Großer Beleg

zum Thema

Entwurf und Implementierung von Sound- und Synchronisationkomponenten für das Smart-MPEG-Projekt

Jörg Nothnagel

Technische Universität Dresden
Fakultät Informatik

31. Mai 2001

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Eines der vielen Gebiete, auf denen Computertechnik eingesetzt wird, ist das der Echtzeitsysteme. Echtzeitsysteme zeichnen sich dadurch aus, daß sie Zeitschranken einhalten müssen, die im Vorfeld festgelegt wurden. Damit eine solche Anwendung (z.B. ABS-System im Kfz) diese Zeitkriterien erfüllen kann, muß das Betriebssystem entsprechende Mechanismen zur Verfügung stellen und seinerseits die Einhaltung von Zeitschranken gewährleisten.

Seit einigen Jahren wird am Lehrstuhl Betriebssysteme der TU Dresden an einem Mikrokernbetriebssystem gearbeitet, bei dem die Frage der Unterstützung von Echtzeitanwendungen im Vordergrund steht. Unter dem Namen „Smart-MPEG-Projekt“ wurde im Jahre 1999 mit der Arbeit an einer Bibliothek und einer Anwendung für dieses Betriebssystem begonnen, deren Ziel es ist, MPEG-1- und MPEG-2-Ströme zu dekodieren und unter Einhaltung von Echtzeitkriterien synchron wiederzugeben.

Aufgabe dieses Beleges ist es, eine Audio- und eine Synchronisationskomponente zu entwickeln und in die Bibliothek des Smart-MPEG-Projektes einzugliedern. Dabei soll zunächst eine Version für das Betriebssystem Linux entstehen, welche dann relativ einfach auf die eigentliche Zielplattform übertragen werden kann. Außerdem sollen entsprechende Anwendungen entwickelt werden, die die neue Funktionalität der Bibliothek nutzen.

Kapitel 2

Stand der Technik

2.1 Der MPEG-Standard

MPEG[1] ist der Sammelbegriff für eine Reihe von Standards zur komprimierten, digitalen Kodierung zusammengehöriger Audio- und Videodaten. Zur Zeit existieren 3 MPEG-Standards:
MPEG-1 (ISO/IEC 11172 (1992)), MPEG-2 (ISO/IEC 13818 (1995)) und MPEG-4 (ISO/IEC 14496 (1998)).

Zwei weitere Standards — MPEG-7 (Fertigstellung vorauss. 2001) und MPEG-21— werden zur Zeit entwickelt.

MPEG-1 und -2 konzentrieren sich auf den „typischen“ Fall der digitalen Videokodierung, -komprimierung, -übertragung und -dekomprimierung, sowie der Möglichkeit der synchronen Wiedergabe (digitales TV, DVD).

MPEG-4 erweitert das Feld der möglichen Inhalte des Datenstroms und definiert eine hierarchische Struktur, so daß komplexe Szenarien beschrieben werden können.

2.1.1 MPEG-1 und MPEG-2

Die Grundlage für diese Arbeit bilden die Standards MPEG-1 und MPEG-2. Sie sind jeweils in 3 Teile untergliedert, welche die Kodierung und Dekodierung von Video- bzw. Audioströmen, sowie die gemeinsame Übertragung und Speicherung so kodierter Ströme in einem Systemstrom definieren.

Beim Kodieren (siehe Abb. 2.2 Seite 5) werden aus den jeweiligen Rohdaten durch Komprimierung und hinzufügen von Metainformationen zunächst

[1] **M**otion **P**icture **E**xperts **G**roup

Audio- bzw. Videoströme erzeugt, die aus Audio- bzw. Video-Frames bestehen.

Ein Video-Frame enthält die kodierten oder dekodierten Daten, die zu einem Bild gehören. Ein Audio-Frame enthält entsprechende Daten einer Anzahl von Soundsamples. Die konkrete Anzahl ist für die verschiedenen im Standard definierten Schichten („layers“) unterschiedlich.

So entstandene Ströme bezeichnet man allgemein als Elementarströme. Elementarströme werden in PES[2]-Pakete variabler Länge verpackt und mit einem Header versehen, welcher u.a. Zeitstempel und eine ID enthält, die eine eindeutige Zuordnung des Paketes zu seinem ursprünglichen Elementarstrom ermöglicht. PES-Pakete verschiedener solcher Ströme werden zu einem Systemstrom gemultiplext, welcher wiederum als eine Folge von Systemstrompaketen repräsentiert wird. Jedem Systemstrompaket geht ein Header voran, der u.a. Informationen über die Zusammengehörigkeit der im Strom enthaltenen PES-Pakete und Werte einer Referenzuhr enthalten kann (siehe Abb. 2.1 Seite 5).

Zusammengehörige Elementarströme werden als Programm bezeichnet und sollen synchronisiert wiedergegeben werden. Ein Unterschied von MPEG-1 gegenüber MPEG-2 besteht darin, daß ein MPEG-2 Strom mehrere Programme enthalten kann.

Im Strom enthaltene Zeitinformationen dienen als Basis für die Synchronisation.

Nicht jedes Paket muß Zeitinformationen enthalten. Laut Standard dürfen aufeinanderfolgende Werte für Präsentationszeiten je Elementarstrom bzw. Werte der Referenzuhr um höchstens 700 ms differieren.

2.1.2 Reihenfolge von MPEG-Video - und MPEG-Audio-Frames im Elementarstrom

Frames werden in der Reihenfolge im Elementarstrom angeordnet, in der sie beim Empfänger dekodiert werden sollen. Diese Reihenfolge stimmt bei Audiodaten mit der Präsentationsreihenfolge[3] überein, weicht bei Videoströmen in der Regel aber von dieser ab. Grund dafür ist das verwendete Kodierungsverfahren der Rohdaten, welches einen Kompromiß zwischen wahlfreiem Zugriff auf einzelne Bilder und einer möglichst hohen Datenkompression erreichen möchte. Zu diesem Zweck werden Bild*beziehungen* hergestellt, d.h. Objekte, die in mehreren Bildern vorhanden sind, werden

[2] **P**acketized **E**lementary **S**tream

[3] Reihenfolge, in der Daten wiedergegeben werden sollen

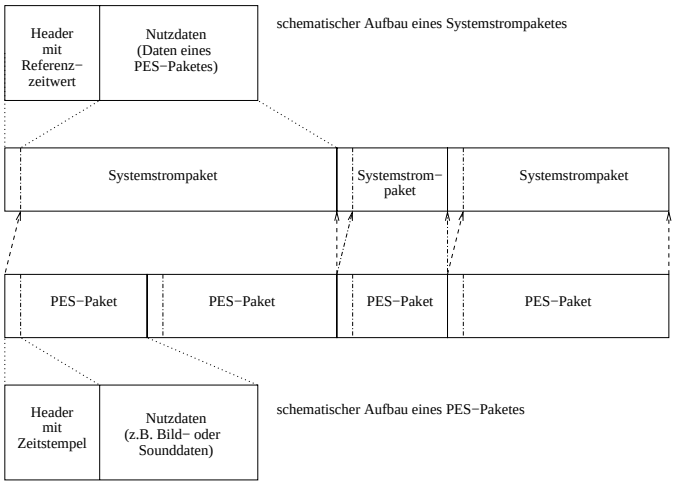

Abbildung 2.1: Schematische Darstellung eines MPEG_Stroms
Audio- und Video- PES-Paketen geht ein Header mit Präsentationszeitstempel voran; Systemstrompakete bestehen aus PES-Paketen und einem Header, der u.a. den Zeitwert der Referenzuhr enthält.

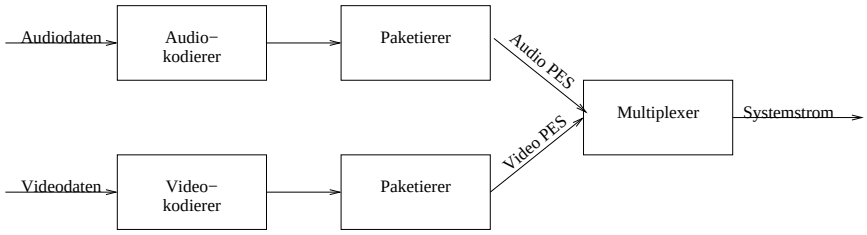

Abbildung 2.2: Rohdaten werden zunächst komprimiert, dann paketiert und schließlich gemultiplext.

einmal kodiert und dann nur noch referenziert. Der MPEG-Standard definiert 3 Haupttypen von Bildern.

