/variable block[6] (FB/VB):** Verarbeitung von Blöcken fester oder variabler

[6] Die Begriffe „Block“ und „Quant“ sind dabei als Synonym zu verstehen

Blockgröße

- **single/multi quant (SQ/MQ):** Verarbeitung eines oder jeweils mehrerer Ein- u. Ausgabequanten
- **single/multi stream (SS/MS):** Verarbeitung der Daten eines oder mehrerer Ein- u. Ausgabeströme

Der in diesem Kapitel vorgestellte CSI-Adaptor besitzt also **FBMQMS-Input** - Eigenschaften und **VBSQSS-Output** - Eigenschaften. Der in der Vorbetrachtung benutzte Framegrößenkonverter hat **FBSQSS-Input** - und **FBSQSS-Output** - Eigenschaften, wogegen der dsi-mpegview ein **FBMQSS-Input** - und ein **VBSQSS-Output** - Konverter ist.

6.5. Steuerung und Kontrolle

Durch die Benutzung des Modelles der schwankungsbeschränkten Ströme sind sowohl die zwischen zwei Konvertern benötigte Puffergröße als auch die Verschiebung des Startzeitpunktes zweier benachbarter Konverter bekannt. Aus diesen Größen lässt sich folgendes ableiten:

6.5.1. Initialisierung und Start

Die Reihenfolge der Konverter ist durch den Aufbau einer Konverterkette festgelegt. Die Puffergrößen werden jeweils zwischen zwei Konvertern entsprechend der sbS-Ergebnisse geplant. Beginnend beim ersten Konverter werden alle Weiteren nach Ablauf der berechneten Vorlaufzeit (relativ zu ihrem Vorgänger) gestartet. Die Semantik des globalen Start-Ereignisses ist in diesem Fall der Start des ersten Konverters der Konverterkette.

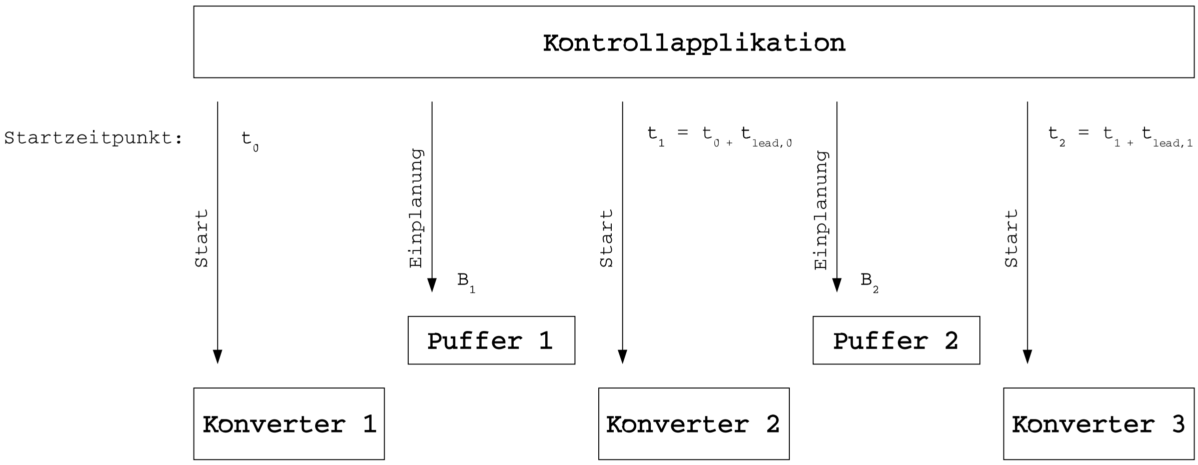

Abbildung 6.7.: Vorlaufzeit und Puffereinplanung

6.5.2. Scheduling

Weiterhin wird jedem Konverter entsprechend der Vorgaben periodisch Rechenzeit zur Verfügung gestellt. Die Bestimmung des Zeitbedarfs sowie dessen Einplanung in Abstimmung mit dem Ressourcenmanager des Betriebssystems soll von der Kontrollapplikation erledigt werden und ist Teil einer weiteren wissenschaftlichen Arbeit am Institut Datenbanken.

6.5.3. Überwachung

Ein wichtiges Kriterium für die erfolgreiche Verarbeitung der Mediendaten durch die Konverterkette ist die Einhaltung der zugesicherten Rechenzeiten gegenüber den einzelnen Konvertern. Wurde auch für jeden Konverter ausreichend Zeit eingeplant, wird kein einzelner Puffer zwischen zwei Konvertern „zu voll“ bzw. „zu leer“ werden und jeder einzelne Konverter kann ohne Verzögerung seine benötigten Daten vom Eingabepuffer lesen und das Ergebnis in den Ausgabepuffer schreiben.

Nun ist es aber durchaus realistisch, dass diese Annahmen verletzt werden. Zum Beispiel könnte einem Konverter vor Ende seines Verarbeitungszyklus' Rechenzeit entzogen werden oder der Konverter braucht für die Bearbeitung eines Quanten länger als er dem Ressourcenmanager mitgeteilt hat.

Da Multimediaanwendungen in die Klasse der Anwendungen mit weichen Echtzeitforderungen fallen, wird dies toleriert.

So ein Fall kann Auswirkungen auf einzelne Konverter oder auf das Gesamtergebnis der Konvertierung haben.

Auf der Ebene des einzelnen Konverters äußert es sich dadurch, dass der Konverter versucht, aus einem bereits leeren Eingabepuffer zu lesen bzw. in einen bereits vollständig gefüllten Ausgabepuffer zu schreiben.
In beiden Fällen kann der Konverter nicht die geplante Arbeit verrichten, sondern muss warten. Es besteht also von Seiten des Konverters die Notwendigkeit, die Kontrollapplikation über eine derartige Situation zu informieren.

Reaktionen der Kontrollapplikation: Je nach Art und Häufigkeit der Signalisierungen der einzelnen Konverter kann die Kontrollapplikation verschiedene Maßnahmen ergreifen, um unter den gegebenen Umständen noch ein zufriedenstellendes Gesamtergebnis zu erreichen.
Ohne hier näher auf die Möglichkeiten der Steuerung eingehen zu wollen, könnte eine derartige Maßnahme zum Beispiel darin bestehen, für einzelne Konverter mehr Rechenzeit einzuplanen oder die gesamte Konverterkette mit einer geringeren durchschnittlichen Datenübertragungsrate neu zu initialisieren.

6.5.4. Ende der Verarbeitung

Für das Ende der Verarbeitung der Daten in der Konverterkette ist ausschlaggebend, wann der letzte Konverter die Daten empfangen und verarbeitet hat. Dieser signalisiert es dann der Kontrollapplikation.

7. Entwurf einer abstrakten Konverterschnittstelle

Mit Hilfe der Erkenntnisse der vorherigen Kapitel soll die Konverterschnittstelle nun entworfen und realisiert werden.

7.1. Programmstruktur innerhalb der DROPS-Umgebung

Es müssen mehrere Konverter sowie eine Kontrollapplikation realisiert werden. Es gibt die Möglichkeit, all diese Teile in einer einzigen Applikation zu implementieren. Dies vereinfacht zwar die Kommunikation und den Datenaustausch zwischen diesen Komponenten, hat jedoch Nachteile bezüglich der Erweiterbarkeit der Konverterumgebung. Schließlich sollen auf einfache und unkomplizierte Art und Weise neue Konverter der Umgebung zugefügt und benutzt werden.
Eine Möglichkeit, diese Erweiterbarkeit auch innerhalb einer einzigen Applikation zu realisieren, wäre das Nachladen von Bibliotheken. Das ist aber z.Z. in DROPS nicht machbar. Deshalb habe ich einen anderen Ansatz gewählt:

Die Kontrollapplikation sowie jeder beteiligte Konverter werden als eigene Applikation realisiert. Die Kommunikation geht auf der Basis von IPC[1] vonstatten, der Datenaustausch wird mittels des DSI realisiert. Abb. 7.1 verdeutlicht diesen Ansatz.

