Berufsakademie Sachsen
Staatliche Studienakademie Leipzig

Untersuchung und Optimierung von Algorithmen zur parallelen Aufnahme und Wiedergabe von Videodaten im DV-Format

Diplomarbeit
zur Erlangung des akademischen Grades eines
Diplom-Informatikers (BA)
in der Studienrichtung Informatik

Eingereicht von: Christian Gräfe
Hahnemannstraße 22
04177 Leipzig
Seminargruppe: IT2005
Matrikelnummer: 205830

Betreuer: Dipl. Informatiker Markus Maspfuhl
CCC Campus-Computer-Center GmbH
Ring 24
04416 Markkleeberg

Leipzig, den 02.06.08

Inhaltsverzeichnis

1 Einleitende Worte

1.1 Einführung

Als 1996 die ersten Produkte für Digital-Video auf den Markt kamen, stand die digitale Filmbearbeitung am PC noch in den Kinderschuhen. Ein wesentlicher Grund dafür war der hohe Anspruch an die Hardware und der große Speicherbedarf von digitalisiertem Videomaterial. Der 1994 eingeführte DV-Standard sollte dies ändern.

DV-Daten benötigen nur etwa 10% des Speicherplatzes ihres analogen Ausgangssignals. Durch diese Komprimierung ist es möglich, hochwertige Videos, bei geringer Qualitätseinbuße, auch an einem normalen PC zu bearbeiten und wiederzugeben.

Durch die etwa gleichzeitig entstandene Schnittstelle IEEE 1394, auch FireWire genannt, ist es möglich, die Daten einer DV-Kamera auf den PC zu übertragen bzw. das Bild der Kamera am PC anzuzeigen und aufzuzeichnen.

Ein Anwendungsgebiet, bei dem Live-Videodaten einer DV-Kamera am PC aufgezeichnet werden, ist im Bereich Sport zu finden. Bei der Videoanalyse von Sportereignissen ist das Aufnehmen und anschließende Auswerten von Videos von großer Bedeutung. Dies kann sich allerdings je nach Sportart als sehr zeitaufwendig erweisen.

Das Videomaterial muss erst komplett aufgenommen werden und kann dann erst, mit Hilfe von spezieller Software, ausgewertet werden. Je nach Genauigkeit der Analyse, kann die Auswertung nochmal das Drei- bis Vierfache der originalen Aufnahmezeit in Anspruch nehmen. Hier wäre es von Vorteil, wenn man das Video bereits während der Aufnahme analysieren könnte.

Mit Hilfe des Programms utilius VS, der CCC Campus-Computer-Center GmbH, ist genau dies möglich. Die Software ermöglicht es, Szenen im Video zu markieren und für diese Szenen Merkmale zu vergeben. Diese Merkmale können später jederzeit ausgewertet werden. Wird das Video mit dem Programm aufgezeichnet, kann diese Markierung sofort, während der Aufnahme, erfolgen. Dadurch ist es möglich, den Zeitaufwand für die Auswertung des Videomaterials zu verkürzen. Da wichtige Szenen markiert sind, ist es nicht nötig, nochmals das gesamte Videomaterial durchzusehen. Es wird nur noch mit den markierten Szenen gearbeitet und diesen Merkmale zugeordnet.

1.2 Problemstellung und Ziel der Arbeit

Eine Verkürzung des Zeitaufwands für die Gesamtauswertung des Videomaterials ist nur erreichbar, wenn mit der Auswertung schon während der Aufnahme begonnen wird. Damit das Video wirklich zeitgleich zur Aufnahme ausgewertet werden kann, ist es unabdingbar, dass man die Wiedergabe des Videos pausieren kann und sich in dem bereits aufgenommenen Material vor und zurück bewegen kann.

Eine während der Aufnahme markierte Szene kann somit gleich, während die Aufnahme weiter läuft, angesehen und ausgewertet werden. Vor allem, wenn für eine Szene eine Vielzahl an Merkmalen vergeben werden muss, ist es unumgänglich, sich diese mehrmals anzusehen bzw. sie sogar langsamer oder Bild für Bild anzuzeigen.