- I-kodiertes Bild: (intra-coded) enthält nur Informationen über sich selbst; das Kodierungsverfahren ist dem JPEG-Verfahren[4] ähnlich; dient als Einstiegspunkt in den Strom; Kompressionsrate ca. 1:30 - 1:50
- P-kodiertes Bild: (predictive-coded) bezieht sich auf vorhergehende Bilder; kann nicht ohne Bezugsbilder dekodiert werden; gute Kompressionsrate (ca. 1.7-mal so hoch wie im I-Bild).
- B-kodiertes Bild: (bidirectional predictive-coded) bezieht sich sowohl auf vorhergehende als auch *nachfolgende* Bilder; kann nicht ohne Bezugsbilder dekodiert werden; beste Kompressionsrate (ca. 8-mal so hoch wie im I-Bild).

Alle Bilder, auf die sich ein B-Bild bezieht, werden vor diesem übertragen, um Wartezeiten beim Dekodieren zu vermeiden. Damit der Video-Dekoder die richtige Präsentationsreihenfolge wiederherstellen kann, werden die Bilder entsprechend durchnummeriert (siehe Abb. 2.3 Seite 6).

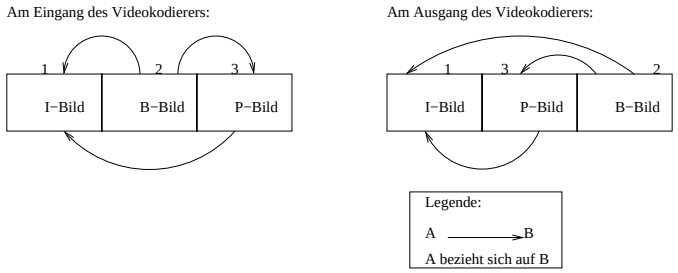

Abbildung 2.3: Video-Frames werden nach dem Kodieren umsortiert und so in die Dekodierreihenfolge gebracht.

2.2 Drops und Linux

Der Lehrstuhl Betriebssysteme der TU Dresden arbeitet seit einigen Jahren an DROPS[5], einem Mikrokernbetriebssystem auf Basis einer L4-Implemen-

[4] **J**oint **P**ictures **E**xperts **G**roup
[5] **D**resden **R**eal-Time **Op**erating **S**ystem

tierung [1]. Dabei steht die Frage der Unterstützung von Anwendungen mit Echtzeitforderungen im Vordergrund.

Die für diese Arbeit interessanten Unterschiede zu Linux liegen in der abweichenden Schnittstelle zu den Treibern für die Soundausgabe; eine einheitliche `/dev/`-Schnittstelle existiert nicht.

Desweiteren sind noch nicht alle Komponenten der C-Bibliothek für DROPS implementiert. Die entsprechende Funktionalität stellt das OSKit [2] bereit. Die Unterschiede zwischen Linux und DROPS-Version des Smart-MPEG-Projektes liegen hier begründet.

2.3 Das Smart-MPEG-Projekt

2.3.1 Motivation

Das Smart-MPEG-Projekt ist eine Anwendung für sowohl DROPS als auch Linux und soll nach seiner Fertigstellung die synchrone Wiedergabe von Video und Audio eines MPEG-1 oder MPEG-2 Stroms unter Einhaltung von Echtzeitanforderungen ermöglichen. Diese Anforderungen sind folgende:

- Wiedergabe der Bildfolge mit nachlassender Qualität bei steigender Systemlast (Verwerfen einzelner Bilder)
- Wiedergabe des Sound mit gleichbleibender Qualität

2.3.2 Grundlegende Konzepte

Hauptbestandteil des Smart-MPEG-Projekts ist eine Bibliothek, die dem Klienten die notwendigen Funktionen zum Initialisieren, Starten und Beenden der Videowiedergabe bereitstellt. Die Bibliothek ist in C++ implementiert und besteht aus weitgehend voneinander unabhängigen Komponenten. Die Abstraktion `Mpeg_buffer` schafft eine einheitliche Schnittstelle für die Blibliothekskomponenten zum Zugriff auf die Eingangsdaten. Zur Vermeidung von Kopieraktionen werden nur Zeiger auf eine Folge dieser Puffer (`Mpeg_buffer_sequence`) zwischen den Komponenten übergeben. Nicht mehr referenzierte Puffer und deren Speicherplatz werden automatisch freigegeben (siehe Abb. 2.4 Seite 9).

2.3.3 Stand der Entwicklung zu Beginn dieser Arbeit

Zu Beginn dieser Arbeit war die Entwicklung der Komponenten zur Videodekodierung und -darstellung größtenteils abgeschlossen. Die Systemstromdemultiplexerkomponente funktionierte mit Einschränkungen. Anwendungen

zur Darstellung von Video unter Linux (X11) sowie DROPS (VGA, VESA) waren vorhanden. (siehe Abb. 2.5 Seite 9).

Aufbau und Wirkungsweise der vorhandenen Komponenten

Aufgabe der Systemstromdemultiplexerkomponente ist es, den MPEG-Systemstrom in seine Elementarströme zu zerlegen, Metainformationen zu extrahieren und diese Daten anderen Komponenten zur Verfügung zu stellen. Sie ist modular aufgebaut und besteht aus den Teilen:

- Systemstromdemultiplexer: hat als Eingabe einen MPEG-Systemstrom, entfernt den Header und leitet die demultiplexten Nutzdaten (PES-Pakete) an Instanzen der 2. Komponente weiter
- PES-Handler Komponente: hat als Eingabe einen Strom von PES-Paketen, entfernt deren Header und stellt als Ausgabe einen Elementarstrom zur Verfügung, welcher verschiedenen Dekodern (z.B Videodekoder) als Eingabe dient.

Die Videodekoderkomponente überführt die Daten eines Video-Elementarstroms in mehreren Schritten in eine im YUV-Format[6] vorliegende Bildfolge. Sie ist ebenfalls modular aufgebaut und hat folgende Hauptbestandteile:

- Videodemultiplexer: hat als Eingabe einen Video-Elementarstrom, extrahiert die gespeicherte Folge von Einzelbildern und stellt diese Folge an seinem Ausgang zur Verfügung.
- `Mpeg_picture_reader`: Implementierungen dieser Schnittstelle benutzen die Ausgabe des Videodemultiplexers und bringen dessen Bildfolge in die vom nachgeschalteten Videodekoder gewünschte Reihenfolge. Insbesondere werden Bilder ignoriert, die infolge eines Fehlers im Eingabestrom nicht dekodiert werden können.
- Videodekoder: dekodiert die von `Mpeg_picture_reader` gelieferte Bildfolge. Seine Ausgabe kann von einer Anwendungskomponente weiter verwendet werden.

Die Anwendungskomponente ist nicht Bestandteil der Smart-MPEG-Bibliothek, sondern nutzt deren Funktionalität zur Ausgabe der Videodaten. Für Linux und DROPS existieren jeweils eigene Implementierungen.

[6] Farbraummodell mit Werten für Luminanz (**Y**) und Chrominanz (**U**, **V**)

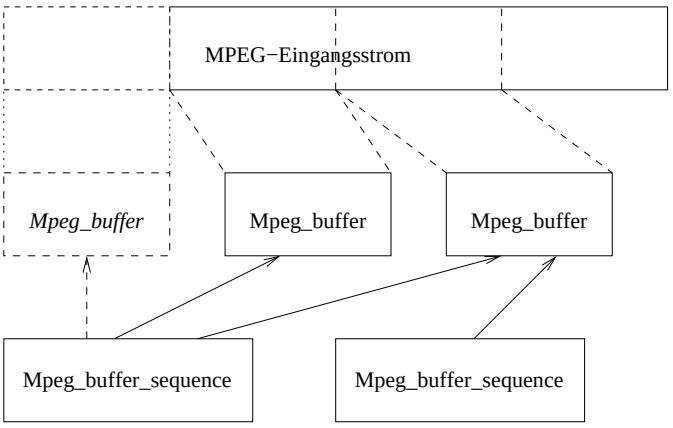

Abbildung 2.4: Zusammenhang `Mpeg_buffer` – `Mpeg_buffer_sequence`
Speicherplatz wird automatisch freigegeben, wenn keine Referenzen mehr existieren (linker `Mpeg_buffer`).