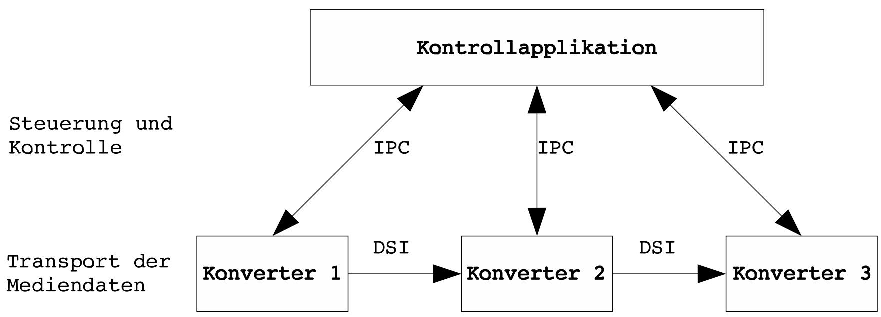

Abbildung 7.1.: allg. Modell: Konverter unter DROPS

Im Folgenden wird das Zusammenspiel der Konverterumgebung mit DROPS erläutert:

Sowohl die benötigten Komponenten des Betriebssystemes als auch die Kontrollapplikation und sämtliche Konverter werden über einen TFTP-Server[2] in den Betriebssystemspeicher geladen. Zum einen geschieht dies beim Aktivieren der DROPS-Umgebung, zum anderen gibt es die Möglichkeit, zur

[1] Interprocess Communication / Interprozess Kommunikation
[2] Trivial File Transfer Protocol - Server: Server zur Dateiablage mit vereinfachtem, reduziertem Befehlssatz

Laufzeit benötigte Programme und Daten mit Hilfe eines Loaders und eines innerhalb der DROPS-Umgebung arbeitenden TFTP-Servers nachzuladen. Letztere Möglichkeit steht erst seit kurzer Zeit zur Verfügung und wird von der hier vorliegenden Implementierung noch nicht genutzt. Eine Einschränkung ist, dass die beteiligten Programmmodule nicht dynamisch gelinkt werden können. Sämtliche Speicherbereiche müssen zu Zeiten der Programmerstellung festgelegt werden. Es ist dann Aufgabe des Programmerstellers, Überlappungen von Speicherbereichen auszuschließen.

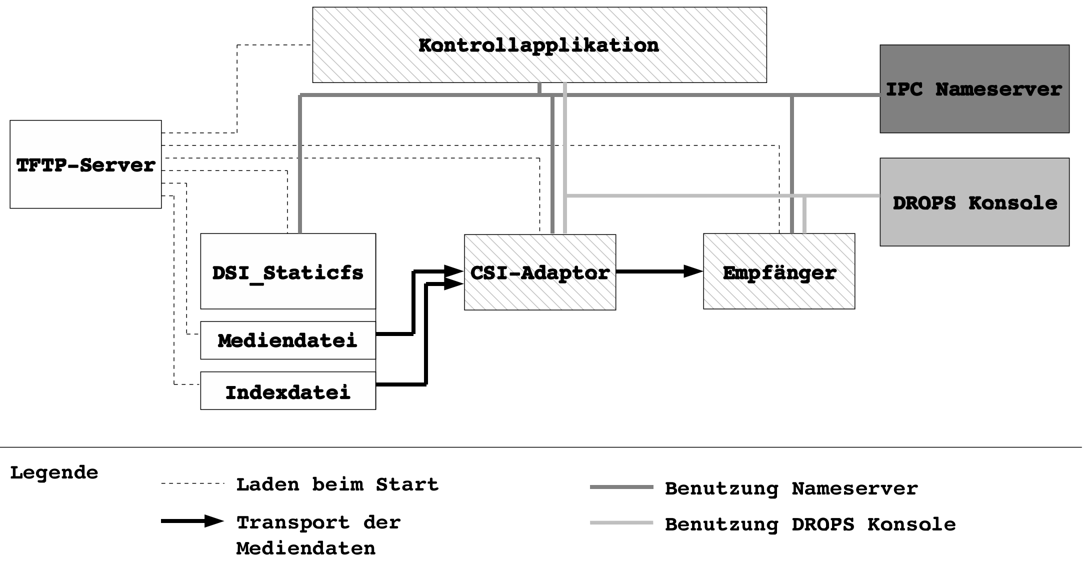

Abbildung 7.2.: Komponenten der Beispielumgebung

Wie in Abb. 7.2 zu sehen ist, werden neben den Programmmodulen auch die Mediendaten zum Zeitpunkt des Betriebssystemstarts über den TFTP-Server in den Arbeitsspeicher geladen. Das liegt darin begründet, dass für die Konverter noch kein Dateisystemzugriff existiert, sondern lediglich eine Ramdisk (StaticFS), aus der sie die Mediendaten lesen und verarbeiten können. Deshalb und weil DROPS momentan nur 256 MB Arbeitsspeicher verwalten kann, wird diese erste Konverterumgebung nur Modellcharakter haben.

Da die Schnittstellen der Ramdisk und die der noch fertigzustellenden Dateisystemanbindung aber ähnlich sein werden, sollte eine Umstellung der Konverterumgebung zu gegebener Zeit nicht allzu problematisch sein.

Beim DROPS Namensserver können die gestarteten Applikationen einen Namen zu ihrer Task-ID[3] registrieren. Über diesen Namen ist es anderen Applikationen später möglich, die Task-ID einer in der DROPS-Umgebung gestarteten Komponente auf Anfrage zu erhalten und mit dieser Komponente Daten auszutauschen bzw. zu kommunizieren.
Für unsere Beispielumgebung ist wichtig, dass alle Konverter ihre Task-ID beim Namensserver hinterlegen. Das StaticFS tut dies sowieso.
Die Kontrollapplikation wird dann während der Initialisierung diese Namen abfragen und ist somit in der Lage, Nachrichten an die Konverter und an das StaticFS zu senden.

[3] Durch die Task-ID wird ein Adressraum identifiziert. Jede unserer Applikationen läuft in einem eigenen Adressraum.

Die DROPS Konsole ermöglicht die grafische Ausgabe von Daten für DROPS Komponenten sowie die Tastatureingabe. Wird die Konsole durch den TFTP-Server geladen und von den Komponenten initialisiert, so erscheint die Ausgabe jeder einzelnen Komponente in einem separaten Konsolenfenster. Da die DROPS-Konsole nur grafische Ausgabe unterstützt, bedient man sich zum Ausgeben von einfachem Text der sogenannten Contxt-Bibliothek, welche eine entsprechende Umsetzung vornimmt.

Konverterkette Wie bereits in Abb. 7.1 angedeutet, werden die Konverter in einer oder mehreren Ketten organisiert sein. In der Modellumgebung ist das erste Glied der Kette die Ramdisk als Datenlieferant.

Es kommen aber neben Dateisystem und Ramdisk auch andere Komponenten als Datenlieferanten in Frage, so z.B. Framegrabberkarte oder Netzwerkanbindung. Für das letzte Glied der Konverterkette gibt es ebenfalls eine Reihe von Möglichkeiten, so z.B. Dateisystem, Anzeigekomponente, Dekoderhardware oder Netzwerk. In der Modellumgebung wird der letzte Konverter lediglich den Empfang der Daten signalisieren.

7.2. Nutzung des DSI

Das DSI stellt eine Schnittstelle für die Kommunikation zwischen Produzent und Konsument bezüglich der Nutzung von gemeinsamem Speicher zur Verfügung. In unserem Fall tauschen die Konverter die Mediendaten in Form von Medienquanten über das DSI aus. Laut [LHR] wurde ursprünglich von Paketen fester Größe und lediglich einer zeitlichen Schwankung der Ankunftsabstände ausgegangen. Dadurch das DSI aber nur die Verwaltung eines gemeinsamen Produzenten-Konsumenten-Speicherbereiches auf unterster Ebene unterstüzt, konnte es problemlos auch für unsere Belange angepasst werden (siehe Kapitel 7.2.2).

DSI bietet eine Reihe von Funktionen zum Senden, Empfangen und Synchronisieren. Damit soll die flexible Nutzung des DSI in den verschiedensten Anwendungsumgebungen gewährleistet werden. Dies soll nun an zwei ausgewählten Stellen gezeigt werden.

7.2.1. Callback-Funktionen

Von einigen Funktionen sind lediglich die Schnittstellen spezifiziert. Die Funktionalität muss der Anwendungsentwickler selbst implementieren. Das betrifft beispielsweise die Mechanismen zur Ereignissignalisierung und Funktionen zur Steuerung der durch einen DSI-Strom verbundenen Komponenten. Die Implementierung dieser sogenannten Callback-Funktionen erfolgt zum Teil in der Kontrollapplikation (Synchronisation) , zum Teil in den Konvertern selbst (Komponentensteuerung). Die Callback-Funktionen sind statische Funktionen. Da aber die Implementierung der Multimediakonverterschnittstelle in C++ - Klassen erfolgen soll (siehe Kapitel 7.3), muss der Aufruf einer statischen Callback - Funktion an eine entsprechende Methode einer Klasseninstanz delegiert werden.

Auf der Seite der Kontrollapplikation werden für jeden Konverter Callback-Funktionen implementiert. Beim Aufruf der Funktionen durch das DSI werden dann die Konverter, welche über die DSI-Ströme „verbunden“ sind initialisiert, gestartet oder gestoppt. Bei jedem Aufruf einer Callback-Funktion in der Kontrollapplikation wird dieser Aufruf an die zugehörige Konverterinstanz weitergeleitet.