Ziel dieser Diplomarbeit ist es, ein System zu entwickeln, was genau dies ermöglicht, die Aufnahme von DV-Daten und die gleichzeitige Wiedergabe dieser. Dabei muss es allerdings möglich sein die Wiedergabe zu pausieren und ihre Geschwindigkeit zu verändern. Darüber hinaus sollte es möglich sein, sich im dem bereits aufgezeichneten Material vor und zurück zu bewegen.

Das Zielprodukt dieser Arbeit ist ein System, welches in die Software utilius VS eingebunden werden kann. Darüber hinaus muss es aber auch ohne Probleme mit anderer, ähnlicher Software funktionieren.

Da diese Diplomarbeit auf DirectShow und das Component Object Model (COM) aufbaut, wird es ohne Probleme möglich sein, das entwickelte System oder nur Teile davon in anderer Software als utilius VS zu verwenden.

1.3 Aufbau der Arbeit

Diese Diplomarbeit gliedert sich in mehrere große Abschnitte. Nach dieser Einleitung folgt die Erläuterung von Grundlagen, welche für das Verstehen dieser Arbeit unerlässlich sind. In den Grundlagen werden die Themen Digital-Video, COM und DirectShow, insbesondere die Filterprogrammierung, erklärt.

Die nächsten Abschnitte folgen den Schritten der klassischen Softwareentwicklung, Analyse, Entwurf, Programmierung und Test bzw. Optimierung.

Anschließend an die Grundlagen wird eine bereits bestehende Lösung vorgestellt und Analysiert. Es werden Vor- und Nachteile, sowie Abgrenzungspunkte, zu dem angestrebten Ziel dieser Arbeit erläutert.

Daraus folgend wird ein grobes Konzept für ein eigenes System erarbeitet. Dieses grobe Konzept wird im darauf folgendem Kapitel Punkt für Punkt verfeinert.

Die Umsetzung dieses Konzeptes in Programmcode wird in ausgewählten Punkten dargestellt und erklärt.

Im Anschluss an die Umsetzung werden einige Testfälle demonstriert. Anhand dieser Testfälle werden Vorschläge für die Optimierung des Systems gegeben.

Im letzten Abschnitt wird schließlich das erarbeitete Ergebnis nochmals zusammengefasst. Des Weiteren werden Hinweise für eine mögliche Erweiterung des Systems gegeben.

2 Grundlagen

2.1 Digital Video

Digital Video (DV) ist der Oberbegriff für den 1994 eingeführten DV-Standard (damals: „Blue Book“, heute: IEC 61834). Er ist mittlerweile der Standard für die Heim- und Semiprofessionelle Videoproduktion geworden. Der Standard definiert den Codec und den Kassettentyp, mit dem die Aufzeichnung erfolgt.[1]

Die Kompression des Videomaterial erfolgt im ersten Schritt durch *Croma Subsampling*. Hierbei wird das Ausgangsmaterial aus dem RGB-Farbraum in den YcbCr-Farbraum überführt und auf 4:2:0 (NTSC: 4:1:1) herunter gerechnet. Als zweiter Schritt erfolgt eine Kompression durch Anwendung der d*iskreten Kosinustransformation* (DCT) bei einer Konstanten Bitrate von 25 Mbit/s. Bei der Kompression handelt es sich um eine Intraframe Kompression, d.h. dass jeder Frame unabhängig von anderen berechnet wird (I-Frame, siehe Kapitel 2.4.1).

Bei den Audiodaten handelt es sich üblicherweise um zwei Kanäle (Stereo), mit einer Auflösung von 16 Bit und einer Samplerate von 48 kHz. Vier Kanäle mit 12 Bit sind ebenfalls möglich.

Die Datenrate des Materials, also die Videodaten, die Audiodaten, sowie Fehlererkennungs- und Fehlerkorrekturdaten, beläuft sich auf etwa 35 Mbit/s.

Die Verbindung der Kamera mit dem PC erfolgt üblicherweise über den gleichzeitig entwickelten Standard IEEE 1394 (auch FireWire oder iLink genannt). Über diese Verbindung können nicht nur die Videodaten übertragen werden, sondern auch Steuersignale. Diese ermöglichen es, die Kamera über den PC zu kontrollieren.