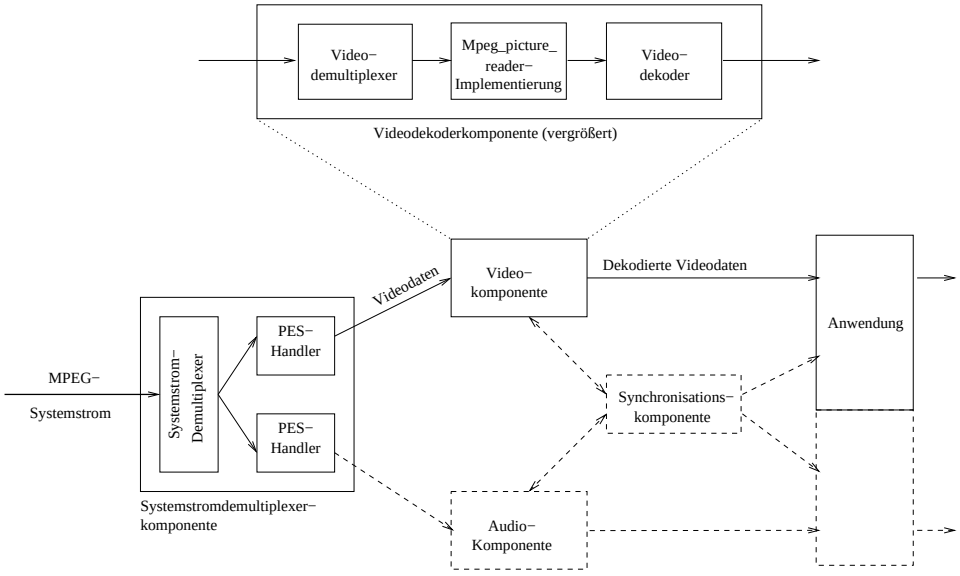

Abbildung 2.5: Komponenten des Smart-MPEG-Projektes.
Im Rahmen dieses Belegs entwickelte Komponenten sind gestrichelt dargestellt.

Kapitel 3

Entwurf

3.1 Ansätze

3.1.1 Die Soundkomponente

Dem Gedanken des Smart-MPEG-Projektes entsprechend, sollte die zu entwickelnde Soundkomponente modular aufgebaut sein, eine wohl definierte, kompakte Schnittstelle zu den anderen Komponenten besitzen, eine `Mpeg_buffer_sequence` als Eingabe verwenden und möglichst alle im MPEG-Standard definierten Soundformate unterstützen. Hieraus ergaben sich 2 grundlegende Ansätze. Soll die Komponente neu entwickelt und implementiert werden oder besteht die Möglichkeit, eine vorhandene Bibliothek in das bestehende Smart-MPEG-Projekt einzubinden?

Eine Eigenimplementierung hat den Vorteil, die o.g. Anforderung optimal umsetzen zu können. Der damit verbundene Arbeitsaufwand ist jedoch verhältnismäßig hoch. Das Einbinden einer vorhandenen Bibliothek reduziert diesen Aufwand, zwingt aber zu Kompromissen im Hinblick auf die Anforderungen. Da es einige frei verfügbare MPEG-Soundbibliotheken gibt, wurde der Gedanke einer Neuimplementierung aufgegeben.

Bei der Suche nach einer geeigneten Bibliothek wurden die Programme `amp` [3], `mpg123` [4] und die der `mp3blaster`-Software [5] zugrundeliegende `mpegsound`-Bibliothek [6] untersucht. Bei den Tests schnitt `mpg123` am besten ab, gefolgt von `mpegsound` und `amp` (siehe Abb. 5.1 Seite 31). Die Wahl fiel schließlich auf die `mpegsound` Bibliothek. Diese Bibliothek unterstützt alle im MPEG-Standard definierten Audioformate mit Ausnahme von Strömen mit variabler Bitrate, besitzt eine übersichtliche Struktur und gute Leistungsfähigkeit. Ein weiteres Kriterium für diese Entscheidung war die Tatsache, daß `mpegsound` genau wie das Smart-MPEG-Projekt unter

objektorientierten Gesichtspunkten entworfen und in der Programmiersprache C++ implementiert wurde, wodurch die Eingliederung zusätzlich vereinfacht und ein guter Kompromiß zwischen Anforderungen und Aufwand erzielt wurde.

3.1.2 Die Synchronisationskomponente

Auch diese Komponente sollte den Anforderungen des Smart-MPEG-Projektes genügen. Wegen des vergleichsweise geringen Aufwands wurde eine Eigenimplementierung angestrebt. Das Hauptaugenmerk der Entwurfsentscheidung richtete sich hier darauf, einen guten Kompromiß zwischen allgemeingültiger Verwendbarkeit und den Besonderheiten der beteiligten Komponenten zu finden, sowie darauf, die durch die Integration notwendigen Änderungen an den bereits fertiggestellten Teilen des Smart-MPEG-Projektes möglichst gering zu halten.

3.1.3 Die Anwendung

Eine Anwendungskomponente soll die Funktionalität der Smart-MPEG-Bibliothek zur Dekodierung beliebiger MPEG-Ströme und die Bereitstellung von Synchronisationsmechanismen in Anspruch nehmen und die dekodierten Audio- bzw. Videodaten auf entsprechende Art und Weise darstellen. Dazu sollen die Ausgabemechanismen der entsprechenden Betriebssystemumgebung genutzt werden. Weiterhin soll diese Komponente zum Testen der Smart-MPEG-Bibliothek verwendet werden können.

Wegen der großen Unterschiede der beiden Zielplattformen Linux und DROPS wurde für jedes System eine eigene Applikation entwickelt. Dazu wurden die schon vorhandenen Anwendungen für die Videokomponente (`mpeg_x11_viewer` für die Linuxvariante und `drops_vesaview` für DROPS) entsprechend erweitert (siehe 2.3.3 Seite 7).

3.2 Entwicklungsumgebung

Die Entwicklung erfolgte auf einem Linux-System mit einem Linux-Compiler und einem Cross-Compiler für DROPS. Zunächst wurde für die Linux-Plattform entwickelt und getestet, danach für die eigentliche Zielplattform crosscompiliert. Dabei waren Anpassungen der Teile der Ausgabe, die Linux-typische Treiberzugriffe benutzen, notwendig, und es mußten vorhandene DROPS-Bibliotheken anstelle der entsprechenden Linux-Bibliotheken eingebunden werden (z.B. semaphore, thread, ...).

3.3 Anpassung der Soundbibliothek

3.3.1 Aufbau der `mpegsound`-Bibliothek

Die gewählte Bibliothek besteht aus folgenden Komponenten:

- `Soundinputstream`: Eingabeschnittstelle; öffnet / liest / schließt Eingabestrom
- `Soundinputstreamfromfile`: Kindklasse von `Soundinputstream`; liest Datei
- `Soundinputstreamfromhttp`: Kindklasse von `Soundinputstream`; liest http
- `Soundplayer`: Ausgabeschnittstelle; gibt dekodierte Daten aus
- `Rawplayer`: schreibt Daten nach `/dev/dsp` (Soundtreiber)
- `Rawtofile`: schreibt Daten in eine Datei
- `Mpegbitwindow`: Hilfsklasse zum Dekodieren von mp3-Strömen
- `Mpegtoraw`: Dekoder für MPEG-Audioströme; liest Daten aus einem `Soundinputstream` und schreibt nach dem Dekodieren in einen `Soundplayer`
- `Wavetoraw`: Dekoder für Wave-Audioströme; liest Daten aus einem `Soundinputstream` und schreibt nach dem Dekodieren in einen `Soundplayer`
- `Fileplayer`: Superklasse für `Wavefileplayer` und `Mpegfileplayer`
- `Wavefileplayer`: Initialisiert die Bibliothek; Instanziiert `Soundplayer`, `Soundinputstreamfrom{http, file}` und `Wavetoraw`; Schnittstelle zum Nutzer
- `Mpegfileplayer`: Initialisiert die Bibliothek; Instanziiert `Soundplayer`, `Soundinputstreamfrom{http, file}` und `Mpegtoraw`; Schnittstelle zum Nutzer

Benutzer der Bibliothek rufen die Methode `openfile(char* in, char* out)` von `Mpegfileplayer` oder `Wavefileplayer` auf, je nachdem, ob ein MPEG- oder Wave-Strom abgespielt werden soll; `in` bezeichnet die Quelldatei, `out` die Ausgabedatei. Unter Zuhilfenahme dieser Parameter werden

die richtigen Objekte von `Soundinputstream`, `Soundplayer` und `Mpegtoraw` bzw. `Wavetoraw` instanziiert.

Durch den Aufruf von `playing()` des `Mpegfileplayer` oder `Wavefileplayer` wird der Dekodier- und Abspielvorgang in Gang gesetzt. Die `Mpegtoraw`- bzw. `Wavetoraw`-Instanz dekodiert die Daten, welche sie von einem `Soundinputstream`-Objekt liest und gibt sie dann sofort an die Ausgabekomponente (`Soundplayer`-Objekt) weiter.