Um dies zu realisieren, wird eine Referenz auf die Konverterinstanz dem DSI zusätzlich zu der

Implementierung der Callback-Funktion übergeben. Für diese Referenzübergabe mussten Anpassungen am DSI vorgenommen werden.

Auf der Konverterseite gab es eine einfachere Lösung für das Problem der Instanz-Referenzen: Da pro ausführbarem Programmmodul (Konverter) nur eine Instanz einer C++ - Klasse existierte, konnte die Referenz auf diese Instanz innerhalb der statischen Callback-Funktion durch einen Zugriff auf einen statischen Teil der Konverterklasse selbst realisiert werden.

7.2.2. Umsetzung des Modelles der schwankungsbeschränkten Ströme

In Kapitel 5 wurde das Modell der schwankungsbeschränkten periodischen Ströme eingeführt und wie in Kapitel 6.1 angedeutet, soll dieses Modell die Grundlage für den Datenaustausch zwischen den Konvertern bilden. Nun soll das DSI auf das Modell der sbS abgebildet werden.

Zur Beschreibung des Mediendatenstromes stehen u.a. folgende Parameter zur Verfügung:

- benötigte Puffergröße zwischen zwei Konvertern (in Byte)
- Vorlaufzeit eines Produzenten vor einem Konsumenten
- Größe der Medienquanten (min/durchschnittlich/max)
- Dauer der Verarbeitung eines solchen Medienquanten (min/durchschnittlich/max)

Die Vorlaufzeit und die Verarbeitungsdauer (Periode, Schwankung) gehören zur Steuerung bzw. zum Scheduling. Das Scheduling berührt nicht das DSI. Lediglich für die einzelnen Applikationen[4] muss entsprechend der Berechnung der schwankungsbeschränkten Ströme Rechenzeit reserviert werden.

Zur Zeit können für Anwendungen[5] in DROPS feste Prioritäten vergeben werden. Lt. [HBB+] ist eine flexiblere Gestaltung des Schedulings geplant. Auf Betriebssystemebene soll das wie folgt aussehen: Das Scheduling basiert auf Prioritäten der einzelnen Tasks. Eine Task kann sich für eine Zeitdauer (Periode) eine Anzahl von Zeitscheiben mit einer bestimmten Priorität reservieren. Auf Anwendungsebene ist das ein garantierendes Scheduling ([Tan95]): In der Anwendung wird für eine zeitliche Periode eine absoluter Betrag an Rechenzeit eingeplant und durch DROPS zur Verfügung gestellt.

Freispeicherverwaltung mit DSI

Die Größe der Medienquanten und die berechnete Puffergröße sind nötig, um den DSI-Puffer zu planen und zu verwalten.

Zu jedem DSI-Pufferbereich existiert eine DSI-Kontrollstruktur. Dort sind alle zu einem Zeitpunkt im Puffer befindlichen DSI-Pakete mit Paketnummer, Startadresse und Paketgröße erfasst.

Ebenso wie die Größe des DSI-Pufferbereiches wird die maximale Anzahl der Einträge in der Kontrollstruktur während der DSI-Initialisierung festgelegt.

Anhand dieser Kontrollstruktur führt DSI auch die Freispeicherverwaltung des Pufferbereiches durch: Ist kein Kontrollstruktureintrag mehr vorhanden, bedeutet dies, der Puffer ist leer. Sind bereits

[4] für Teile davon, für die sogenannten „Work-Threads“
[5] für einzelne Threads

maximal viele (siehe Initialisierung Kontrollbereich) Kontrollstruktureinträge (und damit DSI-Pakete) vorhanden, wird auch der Pufferbereich als voll angenommen.

Da die Quanten des abzubildenden Medienstromes eine variable Größe haben dürfen, reicht die Freispeicherverwaltung des DSI in unserem Fall nicht aus, da über die Paketanzahl nicht auf die Belegung des DSI-Puffers geschlossen werden kann.

Es wurde deshalb eine eigene Freispeicherverwaltung auf Ebene des tatsächlich benutzen Speicherplatzes implementiert.

Da in den Konvertern bereits vor Beginn der Verarbeitung klar sein muss, ob für das kommende Paket noch genügend Speicher im Puffer zur Verfügung steht oder nicht[6], muss die Paketanforderung (und damit die Freispeicherprüfung) mit der maximal zu erwartenden Paketlänge durchgeführt werden.

Damit die Anforderung des Produzenten bzgl. freien Speicherbereiches im Puffer nicht unbegründet abgelehnt wird, muss der DSI-Puffer um eine maximale Paketlänge größer sein, als aus der sbS-Berechnung hervorgeht[7].

Umsetzung des Ringpuffermodells

Der verwendete DSI-Pufferbereich unterscheidet sich von dem im sbS-Modell angenommenen Ringpuffer dadurch, dass der DSI-Puffer sehr wohl eine feste Anfangsadresse und ein feste Endadresse besitzt. Folgendes Problem tritt nun zu Tage: Im DSI-Puffer wäre theoretisch genügend Speicherplatz für ein weiteres Paket maximaler Länge verfügbar, allerdings befindet sich ein Teil dieses Speicherplatzes am „oberen Ende“ , der andere Teil am „unteren Ende “ des DSI-Pufferbereiches. Da das Paket in seiner Gesamtheit von der Verarbeitungsfunktion des Produzenten geschrieben und von der Verarbeitungsfunktion des Konsumenten gelesen werden soll, kommt eine Teilung des Paketes zunächst nicht in Frage.

Eine Lösung wäre wiederum die Vergrößerung des Pufferbereiches um eine maximale Paketlänge. Dadurch, dass der DSI-Puffer aufgrund der Speicherreservierung des Produzenten bereits größer als durch das sbS-Modell berechnet eingeplant wurde, wird an dieser Stelle kein Problem auftreten.

7.2.3. Senden und Empfangen mit DSI

In Kapitel 7.5 ist der Programmablauf ausführlich beschrieben. Dazu sind in Abb. 7.7 die Schnittstellen zu DSI abgebildet.

7.2.4. Zusammenfassung

Das DSI stellt eine gute Grundlage für den Einsatz von Multimediakonvertern zusammen mit dem Modell der schwankungsbeschränkten periodischen Ströme dar. An vielen Stellen konnten die DSI-Schnittstellen in gekapselter Form problemlos genutzt werden, an manchen Stellen war anwendungsspezifische Implementierung notwendig, durch das DSI vorgesehen und unterstützt.

[6] andernfalls kann die Ausgabe der Verarbeitung nicht direkt in den Ausgabepuffer geschrieben werden, sondern müsste lokal gespeichert und dann kopiert werden

[7] Die Anforderung würde abgelehnt werden, wenn kein maximal großes Paket mehr im Puffer Platz findet. Da das folgende Paket des Produzenten jedoch höchstwahrscheinlich eine kleinere Größe als die Maximale aufweisen wird, muss in diesen Fällen der Anforderung stattgegeben werden können

7.3. Objektorientiertes Modell

Obwohl sämtliche benutzte DROPS-Komponenten in C implementiert sind, habe ich mich entschlossen, die Konverterumgebung als objektorientierte Struktur zu entwerfen. Das hatte folgende Gründe:

- Möglichkeiten der Vererbung
- Möglichkeiten des Überschreibens einzelner Methoden in Unterklassen, dadurch Spezialisierung
- übersichtliche Programmstruktur, dadurch geringerer Einarbeitungsaufwand für den Anwendungsentwickler

Die entwickelte Multimediakonverterschnittstelle ist dabei teilweise in C++, teilweise in C implementiert, um die Komponenten der DROPS-Umgebung entsprechend nutzen zu können. Der Anwendungsentwickler als Nutzer dieser Multimediakonverterschnittstelle muss *nur* auf den in C++ implementierten Schnittstellenteil zugreifen.