Die übertragenen Videodaten werden in einem Containerformat gespeichert. Dabei handelt es sich meist um AVI oder QuickTime. Bei AVI wird dabei zwischen zwei Typen unterschieden. Bei dem einen wird nur das multiplexte Audi/Video Signal gespeichert (Typ 1) und bei dem anderen werden die beiden Signale getrennt gespeichert (Typ 2). Heute üblich ist die Verwendung von Typ1. Typ2 wird nur zu Kompatibilitätszwecken unterstützt, da „Video for Windows“, der Vorläufer von DirectShow, nur mit diesem Format arbeiten kann.

1 nach Wikipedia: DV [WIKI_1]

2.2 Component Object Model

2.2.1 Allgemeines Konzept

Das *Component Object Model* (COM) ist die logische Schlussfolgerung aus der Objekt-orientierten Programmierung. Einzelne Programmteile werden in Module ausgelagert, die daraufhin universal eingesetzt werden können. Dabei ist es vollkommen egal in welcher Programmiersprache das Modul verfasst wurde, es kann in jedes beliebige andere Programm eingebunden werden.

COM-Objekte werden bei Bedarf dynamisch geladen. Aus diesem Grund müssen die vorhandenen Objekte im System registriert werden. Den Objekten wird ein *Globally Unique Identifier* (GUID) zugeordnet. Dabei handelt es sich um ein 128 Bit Integer, welcher garantiert eindeutig ist.[2]

Die Unabhängigkeit von COM-Objekten zur Programmiersprache in der sie verfasst sind, wird erreicht durch die Verwendung von *virtual function tables* (vtables). Ein COM-Objekt hat nach Außen hin nur Funktionen und keine Attribute. Die vtables enthalten Pointer auf diese Funktionen. Somit können alle Programmiersprachen die Funktionen über Pointer aufrufen, zum Beispiel C, C++, Small Talk, Ada, und selbst Basic, von COM profitieren.

Für die Kommunikation mit COM-Objekten werden *Interfaces* verwendet. Ein Objekt stellt seine Funktionen über Interfaces zur Verfügung. Durch diesen Mechanismus wird eine Kapselung der Daten im Objekt erreicht. Darüber hinaus ermöglicht es die Verwendung von Interfaces, dass COM-Objekte beliebig erweiterbar sind. Stellt ein Objekt neue Funktionen zur Verfügung, werden diese in einem neuen Interface zusammengefasst, wobei die alten Funktionen über die bestehenden Interfaces angesprochen werden können.

Jedes COM-Objekt stellt das Interface IUnknown zur Verfügung. Dieses beinhalten die Funktionen AddRef und Release, welche für das *Reference Counting* zuständig sind. Als wichtigste Funktion von IUnknown gilt allerdings QueryInterface. Der Funktion wird eine Interface-ID (IID), welche ebenfalls eine GUID ist, übergeben. Unterstützt das Objekt dieses Interface, erhält man den Pointer zu diesem, wird es nicht unterstützt, gibt die Funktion E_NOINTERFACE zurück.

2 nach MSDN: The Component Object Model: A Technical Overview [COM_1]

2.2.2 Microsoft Interface Definition Language

Zur Beschreibung einer Schnittstelle, also ihrer Funktionen und deren Parameter, gibt es die *Interface Definition Language* (IDL). Für Windows-Betriebssysteme gibt es die Microsoft IDL (MIDL), sie ermöglicht die Definition von COM und „remote procedure call“ (RPC) Schnittstellen.[3]

Der Syntax der IDL ist stark an die C-Syntax angelehnt. Bei der Definition einer Schnittstelle, sowie der Definition von Funktionen in dieser Schnittstelle, spielen Attribute eine wichtige Rolle. Das Attribut [out] gibt zum Beispiel an, dass über einen Parameter einer Funktion Daten zurückgegeben werden. Auf diese Weise kann nicht nur die Richtung des Datenflusses angegeben werden, sondern diverse weitere Eigenschaften. Um einen Parameter Beispielsweise als Text zu kennzeichnen, wird das Attribut [string] verwendet. Ein wichtiges Attribut, was direkt am Interface gesetzt wird, ist [uuid], es gibt die IID des Interfaces an.

Der MIDL-Compiler unterstützt die Erzeugung von Typ-Bibliotheken. In ihnen werden eine oder mehre Schnittstellen zusammengefasst und Binär, in einer Datei, gespeichert. Darüber hinaus können in so einer Typ-Bibliothek auch Informationen über COM-Objekte gespeichert werden. Diese Datei kann dann in unterschiedliche Programmierumgebungen eingebunden werden.