Abbildung 3.1 auf Seite 14 stellt die Klassenabhängigkeiten grafisch dar.

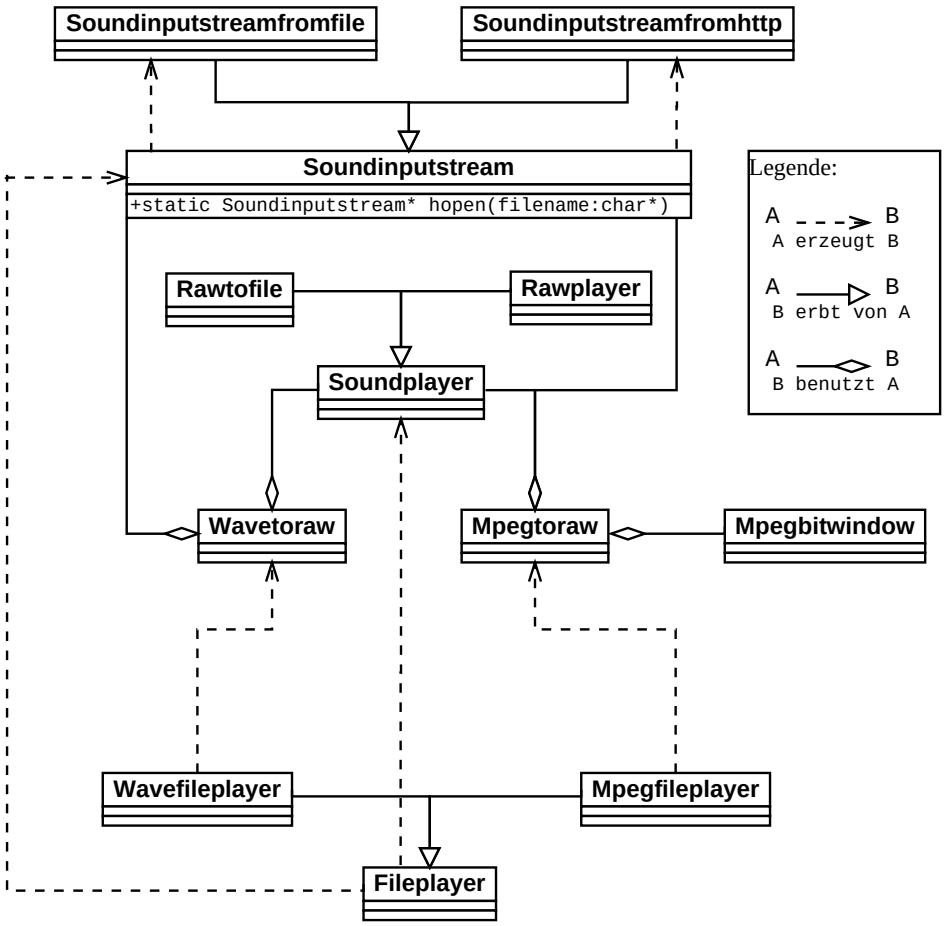

Abbildung 3.1: Die mpegsound-Bibliothek vor der Anpassung: Eingabestrom ist eine Datei bzw. ein http-Strom. Die Soundausgabe erfolgt automatisch nach dem Dekodieren.

3.3.2 Die Sound-Komponente des Smart-MPEG-Projektes

Die im letzten Abschnitt vorgestellte Sound-Bibliothek bildet die Grundlage für die Sound-Komponente des Smart-MPEG-Projektes. Folgende Erweiterungen und Anpassungen mußten vorgenommen werden:

Um die Soundausgabekomponenten der Bibliothek im Zusammenhang mit der beabsichtigten synchronen Ausgabe von Audio- und Videoströmen verwenden zu können, mußte die direkte Ausgabe der dekodierten Daten verhindert werden. Weiterhin mußte die Eingabeschnittstelle geändert werden, damit `Mpeg_demuxed_sequence`-Objekte akzeptiert werden.

Diese Anpassungen geschahen hauptsächlich dadurch, daß die entsprechenden Klassen überladen, bzw. neu implementiert wurden. Daten werden nicht mehr direkt ausgegeben, sondern in `Mpeg_audio_frame`-Instanzen zwischengespeichert (siehe Abb. 3.2 Seite 16).

Außerdem waren kleinere Änderungen in der Bibliothek selber nötig. Der Sichtbarkeitsbereich einiger Methoden und Variablen wurde von `private` nach `protected` geändert, um von abgeleiteten Klassen auf diese zugreifen zu können.

Es folgt nun eine Aufstellung der neu entwickelten Komponenten.

- `Mpeg_audio_input_wrapper`: Eingabeschnittstelle; liest Eingabestrom (`Mpeg_buffer_sequence`)
- `Mpeg_raw_to_sound`: schreibt Daten nach `/dev/dsp` (Soundtreiber); erweitert `Soundplayer`
- `Mpeg_raw_to_file`: schreibt Daten in eine Datei; erweitert `Soundplayer`
- `Mpeg_audio_frame`: repräsentiert einen dekodierten Audioframe; benutzt `Soundplayer` für seine Ausgabe
- `Mpeg_audio_to_raw`: Dekoder für MPEG-Audioströme; liest Daten aus einem `Mpeg_audio_input_wrapper` und schreibt nach dem Dekodieren in einen `Soundplayer`; erzeugt Instanzen von `Mpeg_audio_frame` zum Zwischenspeichern dekodierter Audiodaten; erweitert `Mpegtoraw`
- `Mpeg_audioplayer`: Initialisiert die Bibliothek; Instanziiert `Mpeg_raw_to_sound` bzw. `Mpeg_raw_to_file`, `Mpeg_audio_input_wrapper` und `Mpeg_audio_to_raw`
- `Mpeg_s_audioplayer`: Superklasse von `Mpeg_audioplayer`; erzeugt Instanz von `Mpeg_audioplayer`; Schnittstelle zum Nutzer