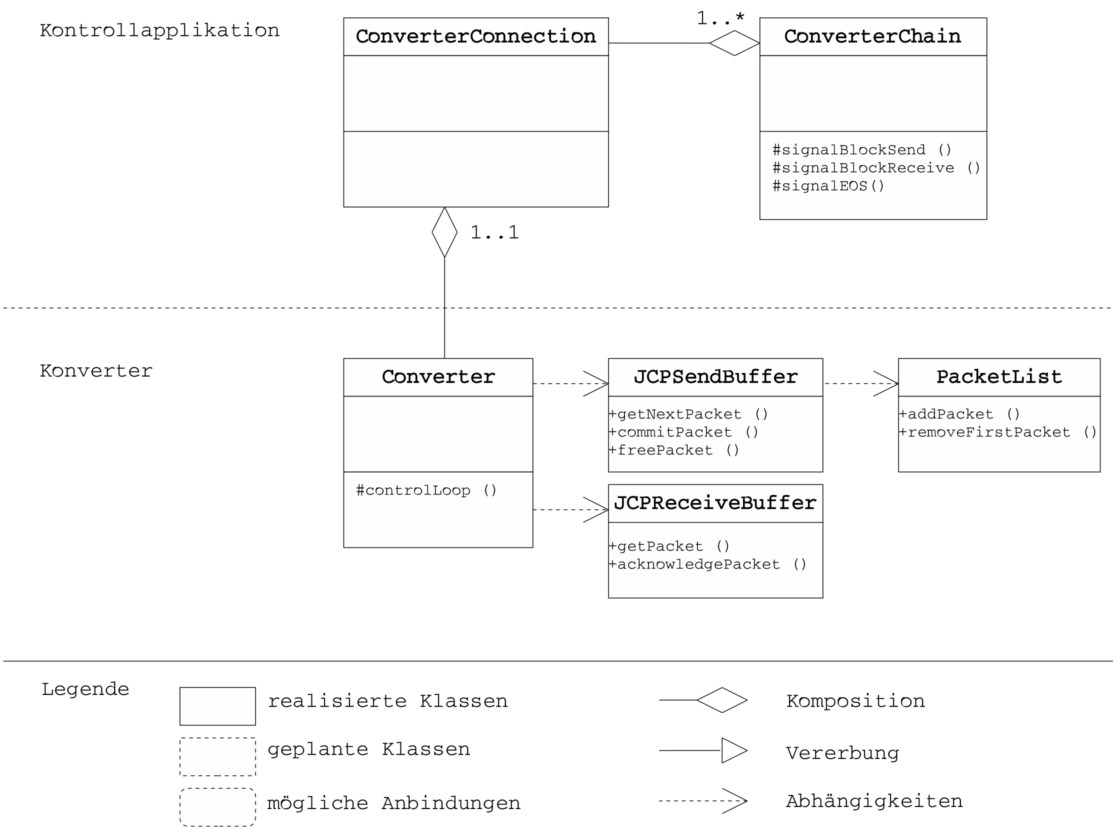

Abbildung 7.3.: objektorientiertes Anwendungsmodell

Anhand des objektorientierten Modelles in Abb. 7.3 werden die entstandenen Klassen und ihre Beziehungen untereinander erläutert:

ConverterChain: Auf der Seite der Kontrollapplikation verwaltet diese Klasse mehrere Verbindungen zu existierenden Konvertern. *ConverterChain* ist eine abstrakte Klasse. In jeder Kontrollapplikation müssen Methoden zur Ereignisbehandlung überschrieben werden.

ConverterConnection: Jede Instanz dieser Klasse repräsentiert die Verbindung zu einem Konverter. Für jeden Konverter werden hier eine Reihe von DSI-Callback - Funktionen hinterlegt, derer sich das DSI zur Steuerung des Datenstromes bedient.

Converter: Die *Converter*-Klasse ist abstrakt. Sie schafft die Rahmenbedingungen für sämtliche zu implementierende Konverter und bietet durch Vererbung und Überschreiben Möglichkeiten der spezifischen Konverter-Implementierung. Die Aufgaben dieser Klasse liegen dabei hauptsächlich in der Initialisierung des Konverters und des DSI. Außerdem wird mit *controlLoop ()* eine Schnittstelle zu einer zentralen Steuerschleife geboten.

JCPSendBuffer: Der *JCPSendBuffer* kapselt alle zum Senden eines Datenpaketes notwendigen DSI-Funktionsaufrufe und übernimmt die Freispeicherverwaltung (voller Puffer) des reservierten DSI-Puffers. Falls das zu sendende Paket / der zu sendende Quant keinen Platz mehr im Puffer findet, wird blockiert und dieses Ereignis an die Kontrollapplikation gemeldet.

PacketList: Die *PacketList* wird vom *JCPSendBuffer* zur Freispeicherverwaltung genutzt. Eine Packetlist stellt dabei eine verkettete Liste von Beschreibungen der im Puffer befindlichen Datenpakete dar.

JCPReceiveBuffer: Als Äquivalent zu *JCPSendBuffer* kapselt diese Klasse die zum Empfangen von Daten notwendigen DSI-Funktionsaufrufe. Wenn durch das DSI ein leerer Puffer signalisiert wird, blockieren diese Funktionsaufrufe und signalisieren an die Kontrollapplikation.

7.3.1. Möglichkeiten der Implementierung spezifischer Konverter

Einige Beispiele für spezifische Konverter sind in Abb. 7.4 dargestellt.

Filter: In dieser Klasse wurde die zentrale Verarbeitungsschleife so gestaltet, dass in jedem Verarbeitungszyklus genau ein (Eingabe-)Quant gelesen und ein (Ausgabe-)Quant geschrieben wird (SQ-Konverter[8]). *Filter* ist ebenfalls abstrakt - weitere Unterklassen müssen nicht mehr die gesamte zentrale Steuerschleife implementieren, sondern lediglich die herausgelöste Verarbeitungsfunktion *processQuant()*.

Sender, Receiver: Diese zwei Unterklassen wurden für Testzwecke erstellt. Der *Sender* sendet fiktive Datenpakete fester/variabler Größe. In einer Konverterumgebung wird später immer das Dateisystem / die Ramdisk die Datenquelle darstellen. Der *Receiver* empfängt beliebige Datenpakete und zeigt dies lediglich an. In der späteren Konverterumgebung werden an seiner Stelle Anzeigekomponenten, Netzwerkübertragungskomponenten oder Dateisystemkomponenten (zur Abspeicherung) stehen. Die zentralen Steuerschleifen wurden an die jeweilige Funktion angepasst.

[8] single-quant - Konverter (nach [MMI02])

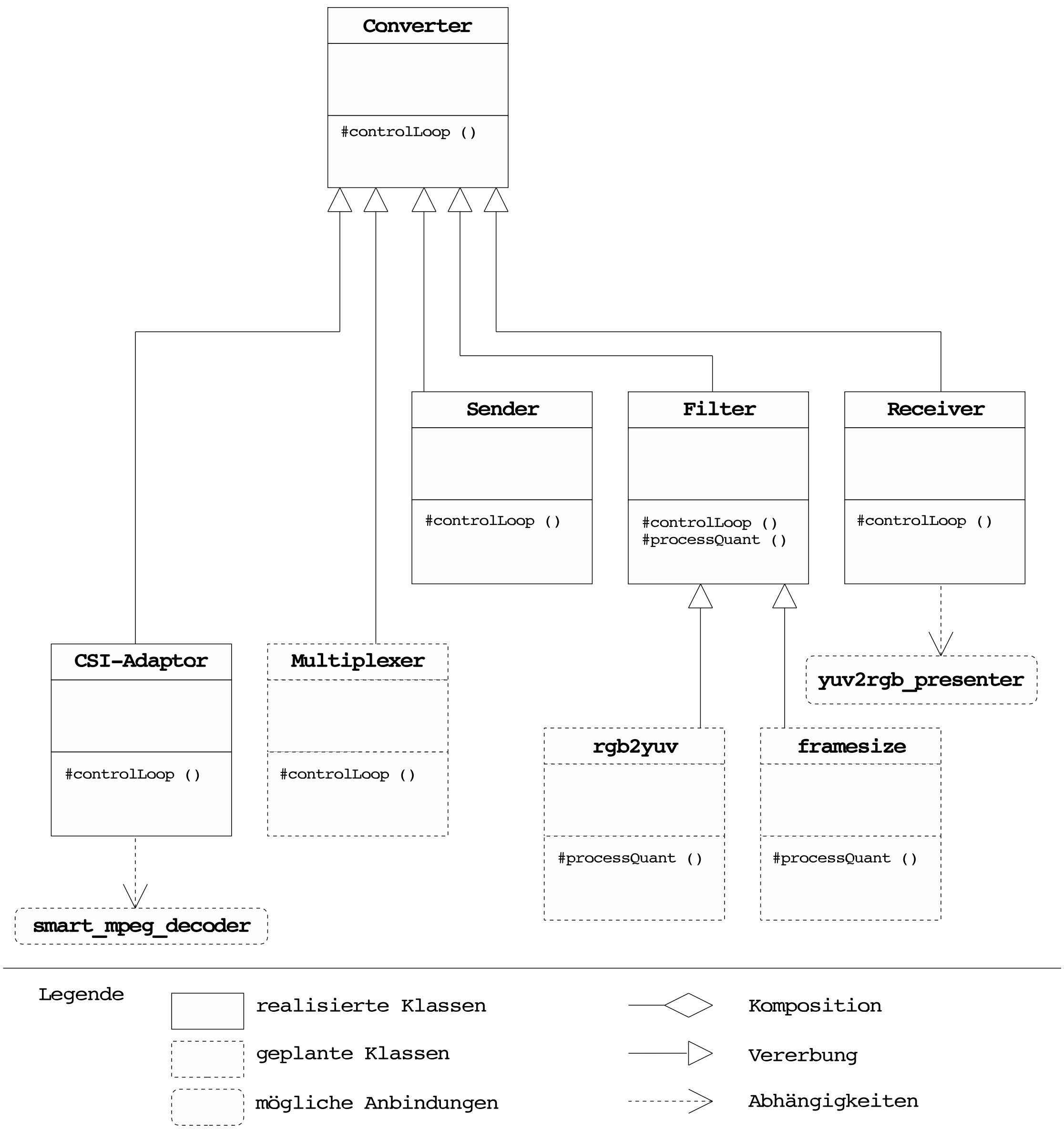

Abbildung 7.4.: Beispiele für spezifische Konverter

CSI-Adaptor: Der *CSI-Adaptor* übernimmt in der Beispielimplementierung die Aufgabe, aus der im Dateisystem abgelegten Mediendatei durch Zuhilfenahme einer ebenfalls im Dateisystem abgelegten Indexdatei entsprechende Quanten zu identifizieren und diese Quanten für die weitere Verarbeitung durch folgende Konverter der Konverterkette zur Verfügung zu stellen. Die zentrale Steuerschleife liest die Daten zweier eingehender Datenströme, verarbeitet diese und erstellt als Resultat einen ausgehenden Datenstrom.

7.3.2. Zusammenfassung

In Anlehnung an Kapitel 3.3 wurden durch das o.g. objektorientierte Programmiermodell Möglichkeiten für die Implementierung verschiedenartigster spezifischer Konverter geschaffen. Die Unterklassen von *Filter* realisieren dabei vorzugsweise single-quant - Konverter, wogegen es mit Unterklassen von *Converter* möglich ist, jede Art von Konverter zu realisieren.

Um multi-quant-Konverter und multi-stream-Konverter sinnvoll implementieren zu können, sollte ähnlich der *Filter*-Klasse eine abstrakte Klasse geschaffen werden, die es dem Anwender erleichtert, in einem Verarbeitungsschritt sowohl mit mehreren Quanten eines Stromes als auch mit Quanten verschiedener Ströme umzugehen.

7.4. Interprozess-Kommunikation

Wie bereits in den vorigen Abschnitten angedeutet, sind sämtliche Komponenten der Konverterumgebung als eigenständige Applikationen implementiert. Die Kommunikation zwischen Diesen erfolgt durch die in DROPS verfügbaren IPC[9]-Mechanismen.
Es wäre ein Mühsames, die gesamte Kommunikation aufbauend auf den L4-IPC-Aufrufen zu realisieren. Deshalb wird im Rahmen sämtlicher DROPS-Entwicklungen Flick[10] als IDL[11]-Kompiler benutzt. Die Definition der Funktionsaufrufe erfolgt in CORBA-IDL[12].

Zu spezifizieren sind dabei lediglich die gewünschten Funktionsaufrufe mit Namen, Parametersatz und entsprechenden Datentypen. Hier ein Beispiel einer derartigen IDL-Beschreibung für zwei ausgewählte Funktionsaufrufe der Kontrollapplikation an den Konverter:

```
...
long open_input(in converter_dataspace_t input_ctrl_ds,
                in converter_dataspace_t input_data_ds,
                out converter_socket_ref_t input_s);


long open_output(out converter_socket_ref_t output_s,
                 out converter_dataspace_t output_ctrl_ds,
                 out converter_dataspace_t output_data_ds,
                 in long buffer_size,
                 in long preload_time);
...
```

[9] Interprocess Communication
[10] Flexible IDL Compiler Kit
[11] Interface Definition Language
[12] Common Object Request Broker Architecture IDL

Davon ausgehend generiert Flick für eine (unidirektionale) Kommunikationsbeziehung jeweils einen Client-Stub und einen Server-Stub, in unserem Fall als C-Programmcode. Der IPC-Server (in diesem gezeigten Beispiel ist das der Konverter) importiert dann die Schnittstellen für die angegebenen Funktionsaufrufe und implementiert dazu die gewünschten Inhalte. Diese Zusammenhänge sind in Abb. 7.5 verdeutlicht.

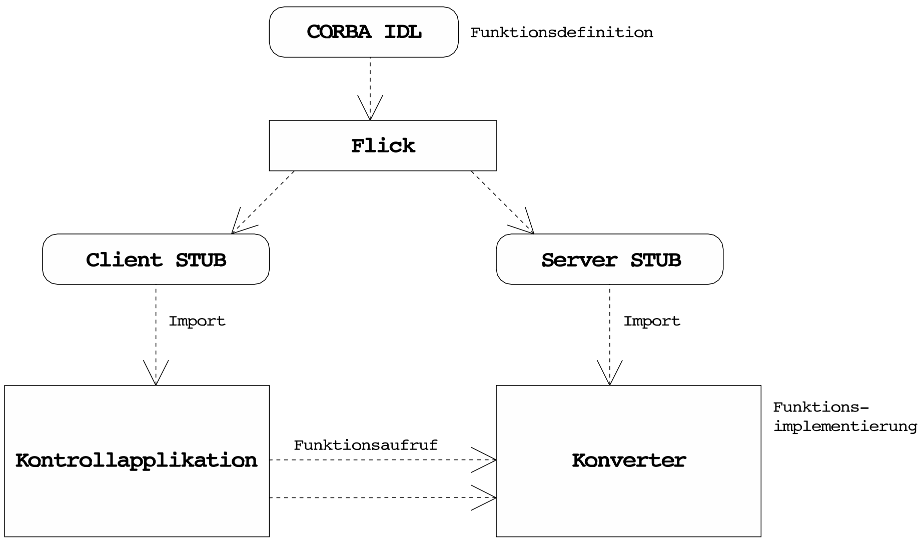

Abbildung 7.5.: IPC mit Flick

7.5. Programmablauf

7.5.1. Initialisierung und Start

Abb. 7.6 stellt schematisch die Zusammenhänge zwischen den einzelnen Programmkomponenten während des Startvorganges der Konverterkette dar. Zu beachten ist dabei die Semantik der Start-Funktion: In diesem Fall stellt das Start-Signal der Anwendung den Start des ersten Konverters dar, d.h. der erste Produzent beginnt sofort zu arbeiten, der erste Konsument nach Verstreichen seiner Vorlaufzeit usw. Für den Anwender bedeutet dies: Nachdem er als Nutzer den Start der Konverterkette initiiert hat, vergeht eine gewisse Zeit, bis das erste Resultat (Ausgabe des letzten Konverters der Kette) vorliegt.

7.5.2. Signalisierungen und Ende

Die Signalisierungsmöglichkeiten sind ebenfalls in Abb. 7.6 dargestellt. Für das Anzeigen von Ereignissen der Konverter an die Kontrollapplikation wurde in dieser Implementierung eigens eine IPC-Kommunikationsbeziehung eingerichtet. Über DSI-Funktionen ist die Signalisierung von Ereignissen ebenfalls möglich. Maßnahmen der Kontrollapplikation zur Reaktion auf Blockierungsereignisse wurden noch nicht untersucht.