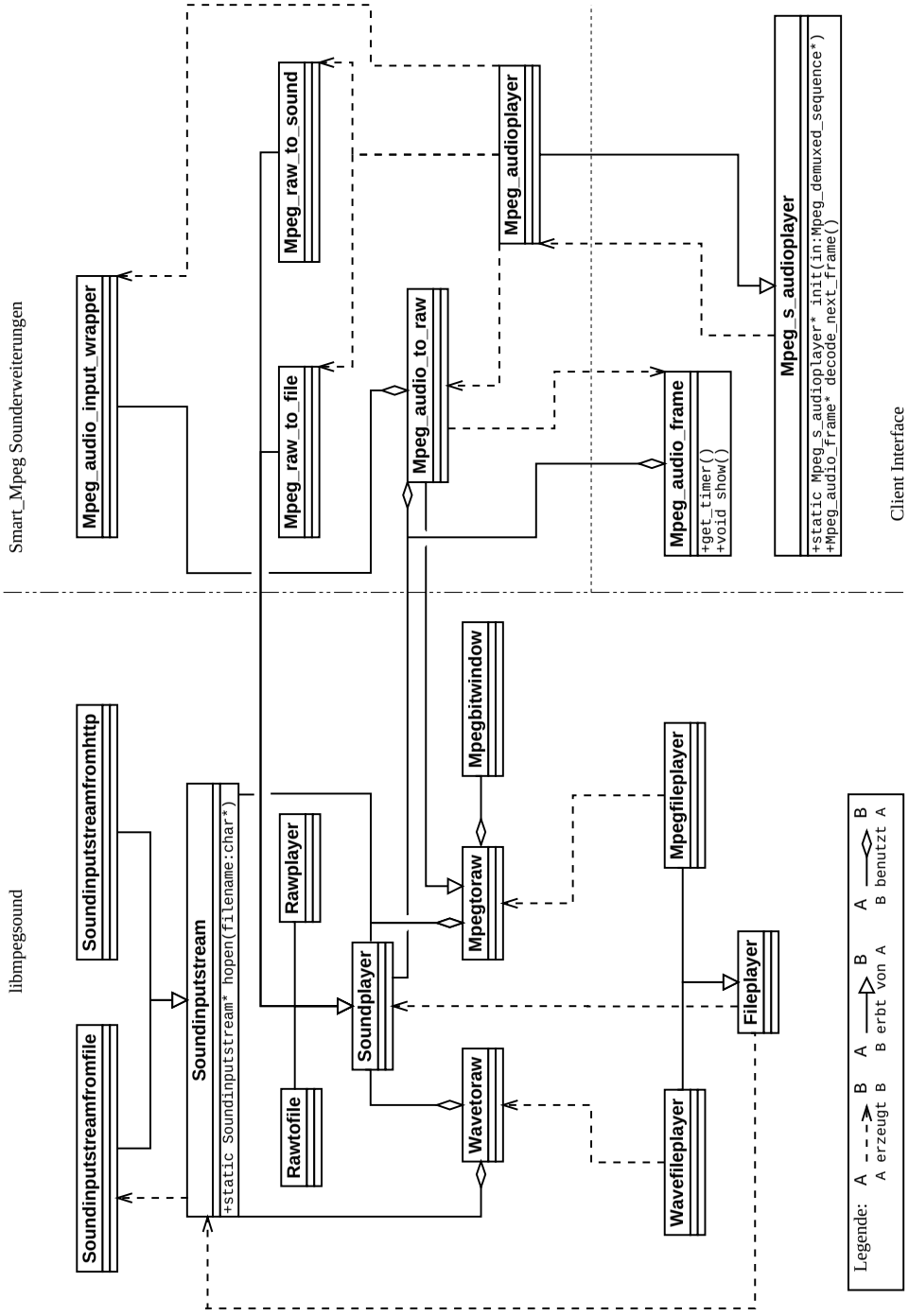

Abbildung 3.2: Die mpegsound-Bibliothek nach der Anpassung. Einige Klassen wurden überladen; Eingabestrom ist eine `Mpeg_demuxed_sequence`; die Soundausgabe erfolgt nun explizit durch Aufruf von `show()` in `Mpeg_audio_frame`

Anwender der Bibliothek rufen zu Beginn die `init()`-Methode des `Mpeg_s_audioplayer` auf und übergeben als Parameter Quell- und Zieldatenstrom sowie ein Objekt der Synchronisationskommponente. `Mpeg_s_audioplayer` erzeugt ein Objekt des Typs `Mpeg_audioplayer`, welches die Rolle des `Fileplayer` der Ausgangsbibliothek übernimmt (siehe 3.3.1). `Mpeg_audioplayer` instanziiert `Mpeg_audio_input_wrapper`, das entsprechende `Soundplayer`-Objekt und `Mpeg_audio_to_raw`.

Der Nutzer erhält die dekodierten Audiodaten in Form von `Mpeg_audio_frame`-Objekten durch Aufrufe der Methode `decode_next_frame()` des `Mpeg_s_audioplayer`.

Die Ausgabe erfolgt für die Linux-Version durch Aufruf von `show()` im `Mpeg_audio_frame`-Objekt unter Zuhilfenahme der integrierten Ausgabekomponenten. Für die DROPS-Variante existiert noch keine solche Komponente. Die Audiodaten können jedoch durch entsprechende Methodenaufrufe aus den `Mpeg_audio_frame`-Objekten ausgelesen und weiterverwendet werden.

3.4 Synchronisation im Smart-MPEG-Projekt

3.4.1 Änderungen an bestehenden Komponenten

Um Synchronisation zu ermöglichen, mußte zunächst die Systemstromdemultiplexerkomponente um eine Schnittstelle zum Abfragen der Präsentationszeit erweitert werden (siehe 2.1 Seite 3). Der hier zurückgelieferte Wert entspricht der Zeit in Millisekunden, die seit dem ersten Wert der Referenzuhr vergangen ist. Wenn seit dem letzten Aufruf dieser Methode kein neuer Präsentationszeitstempel vorliegt, wird 0 zurückgeliefert.

Um die Funktion der Smart-MPEG-Bibliothek auch im Betrieb mit mehreren Threads zu gewährleisten, wurden die kritischen Bereiche des Systemstromdemultiplexers durch eine Sperre gegen den gleichzeitigen Zugriff durch mehr als einen Thread geschützt (siehe Abb. 2.5 Seite 9).

Weiterhin war es erforderlich, Mechanismen zur Erzeugung und Weitergabe von `Mpeg_presentation_timer`-Instanzen in den Videodekoder zu integrieren.

3.4.2 Funktionsweise der Synchronisationskomponente

Video- bzw. Audiodekoder ordnen jedem Video- bzw. Audio-Frame eine Instanz von `Mpeg_presentation_timer` zu, welcher der aktuelle Präsentationszeitstempel und die aktuelle Frame-Rate übergeben wird.

`Mpeg_presentation_timer` bietet Funktionen an, mit deren Hilfe erfragt werden kann, ob der Präsentationszeitpunkt schon überschritten wurde, oder nicht.

Die Berechnung des Präsentationszeitpunktes t_p erfolgt nach der Gleichung:

$$t_p = t_{init} + (t_{stamp} * factor) + delay$$

mit
t_{init} = Zeitpunkt der Initialisierung der Synchronisationskomponente
t_{stamp} = Wert des aktuellen Präsentationszeitstempel

$factor$ und $delay$ sind Konstanten, die bei der Initialisierung der Synchronisationskomponente gesetzt werden und dazu dienen, die Darstellung um $factor$ zu dehnen bzw. um $delay$ Millisekunden zu verzögern.

Ist kein Zeitstempel vorhanden, erfolgt die Berechnung der Zielzeit unter Zuhilfenahme der Frame-Rate nach folgender Gleichung:

$$t_p = t_{p_l} + (factor * \frac{1000}{r_{frame}} * (timecode - timecode_l))$$

mit
t_{p_l} = Präsentationszeitpunkt des Vorgänger-Frames
$timecode_l$ = Zeitcode des Vorgänger-Frames
$timecode$ = Zeitcode des aktuellen Frames
r_{frame} = Die für den aktuellen Frame gültige Frame-Rate

Der Zeitcode ist ein monoton wachsender Zähler, der Video-Frames entsprechend ihrer Präsentationsreihenfolge kennzeichnet (siehe 2.1.2 Seite 4). Aufgrund der Übereinstimmung von Präsentations- und Dekodierreihenfolge im Falle von Audio-Frames vereinfacht sich dort die Berechnung zu:

$$t_p = t_{p_l} + (factor * \frac{1000}{r_{frame}})$$

Der aktuelle Status eines Frames in Bezug auf seine Präsentationszeit kann über folgende Funktionen ermittelt werden:

- `early()` liefert „wahr“, wenn der Präsentationszeitpunkt noch nicht erreicht wurde
- `too_late()` liefert „wahr“, wenn der Präsentationszeitpunkt bereits überschritten wurde

Weiterhin stehen folgende Funktionen zum Führen einer Statistik zu Verfügung.

- `skipped(), decoded(), played()` zum Führen einer Statistik
- `statistic()` zur Ausgabe der Statistik

Innerhalb eines Zeitfensters von 50 ms vor Erreichen des Präsentationszeitpunktes liefern sowohl `early()` als auch `too_late()` „falsch“.

Anwendungen werden typischerweise eine Schleife folgender Art benutzen, um den Präsentationszeitpunkt zu bestimmen.

```
while(pres_timer.early())
  usleep(1000);//wait
if(! pres_timer.too_late())
{
  //show
}
```

3.4.3 Besonderheiten des `Mpeg_video_pres_timer`

Anders als bei einem MPEG-Audiostrom, bei dem die Audio-Frames bereits in anzeigechronologischer Reihenfolge übertragen werden, weicht diese Reihenfolge bei Videoströmen in der Regel ab (siehe 2.1.2 Seite 4). Für Bilder ohne eigenen Zeitstempel (siehe 2.1.1 Seite 3) kann die tatsächliche Präsentationszeit erst errechnet werden, nachdem die Präsentationsreihenfolge vom Dekoder wiederhergestellt wurde und somit die aktuellsten Bezugszeiten verfügbar sind (siehe Abb. 3.3 Seite 20). Der `Mpeg_video_pres_timer` berücksichtigt diese Besonderheit und errechnet den Ausgabezeitpunkt in solchen Fällen beim ersten Aufruf von `early()` bzw. `too_late()` durch den Nutzer für das entsprechende Bild.

Videodekoder können die Funktion `drop_pre_decode()` aufrufen, um festzustellen, ob das zugehörige Bild bereits vor dem Dekodieren seine Präsentationszeit überschritten hat. Diese Funktion errechnet für Bilder ohne eigenen Zeitstempel auf Grundlage vorhandener Zeitstempel einen vorläufigen Darstellungszeitpunkt. Dieser kann vom tatsächlichen Zeitpunkt abweichen, wenn der Aufruf vor der Wiederherstellung der Darstellungsreihenfolge erfolgte. Für das in Abb. 3.3 auf Seite 20 dargestellte Beispiel würde `drop_pre_decode()` für Bild 3 „wahr“ zurückliefern, wenn die aktuelle Systemzeit z.B. 128 betragen würde und Bild 4 noch nicht gelesen worden wäre.