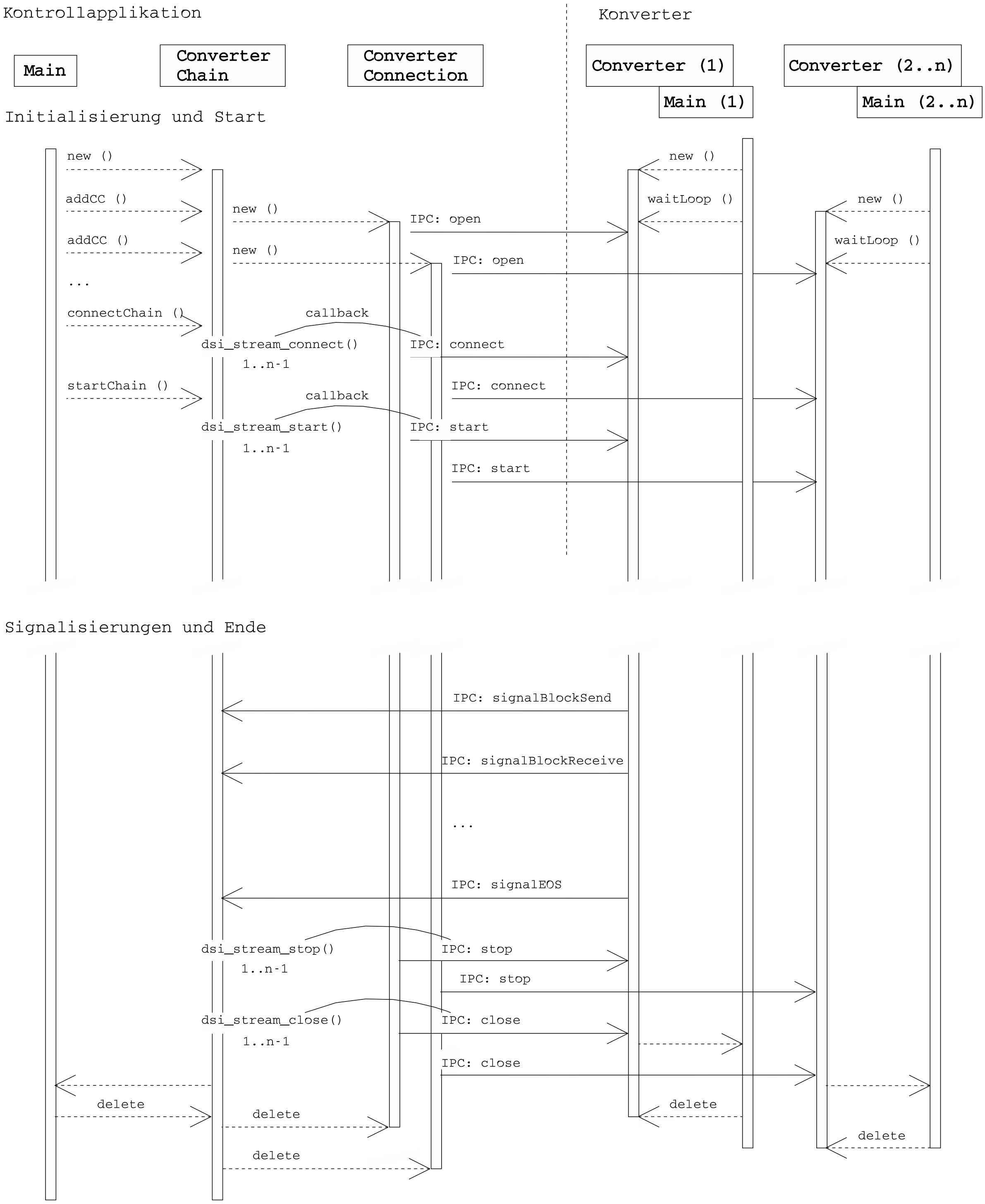

Abbildung 7.6.: Programmablaufplan

7.5.3. Daten senden und Daten empfangen

In Abb. 7.7 ist die Vorgehensweise beim Datenaustausch zwischen einem Produzenten und einem Konsumenten abgebildet.

Die Ausgangssituation dieses Ablaufdiagrammes ist ein voller DSI-Puffer. Erst zu dem Zeitpunkt, zu dem der Konsument den „Verbrauch“ eines Paketes signalisiert, ist die Paketanforderung des Produzenten erfolgreich. Während der Wartezeit legt sich der Produzent Semaphore-gesteuert schlafen. Nach jedem verbrauchten Paket prüft er dann erneut, ob genügend Speicherplatz zur Verfügung steht.

Der Produzent füllt nun das Paket mit Daten und teilt dem DSI mit, wann er damit fertig ist. Angenommen, der DSI-Puffer sei in der Zwischenzeit wieder vollständig geleert worden, wartet diesmal der Konsument auf ein Paket zum Verarbeiten. Auch diese Blockierung wird durch Semaphore gesteuert.

7.6. Entwickelte Anwendungen

In Anlehnung an das Modell aus Kapitel 7.3 entstanden verschiedene Anwendungen:

Zunächst wurde eine Kontrollapplikation entwickelt, die es dem Nutzer ermöglicht, zwischen verschiedenen vorkonfigurierten Konverterketten auszuwählen. Bei der Zusammenstellung der Konverterketten wurde lediglich auf funktionale Richtigkeit geachtet.

Eine Einplanung von Puffern und Vorlaufzeiten zwischen den Konvertern ist ebenfalls möglich. Dazu müssen vorhandene Konverter zunächst bezüglich ihrer Laufzeiteigenschaften untersucht werden und die sbS-Berechnung durchgeführt werden.

Das Scheduling der einzelnen Komponenten ist vorgesehen und kann realisiert werden, sobald die nötigen Schnittstellen des Betriebssystems bekannt und verfügbar sind.

Als Grundlage wurde eine Reihe von Konvertern zu Demonstrationszwecken entwickelt:

- CSI-Sender: sendet Datenpakete fester Größe
- CSI-RGB_Sender: generiert Frames im RGB24-Format und sendet diese
- CSI-Generic_Receiver: signalisiert den Empfang von beliebigen Paketen und deren Größe
- CSI-Adapter: liest eine Mediendatei und eine Indexdatei aus der Ramdisk, wendet die in der Indexdatei gespeicherte Zerlegungsbeschreibung auf die Mediendatei an
- CSI-RGB2YUV - Konverter: liest einzelne Videoframes im RGB24-Format und erstellt daraus Frames im YUV 4:4:4 - Format
- CSI-YUV_Presenter: liest Videoframes im YUV 4:4:4 - Format, konvertiert diese in das YUV 4:2:0 - Format und stellt diese Frames grafisch in der DROPS-Konsolenumgebung dar

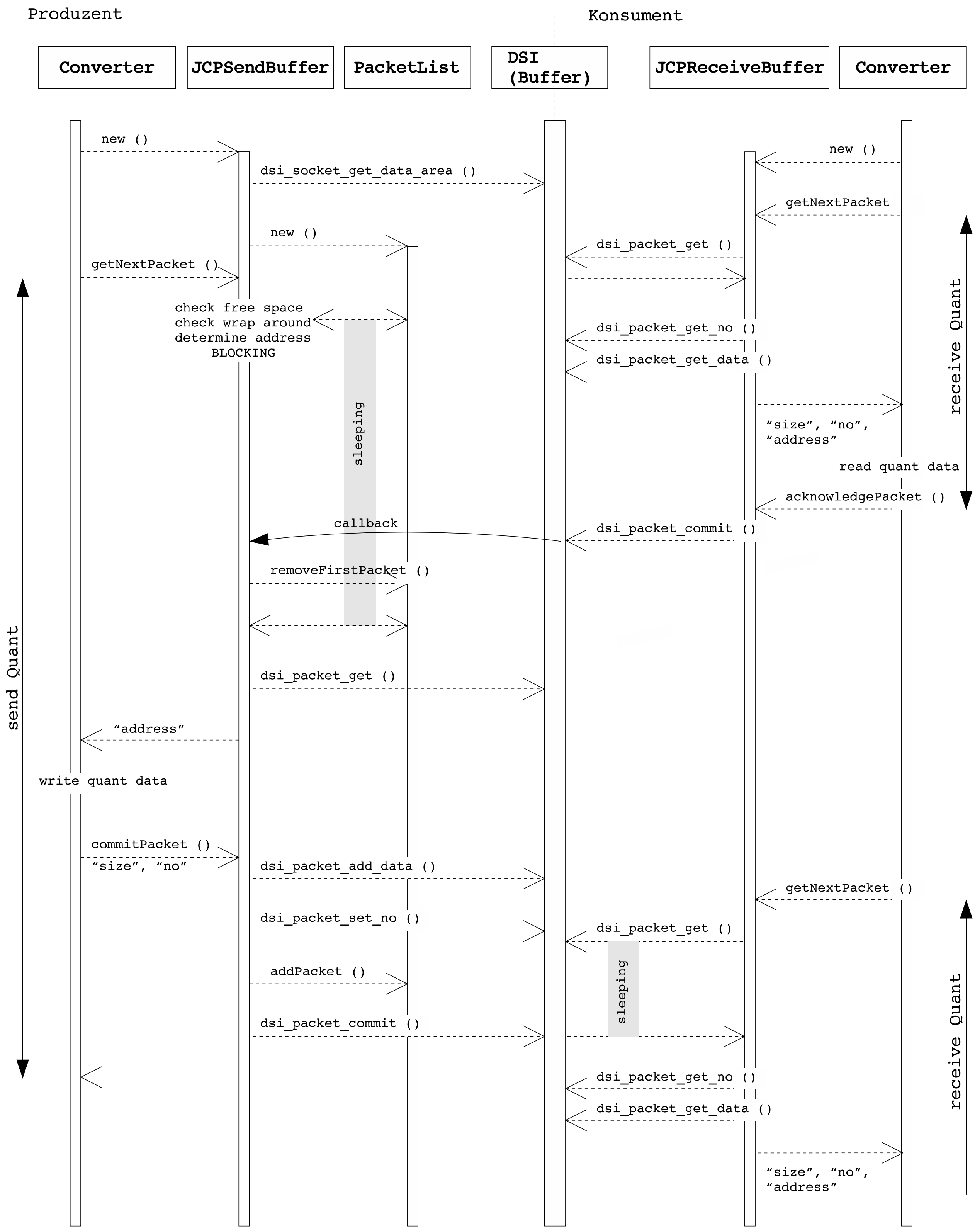

Abbildung 7.7.: Programmablauf beim Senden/Empfangen

Folgende Konverterketten sind vorkonfiguriert:

- CSI-RGB_Sender - CSI-RGB2YUV - CSI-YUV_Presenter
- DSI-StaticFS - CSI-Adaptor - CSI-Generic_Receiver
- DSI-StaticFS - CSI-Adaptor - CSI-RGB2YUV - CSI-YUV_Presenter

Die gewählte Konverterkette kann nun gestartet werden. Während der Arbeit der Konverter können verschiedene Ausgaben der einzelnen Komponenten beobachtet werden, die Aufschluss über Produktion und Verbrauch einzelner Pakete sowie über Blockierungen beim Senden und Empfangen von Paketen geben.