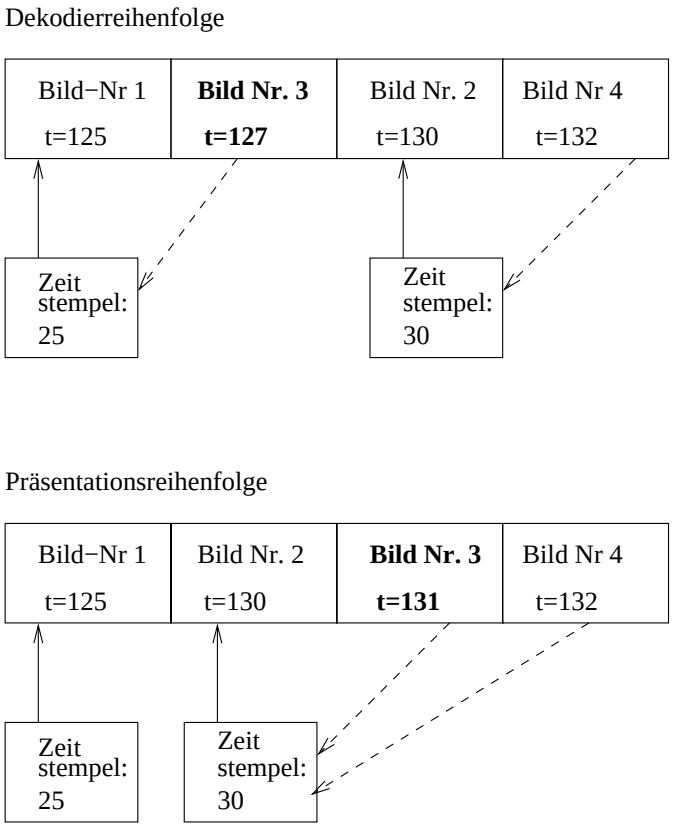

Abbildung 3.3: Besonderheit der Präsentationszeitberechnung von Bildern ohne Zeitstempel:
Die korrekte Berechnung kann erst nach Wiederherstellung der Präsentationsreihenfolge der Bilder erfolgen; ansonsten kann es zu Fehlberechnungen kommen (obere Darstellung).
$t_{init} = 100$; $r_{frame} = \frac{1}{s}$; $t = Präsentationszeitpunkt$

3.5 Entwicklung der Anwendungskomponente

Bei der Entwicklung der Anwendung wurde von der vorhandenen Linux-Videoanwendung (`mpeg_x11_viewer`) ausgegangen. Diese wurde entsprechend erweitert, wobei der Alghorithmus von Dekodierung, Zeitstempelvergleich und Ausgabe in mehreren Variationen umgesetzt wurde. Diese werden im Kapitel Implementierung ab Seite 23 ausführlich dargestellt.

Für die DROPS-Anwendung bildete `drops_vesaview` die Grundlage. Die Videoausgabe erfordert eine VESA-2 kompatible Grafikkarte.

Mangels eines allgemeingültigen Soundtreibers für die DROPS-Umgebung wurde das Paket ibm-sound, welches die Soundausgabe auf einem IBM-T20 Notebook ermöglicht, so abgeändert, daß dekodierte Sounddaten zwar empfangen, aber nicht mehr ausgegeben werden. Stattdessen wird eine Statistik der übertragenen Bytes erstellt.

`drops_vesaview` wurde mit der leistungsfähigsten Variante der Linux-Anwendung und der abgewandelten Version des ibm-sound Paketes kombiniert. Die entstandene Anwendung (`drops_vesa_player`) kann nun zu Testzwecken verwendet werden.

Sobald ein Soundtreiber für DROPS vorliegt, kann `drops_vesa_player` durch relativ kleine Änderungen angepaßt werden.

Kapitel 4

Implementierung

Die im Rahmen dieses Belegs entstandenen Anwendungen führen nach ihrer Initialisierung, in der die beteiligten Komponenten (Smart-MPEG-Bibliothek, Darstellungskomponenten ...) auf den Betrieb vorbereitet wurden, folgende Aktionsfolge aus:

```
while(frames_left())
{
  decode_next_audio/video_frame();
  while(timer.early()); //wait
  if(!timer.too_late)
   show_audio/video_frame
}
```

4.1 Die Linux-Anwendung

Alle nachfolgend aufgeführten Leistungsunterschiede wurden auf einem Testrechner mit Pentium III Prozessor, einer Taktfrequenz von 500 MHz und 128 MB Arbeitsspeicher beobachtet.

4.1.1 Varianten

In der ersten Version der Linux-Anwendung wurde o.g. Ablauf als *eine* Schleife implementiert (siehe Abb. 4.1 Seite 24).

Das Ergebnis war allerdings unbefriedigend. Auf dem Testrechner kam eine fließende Ausgabe von Bild oder Ton nicht zustande. Eine große Anzahl von sowohl Audio- als auch Video-Frames wurden verworfen, weil sie ihr Präsentationszeitfenster überschritten hatten. Ursache für dieses schlechte

Verhalten ist die streng sequentielle Abarbeitung der Aktionsfolge, wodurch sich Verzögerungen summieren, die durch Dekodierung und Ausgabe entstehen.

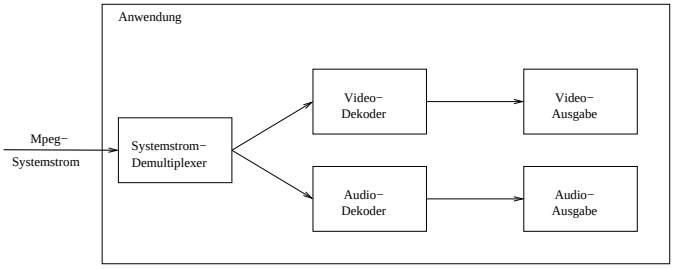

Abbildung 4.1: Variante 1: Ein Thread steuert den gesamten Ablauf.

Auf der Suche nach einer besseren Lösung entstanden im Verlauf der Arbeit 4 weitere Varianten, welche die Auswirkungen dieser Verzögerungen durch Parallelisierung der Abläufe eindämmen. Voraussetzung für solche Implementierungen ist die Multithreadfähigkeit der Smart-MPEG-Bibliothek, welche zu diesem Zweck dort eingebaut wurde (siehe 3.4.1 Seite 17).

Diese Varianten erzielen deutlich bessere Ergebnisse als die ursprünglichen Version. Auf dem Testrechner wurden Bild und Ton mit guter Qualität dargestellt. Es gibt dennoch Unterschiede im Detail, welche im folgenden aufgeführt werden.

Variante 2 In dieser Version läuft o.g. Aktionsfolge für Audio- und Videoanwendung in verschiedenen Threads ab, sodaß die Auswirkungen der Blockierungszeiten auf den jeweiligen Thread beschränkt bleiben (siehe Abb. 4.2 Seite 25).

Negativ wirkt sich hier das Fehlen von Puffern für dekodierte Bilder bzw. Sound aus. So kann es bei Verzögerungen im Eingabestrom (z.B. durch Plattenzugriffe zum Lesen des kodierten MPEG-Stroms) zu einer störenden Unterbrechung der Ausgabe kommen.

Variante 3 Diese Implementierung entkoppelt die Dekodierung von der Ausgabe. Sie ähnelt der Ausgangsvariante mit dem Unterschied, daß dekodierte Bilder bzw. Sound nicht direkt ausgegeben, sondern jeweils in einen Puffer geschrieben werden. Dieser Puffer wird von einer in einem anderen

Thread laufenden Komponente gelesen, welche den Test der Präsentationszeit sowie die Weiterleitung an die Ausgabekomponente durchführt (siehe Abb. 4.3 Seite 26).

Die Einführung des Puffers erlaubt eine unterbrechungsfreie Wiedergabe über Lesezugriffe hinweg und behebt so das in Variante 2 aufgetretene Problem. Allerdings hat sich die Soundqualität im Vergleich zu Variante 2 verschlechtert – es tritt ein periodisches „Knacken“ auf. Die Ursache dafür liegt wohl darin, daß Abhängigkeit zwischen Video- und Audiodekodierung wiederhergestellt wurde.

Variante 4 Diese Variante faßt die Idee von Variante 2 erneut auf und kombiniert sie mit dem Pufferkonzept der Variante 3. Das Resultat ist eine Entkopplung von Audio- und Videodekodierung auf der einen und eine Trennung von Dekodierung und Ausgabe auf der anderen Seite (siehe Abb. 4.4 Seite 26).

Tatsächlich wurden dadurch die Nachteile der Vorgänger beseitigt und eine gleichbleibend gute Sound- und Bildqualität erreicht, die unterbrechungsfrei über lesebedingte Verzögerungen hinweg fortgesetzt werden kann.

Variante 5 In dieser Implementierung wird ebenfalls das Konzept von Variante 4 umgesetzt, mit dem Unterschied, daß Video- und Audiodekodierung in getrennten *Prozessen* ablaufen und die Entkopplung damit total ist (siehe Abb. 4.5 Seite 27).

Das Ergebnis ist gleichzusetzen mit dem der Variante 4. Die Ressourcenanforderungen dürften hier allerdings höher sein, da sämtliche Komponenten 2 mal instanziiert werden. Eine Synchronisation gemeinsamer Zugriffe im Systemstromdemultiplexer muß hier nicht erfolgen. Stattdessen

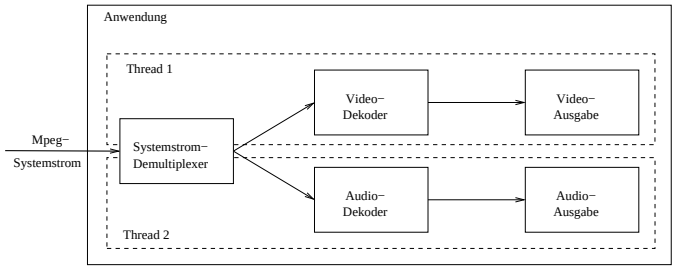

Abbildung 4.2: Variante 2: Ein Thread für Video, ein Thread für Audio

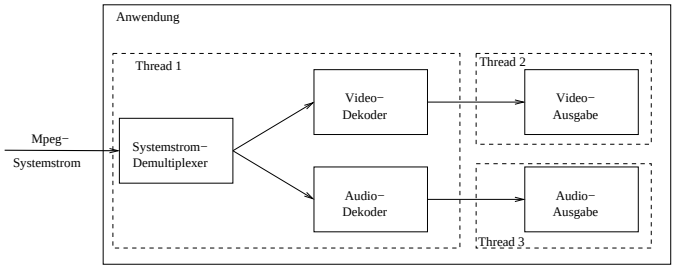

Abbildung 4.3: Variante 3: Ein Thread zum Dekodieren und jeweils ein Thread zum Anzeigen

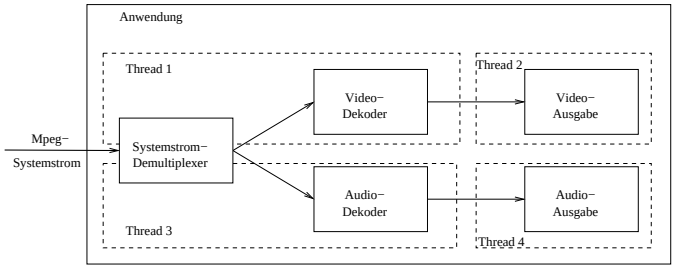

Abbildung 4.4: Variante 4: Je ein Thread zum Dekodieren von Audio und Video, sowie je ein Thread zum Anzeigen

verlagert sich dieses Problem auf das Betriebssystem, weil beide Prozesse auf dieselbe Quelldatei zugreifen.

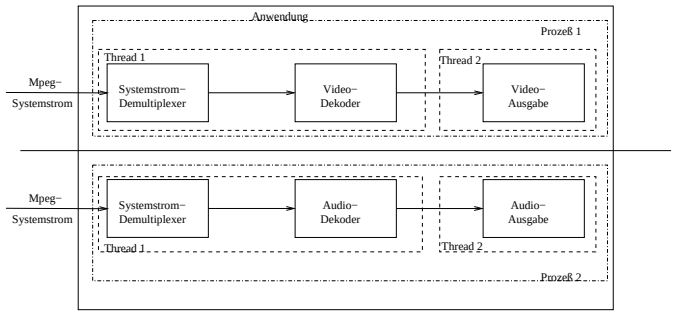

Abbildung 4.5: Variante 5: Je 1 Linux-Prozeß zum Dekodieren von Video und Audio, sowie je ein Thread zum Anzeigen

Fazit Varianten 4 und 5 erzielen die besten Ergebnisse. Wegen ihres einfacheren Aufbaus und geringeren Ressourcenanforderungen wurde Variante 4 als Grundlage für die Entwicklung der DROPS-Anwendung ausgewählt. Abbildung 4.6 zeigt eine Gegenüberstellung der Varianten. Als Teststrom diente ein MPEG-1 Systemstrom mit einer Nenn-Video-Framerate von 29.97/s und einer Bitrate von 1.12 MBit/s.

4.1.2 Der `mpeg_x11_player`

Auf Grundlage des `mpeg_x11_viewers` entstand eine Anwendung, die Video- und Audiodaten synchron wiedergeben kann. Die Videoausgabe benutzt dabei das X-Windows System. Sound wird wahlweise in eine Datei, oder nach `/dev/dsp` ausgegeben. Alle o.g. Varianten wurden implementiert und können über Kommandozeilenparameter aktiviert werden. Desweiteren stehen Kommandozeilenoptionen zur Verfügung, die eine zeitverschobene bzw. gedehnte Wiedergabe ermöglichen.
Aufruf: `mpeg_x11_player [Parameter] Quelldatei Soundausgabe`
Für eine Liste der Kommandozeilenparameter siehe Tabelle 4.1.

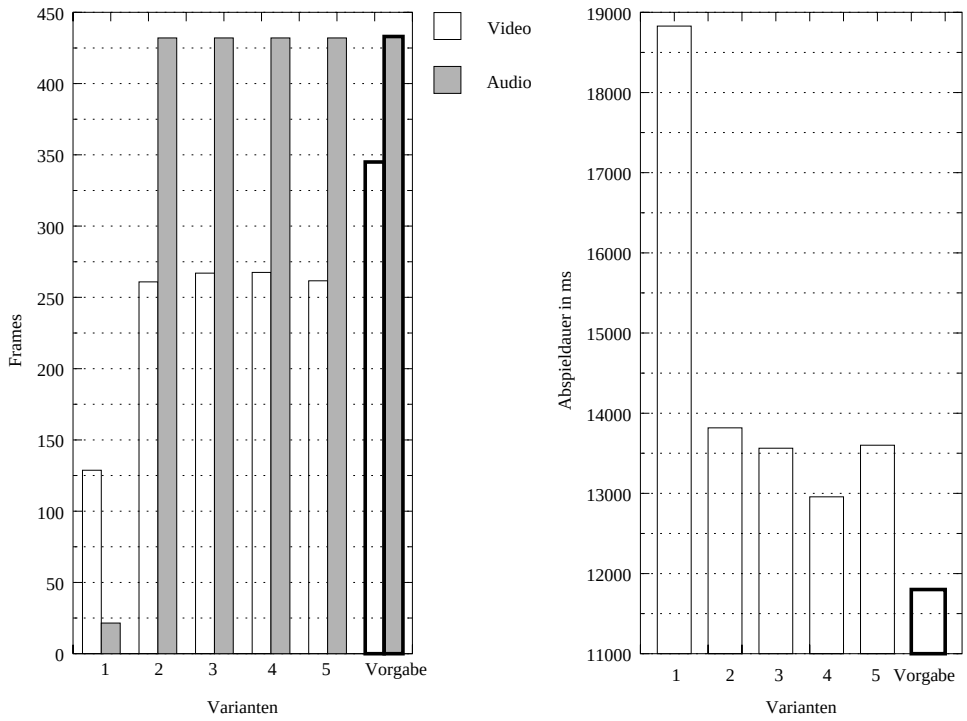

Abbildung 4.6: Die Varianten im Vergleich:
links: Anzahl der Frames, die im Präsentationszeitraum dargestellt wurden
rechts: Zeiten zur Darstellung aller Frames
Vorgabe: Anzahl der Frames (links) bzw. Präsentationszeitraum (rechts) des Teststroms

Parameter	Bedeutung
-0	benutze Variante 1
-1	benutze Variante 2
-2	benutze Variante 3
-3	benutze Variante 4
-4	benutze Variante 5
-d x	warte x Sekunden bis zum Start der Wiedergabe
-f x	verzögere Wiedergabe um Faktor x
-n	Videodarstellung mit halber Größe
-o x	beginne Wiedergabe ab Zeitpunkt x (ca.)