Weiterhin ist eine Quellcode-Dokumentation zu den entwickleten Anwendungen und Klassen verfügbar.

8. Evaluierung des entwickelten Systems und Ausblick

8.1. CSI

Ausgehend von der Aufgabenstellung dieser Diplomarbeit wurde eine abstrakte Multimediakonverterschnittstelle entworfen. Bezüglich der Datenübergabe kann man die hier entwickelten Komponenten als *Component Streaming Interface* (CSI) bezeichnen. Dieses CSI kapselt das zugrundeliegende DSI gegenüber dem Anwender.

Dabei wurden einerseits die vielfältigen Möglichkeiten des DSI durch das CSI in sinnvoller Art und Weise auf die Problemstellung eingegrenzt und andererseits das CSI gegenüber dem DSI um konverterkettenspezifische Funktionalität erweitert.

8.2. Erweiterung des CSI

In der bisher entwickelten Form erlaubt es das CSI lediglich, sehr einfache Konverter zu implementieren. Weiterentwicklungen sollten eine Grundlage für Konverter mit mehr als einem Ein- oder Ausgangsstrom bilden und damit das Modell der Multiplexer und De-Multiplexer unterstützen. Dafür muss ausgehend von den schwankungsbeschränkten Strömen auch noch eine theoretische Grundlage bzgl. der Pufferberechnung und des Schedulings geschaffen werden. Möglichkeiten der Synchronisation mehrerer parallel verarbeiteter Ströme werden Bestandteil der CSI-Erweiterungen sein.

Das objektorientierte Modell des Component Streaming Interface wird in diesem Zusammenhang ebenfalls erweitert bzw. neu strukturiert. Die Grundlage dafür wird die Konverterklassifizierung nach [SMW02] bilden.

8.3. Steuerung und Nutzerinteraktion

Bezüglich der Steuerung der einzelnen Konverter bzw. der Konverterkette wurden erste Möglichkeiten aufgezeigt. Die Verfahrensweise bei konzeptionell aufwendigeren Nutzerinteraktionen wie Pause, Vorspulen, Rückspulen, schnelle Wiedergabe usw. konnte im Rahmen dieser Diplomarbeit nicht ausreichend untersucht werden und muss Ziel weiterer wissenschaftlicher Arbeiten bleiben.

Pause / Resume

Zur Realisierung einer Pause/Resume - Funktion müssen alle Konverter inklusive des datenliefernden Dateisystems einzeln angehalten werden. Wenn die Pufferfüllstände erhalten bleiben, könnte eine

Wiederaufnahme der Arbeit ohne erneut einzuhaltende Vorlaufzeiten geschehen.

Problematisch ist dabei das Dateisystem: es bietet zur Zeit lediglich Funktionen zur Übertragung einer Datei vom Anfang bis zum Ende. Es sollte im Dateisystem entweder eine direkte Unterstützung einer Pause-Funktion geben (mit Erhaltung der Puffer) oder die Start-Position innerhalb einer Datei kann beim Lesen bestimmt werden. (Damit wäre die Semantik der Pause-Funktion bezogen auf das Dateisystem *Stop* und *Start ab Position x*)

Eine Pause-Funktion durch Blockierungen allein (z.B. hervorgerufen durch einen Arbeitsstop des letzten Konverters in einer Kette) ist nicht ohne weiteres möglich: Das Fortsetzen der Arbeit dürfte in diesem Fall nicht nur durch ein Lösen der Blockierung geschehen, es müssten auch neue Vorlaufzeiten der einzelnen Konverter realisiert werden, da sämtliche Puffer durch das Blockieren vollständig gefüllt sind.

Vorspulen, Rückspulen

Das vom Videorecorder bekannte „Spulen“ gibt es in Abhängigkeit des verwendeten Speichermediums nur noch selten: Oft werden Massenspeicher mit wahlfreien Zugriff verwendet. Das Problem wird also auf ein „Neupositionieren “ innerhalb eines Datenstromes reduziert. In einer Konverterkette ist vor allem das Dateisystem bzw. der Datenlieferant für diese Neupositionierung zuständig. Wie schon beim Pause/Resume - Problem sollte auch hier das Dateisystem entsprechende Unterstützung bieten. Zu beachten ist allerdings, dass eine Operation wie „Neupositionieren im Dateisystem“ nicht in Echtzeit ausgeführt werden kann und dadurch ein kontinuierlicher Datenfluss durch die Konverterkette nicht gewährleistet wird.

schnelle Wiedergabe

Für die Umsetzung einer sogenannten „Zeitraffer“ - Funktion ist es keinesfalls notwendig, sämtliche Daten mit erhöhter Geschwindigkeit durch die gesamte Konverterkette zu übertragen. Ist der Endpunkt der Kette z.B. eine Anzeigefunktion für ein Video, wird dort auch bei schneller Wiedergabe die gleiche Framerate wie bei Wiedergabe mit normaler Geschwindigkeit benötigt. Es wird also mit einer erhöhten Framerate aus dem Dateisystem gelesen, jedoch werden einzelne Frames bei der Darstellung ausgelassen.

Mit einer derartigen Aufgabe wäre das Dateisystem höchstwahrscheinlich überfordert. Allerdings könnte der erste Konverter der Kette derartige Entscheidungen treffen und somit die Datenrate und die damit verbundenen Bearbeitungszeiten für alle folgenden Konverter reduzieren.

langsame Wiedergabe

Anders stellt sich die Problematik bei langsamer Wiedergabe („Zeitlupe“) dar. Die Framerate, mit der beispielsweise das Video im Dateisystem gespeichert ist, orientiert sich an der normalen Wiedergabegeschwindigkeit. Bei der langsamen Wiedergabe werden nun mehr Frames benötigt, als im Dateisystem vorhanden sind. Wird der gesamte Datenstrom mit einer niedrigeren Daten- und Frameübertragungsrate eingeplant, wirkt die Wiedergabe „ruckelig“(Wie von klassischem Videorekorder bzw. DVD-Spieler bekannt ist). Eine Möglichkeit wäre hier die Interpolation der fehlenden Frames. Dies könnte durch einen einzigen Konverter vorgenommen werden. Befindet sich dieser noch dazu am Ende der Konverterkette, werden Datenvolumen und Verarbeitungszeiten durch die Interpolation nicht unnötig erhöht.

8.4. Einplanung mit dem Modell der schwankungsbeschränkten Ströme

Wenn die Konverterkette nach dem sbS-Modell geplant wird, beinhaltet dies sowohl die Einplanung der Puffergrößen als auch die Einplanung des zeitlichen Verhaltens (Startzeitpunkt, Scheduling) der einzelnen Komponenten. Ersteres wurde in der Beispielanwendung bereits umgesetzt. Das Scheduling wurde vorerst vernachlässigt, da noch keine fundierten Aussagen zum DROPS-Scheduling für Nutzerprogramme vorlagen und die Beispielapplikation hauptsächlich in einer VMware-Umgebung getestet wurde. In der VMware-Umgebung ist die Echtzeitfähigkeit keinesfalls gewährleistet.

Aus diesen Gründen konnten die Ergebnisse einer sbS-Berechnung nicht dahingehend kontrolliert werden, dass bei korrekter Einplanung und Ressourcenzusicherung keine Pufferüber- oder Unterläufe auftreten.

Abgesehen davon müssen die Berechnungsvorschriften für Puffergrößen und Vorlaufzeiten von *einer* Produzenten - Konsumenten - Beziehung auf die Einplanung einer ganzen Konverterkette übertragen werden.