Tabelle 4.1: Kommandozeilenparameter des `mpeg_x11_player`

Kapitel 5

Leistungsbewertung

5.1 Leistung der Audiokomponente

Zur Bestimmung der Leistung der Audiokomponente wurden Tests mit einem MPEG-Audiostrom durchgeführt, der folgende Parameter aufwies:

- MPEG-Standard: 1; Layer II
- Datenrate: 224 kbit/s,
- Sample-Rate: 44100 Hz Stereo
- Abspieldauer laut Präsentationsinformationen: 10.02s

Dieser Strom war zuvor aus einem MPEG-Systemstrom gewonnen worden.

Testkandidaten waren die schon unter 3.1.1 auf Seite 11 aufgeführten Programme, sowie die im Rahmen dieses Belegs entwickelte Smart-MPEG-Sound-Komponente in Verbindung mit Variante 4 der Anwendungskomponente (siehe 4.1 Seite 23). Für die Smart-MPEG-Sound-Komponente wurde ein zusätzlicher Testlauf durchgeführt, bei welchem der ursprüngliche MPEG-Systemstrom als Eingabe diente.

Als Testplattform diente wieder der unter 4.1 Seite 23 angeführte Rechner.

Im Resultat des Tests (siehe Abb. 5.1 Seite 31) ist zu erkennen, daß die Smart-MPEG-Sound-Komponente im Vergleich zu der ihr zugrunde liegenden `mpegsound`-Bibliothek Leistungseinbußen von gut 40% aufweist und somit schlechter abschneidet als die anderen getesteten Programme. Die Ursachen für dieses schlechte Abschneiden sind in zusätzlichen Kopieraktionen

der dekodierten Audiodaten zur Übergabe an `Mpeg_audio_frame`-Objekte und dem erhöhten Aufwand, der durch die Verwendung der `Mpeg_buffer` (siehe 2.3.2 Seite 7) entsteht, zu suchen.

Unerwartet war der fehlende Leistungsabfall bei Verwendung eines MPEG-Systemstromes anstelle eines MPEG-Audio-Stromes als Eingabe der Smart-MPEG-Soundkomponente. Dieses Verhalten ist nur dadurch erklärbar, daß Eingabedaten vom Betriebssystem gepuffert wurden. Für längere Ströme ist hier eine leichte Verschlechterung zu erwarten.

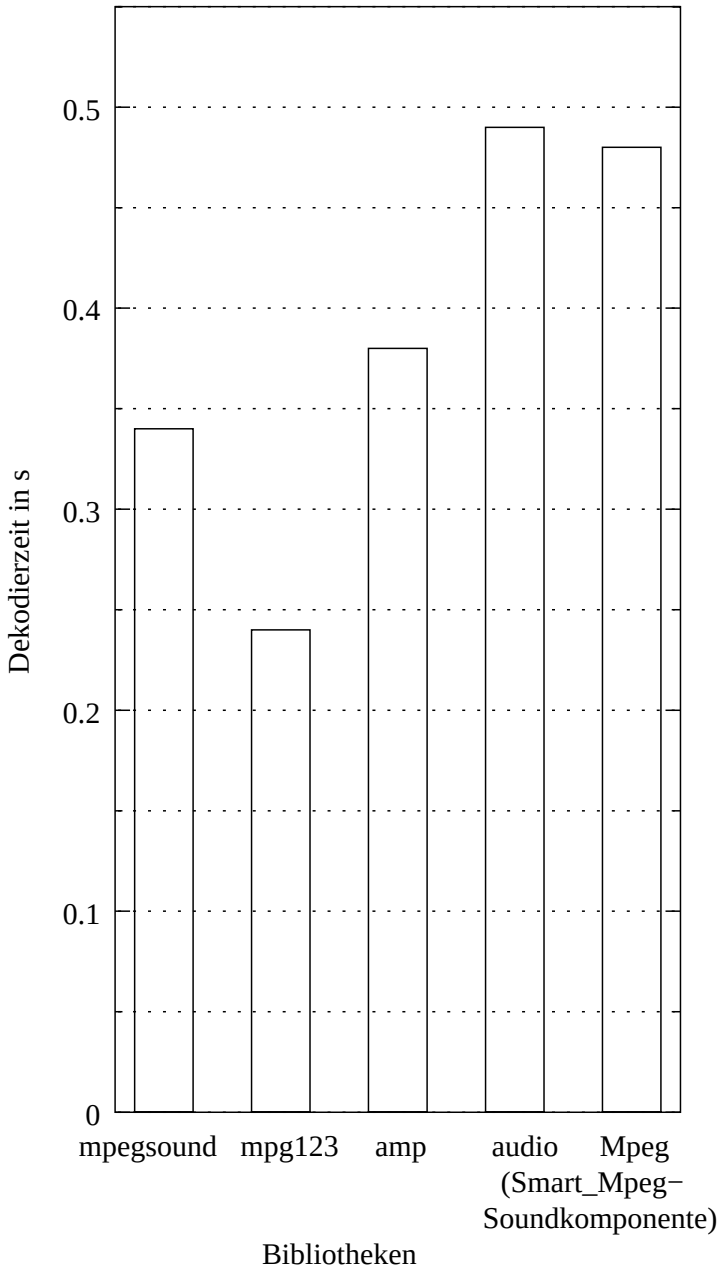

Abbildung 5.1: Dekodierzeiten der Soundkomponente im Vergleich mit anderen Programmen. Die beiden rechten Balken zeigen den Vergleich zwischen Verwendung eines Audiostroms bzw. MPEG-Systemstroms.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Das im Rahmen dieses Beleges gestellte Ziel der Entwicklung einer Sound- und Synchronisationskomponente und deren Einbindungen in das Smart-MPEG-Projekt wurde erreicht. Sowohl unter Linux, als auch unter DROPS sind diese Komponenten funktionsfähig.

Die Aufgabe, für jedes der beiden Systeme eine Anwendung zu entwickeln, wurde nur zum Teil erfüllt. Für die Linux-Plattform entstand eine Anwendung, die die Funktionalität der Smart-MPEG-Bibliothek nutzt und in verschiedenen Varianten umsetzt. Die leistungsfähigste Variante bildete die Grundlage zur Entwicklung der DROPS-Anwendung.

Die DROPS-Anwendung konnte nicht fertiggestellt werden, weil ein allgemeingültiger Sound-Treiber für diese Umgebung noch nicht existiert. Stattdessen wurde ein vorhandener Treiber für das IBM-T20 Notebook abgeändert, sodaß die Anzahl der übertragenen Bytes ermittelt wird. Weiterhin blieb die Frage der Echtzeitfähigkeit dieser Anwendung auf der Strecke.

6.2 Ausblick

Für die weitere Arbeit am Smart-MPEG-Projekt sollte die Vervollständigung der DROPS-Anwendung im Vordergrund stehen. Dafür muß zunächst ein allgemeingültiger Sound-Treiber entwickelt werden.

Weiterhin wäre die Erweiterung der Smart-MPEG-Bibliothek um Funktionen zum Vor- und Zurückspulen des MPEG-Stroms wünschenswert.

Eine nützliche Ergänzung der `Mpeg_presentation_timer`-Schnittstelle wäre eine Funktion, die es erlaubt, den Präsentationszeitpunkt in Werten der Systemzeit zu erfragen. Mehrfache Aufrufe von `early()` könnten so entfallen, und eine flexiblere Gestaltung der Ausgabestrategie durch den Anwender wäre möglich.

Ein weiterer Gesichtspunkt ist die Vereinheitlichung von `Mpeg_video_pres_timer`, `Mpeg_audio_pres_timer` und der für die Videokomponente entwickelten, in dieser Arbeit jedoch nicht vorgestellten `Mpeg_prio_timing`-Schnittstelle, um das Design zu vereinfachen.

Literaturverzeichnis

[1] Alan Au; Gernot Heiser. *L4 User Manual, Version 1.14*. School of Computer Science and Engineering, The University of New South Wales, Sydney 2052, Australia, March 1999.

[2] The Flux Research Group. *Flux Operating System Toolkit Version 0.97*. `http://www.cs.utah.edu/projects/flux/`, Department of Computer Science, University of Utah, January 1999.

[3] Tomislav Uzelac. *amp Version 0.7.6*. `http://www-users.cs.umn.edu/~wburdick/gamp/`, 1996,1997. `gamp` ist eine freie `amp`-Implementierung.

[4] Michael Hipp; Oliver Fromme. *mpg123*. `http://www.mpg123.de`.

[5] Bram Avontuur. *mp3blaster Version 2.0b6*. `http://www.stack.nl/~brama/mp3blaster.html`, 1997–2001.

[6] Woo jae Jung. *mpegsound Bibliothek*. `jwj95@nownuri.net`, 1997. Bestandteil von mp3blaster.