8.5. Formatbeschreibung

Eine noch laufende Diplomarbeit am Institut Datenbanken beschäftigt sich mit einer Formatbeschreibung für die durch Konverter und Konverterketten zu verarbeitenden Daten. Basierend auf den daraus gewonnenen Erkenntnissen muss das CSI angepasst werden. Es müssen Quantengrößen spezifiziert werden und es sollte eine Formatprüfung in den einzelnen Konvertern durchgeführt werden. Zur Zeit obliegt es der Kontrollapplikation, die Konverter bezüglich deren Ein- und Ausgabeformat „richtig “ zu verbinden.

Je nachdem, wie das Konzept der Elementarkonverter umgesetzt wird, ist es zwar möglich, dass ein Konverter jeweils nur ein Eingabe- und Ausgabeformat[1] verarbeiten kann. Nichtsdestotrotz unterstützt jedes Format eine Menge von Parametern, die über Framegröße, Farbtiefe, etc. entscheiden. Diese Meta-Informationen sollten entweder zusammen mit den rohen Mediendaten von Produzent zu Konsument mit übertragen werden[2] oder dem einzelnen Konverter jeweils durch die Kontrollapplikation mitgeteilt werden.

8.6. Parameterbestimmung

Die für das sbS-Modell notwendigen Parameter sind in Kapitel 5 nachzulesen. Zusätzlich sind für das CSI noch Parameter wie z.B. die maximale Quantengröße anzugeben. Von einem existierenden Multimedia-Datenstrom an diese Parameter zu gelangen ist keinesfalls einfach. In einigen Fällen kann der Datenstrom vorher analysiert werden. Dies ist mit hohem Aufwand verbunden.

Es ist ebenfalls möglich, allein aus der Formatbeschreibung die Parameter für die zu verarbeitenden Multimedia-Daten zu gewinnen. Schließlich sind die Grenzen für z.B. Framegröße, Framera-

[1] im Sinne von Dateiformaten, Kodierungen
[2] z.B. als Scather-Gather-Elemente

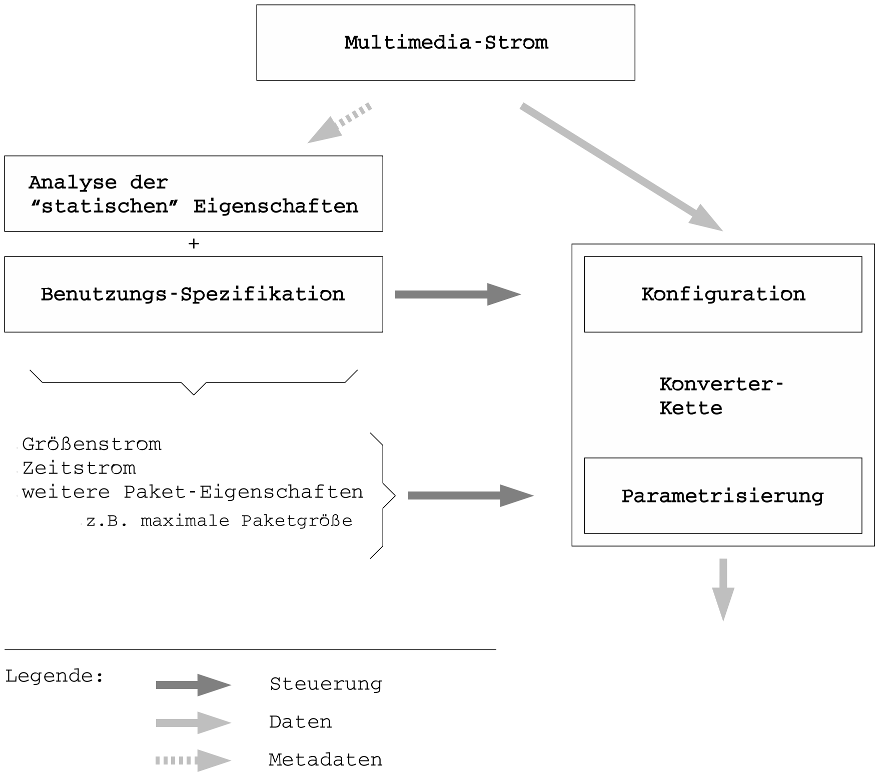

Abbildung 8.1.: Konfiguration, Benutzungs-Spezifikation, Parametrisierung

te oder Farbtiefe oft schon durch das Speicherformat bzw. die Kodierung festgelegt. Eine derartige Abschätzung wird vermutlich sehr ungenau und verschwendet deshalb wertvolle Ressourcen.

Die Lösung wird in einem Kompromiss aus beiden Ansätzen bestehen: Die Formatbeschreibung setzt die Grenzen fest und eine teilweise Analyse hilft, die Multimedia-Ströme noch genauer und besser zu charakterisieren.

In Abbildung 8.1 ist schematisch der Einfluss von Parametern auf den gesamten Konvertierungsprozess dargestellt.

Die „Konfiguration" umfasst dabei die funktional richtige Zusammenstellung der Konverterkette. Dies ist natürlich von der gewünschten Konvertierung[3] abhängig.

8.7. DROPS Umgebung

Viele Komponenten von DROPS befinden sich in ständiger Weiterentwicklung. Es wird dabei Funktionalität hinzugefügt und es werden Fehler beseitigt. Dieser Zustand hilft uns insofern, dass in diesem frühen Entwicklungsstadium Ideen und Vorschläge eingebracht und relativ problemlos eingearbeitet werden können.

In Anlehnung an die bereits angesprochenen Probleme bei der Nutzerinteraktion in Zusammenhang

[3]in der Abbildung als „Benutzungs-Spezifikation" bezeichnet

mit dem Dateisystem wären Erweiterungen des Paketes *StaticFS* (Ramdisk) denkbar. Außerdem sollte bald auch ein Festplatten-Dateisystem für DROPS-Anwendungen zur Verfügung stehen. Weiterhin müssen die Möglichkeiten des Scheduling für eigene Anwendungen spezifiziert werden, um das Modell der schwankungsbeschränkten Ströme konsequent umsetzen zu können.

A. beiliegende CDROM

Auf der beiliegenden CDROM befinden sich:

- Quellcode des CSI
- Dokumentation zum Quellcode in HTML
- dieses Dokument als PS und PDF
- ein VMware-Image zum Starten der Beispielkonverter

Installationshinweise existieren sich als Readme-Datei im jeweiligen Unterverzeichnis.

Abbildungsverzeichnis

Literaturverzeichnis

[avi97] *OpenDML AVI File Format Extensions*, September 1997.

[Buc94] Buck, J.: *Das MPEG-Verfahren.* Franzis-Verlag, april 1994.

[HBB+] Härtig, Hermann, Robert Baumgartl, Martin Borriss, Claude-Joachim Hamann, Michael Hohmuth, Frank Mehnert, Lars Reuther, Sebastian Schönberg und Jean Wolter: *DROPS OS Support for Distributed Multimedia Applications.*

[HMMW01] Hamann, Claude-Joachim, Andreas Märcz und Klaus Meyer-Wegener: *Buffer Optimization in Realtime Media Servers using Jitter-constrained Periodic Streams.* Technischer Bericht, TU-Dresden, January 2001.

[Hoh00] Hohmuth, Michael: *Smart_MPEG: MPEG decoder library specification and manual*, December 2000.

[Kei96] Keith, Jack: *Video Demystified: A Handbook for the digital engineer.* Hightext Interactive Inc., 1996.

[LHR] Löser, Jork, Herrmann Härtig und Lars Reuther: *A Streaming Interface for Real-Time Interprocess Communication.*

[LR01] Löser, Jork und Lars Reuther: *DSI: DROPS Streaming Interface*, january 2001.

[Mar] Marder, Ulrich: *Transformation Independence for Multimedia Systems.*

[MMI02] MMIS Workshop: *Konverter - und ihre Arbeit - Klassifikation von Konvertern*, Oktober 2002.

[MW] Meyer-Wegener, Prof. Dr. Klaus: *Daten- und Wissensbanken.* Script zur Vorlesung.

[SDA02] SDA Workshop: *bandwidth-based converter description for realtime scheduling at application level in media servers*, April 2002.

[SMW02] Suchomski, Maciej und Klaus Meyer-Wegener: *General Classification of AV Stream Conversions*, September 2002.

[SN95] Steinmetz, Ralf und Klara Nahrstedt: *Multimedia: Computing, Communications & Applications.* Prentice-Hall Inc., 1995.

[Ste99] Steinmetz, Ralf: *Multimediatechnologie.* Springer-Verlag, 1999.

[Tan95] TANENBAUM, ANDREW S.: *Moderne Betriebssysteme.* 1995.

[Wei] WEISS, ALEXANDER: *Multimedia-Datenformate.*