3 nach MSDN: Microsoft Interface Definition Language [COM_2]

2.3 DirectShow

2.3.1 Allgemeines und Geschichtliches

Bei DirectShow handelt es sich um eine Reihe von COM-Schnittstellen und Objekte, die von Microsoft zur Verfügung gestellt werden. Die meisten Abspielprogramme für Audio oder Video unter Windows, wie zum Beispiel der MediaPlayer, nutzen DirectShow. Es stellt, ähnlich wie QuickTime auf dem Mac oder gstreamer unter Linux, Funktionen für das Lesen, Verarbeiten und Schreiben von Audio- und Videodaten zur Verfügung. Es bietet außerdem Decoder für verschiedene Standardformate, wie zum Beispiel AVI, ASF, MPEG, WMV sowie MP3, WMA und WAV.[4]

DirectShow gibt es schon länger als DirectX. Microsoft veröffentlichte Ende 1992 *Video for Windows*. Es legte den Grundstein für die Wiedergabe von Videodaten auf dem Windows-Betriebssystem. Video for Windows wurde 1996 durch das COM-Basierte ActivMovie ersetzt, welches schließlich 1997 in DirectX5 einging. Nach der Umbenennung in DirectShow wurde es erst als Hauptbestandteil des DirectMedia SDK verbreitet und mit der Veröffentlichung von DirectX7 schließlich vollständig integriert. Seit 2005 ist DirectShow nicht mehr Bestandteil des DirectX SDKs. Es ist nun im Windows Plattform SDK enthalten und soll voraussichtlich durch *Media Foundation* abgelöst werden.[5]

4 MSDN: Introduction to DirectShow [DS_1]
5 Wikipedia: DirectShow [WIKI_2]

2.3.2 Filterkonzept

Da DirectShow, auf dem Component Object Model basiert, ist es sehr leicht zu erweitern. In DirectShow werden so genannte *Filter* für die Wiedergabe der Daten genutzt. Bei jedem Filter handelt sich sich um ein COM-Objekt. Sie werden zur Verarbeitung der Daten in einem *Filtergraphen* kombiniert. Es gibt mehrere Filtertypen[6]:

- *Source Filter* erzeugen den Inputstream für den Filtergraphen. Die Daten können aus einer Datei, einem Internet- bzw. Netzwerkstream oder von einem angeschlossenem Gerät kommen. Durch die enge Zusammenarbeit mit dem Windows Driver Model (WDM) kann multimediale Peripherie direkt als Source Filter ausgewählt werden. Source Filter können nochmal in zwei Typen unterteilt werden[7]:
    - *Push*: es werden fortlaufend Daten auf den Output Pin ausgegeben. Dies trifft für Live-Quellen, wie ein Internetstream oder eine Kamera zu.
    - *Pull*: die Daten können bei Bedarf am Output Pin gelesen werden. Ein Pull-Filter tritt immer in Verbindung mit einem *Parse Filter* auf. Dieser sendet die Anfrage für die Daten.
- *Transform Filter* verändern den Datenstrom, der durch sie hindurchgeht. Der Filter bekommt Daten aus einem anderen Filter, verändert diese und gibt sie an den nächsten Filter weiter. Transfom Filter werden zum Beispiel zum Decodieren und Encodieren eines Videosignals eingesetzt. Unter die Gruppe der Transform Filter fallen noch zwei weitere Filtertypen:[8]
    - Der *Splitter Filter* teilt einen Eingangsstrom in mehre Ausgangsströme auf. Meist handelt sich dabei um die Zerlegung in Audio und Video.
    - Das Gegenstück dazu ist der *Mux Filter*. Er verbindet mehrere Eingangsströme zu einem Ausgangssignal. Es können Audio- und Videodaten verbunden oder mehrere Videosignale kombiniert werden.
- *Renderer Filter* geben den Datenstrom aus. Die Daten werden auf den Bildschirm, den Lautsprecher oder sogar in eine Datei ausgegeben. Da DirectShow auch DirectDraw (DirectX Graphics) und DirectSound nutzt, kann es die Daten ohne Umwege direkt auf die Hardware ausgeben.

6 nach [DS_2], Part I, Chapter 1, Abs.: Filters
7 nach [DS_2], Part III, Chapter 12, Abs.: Source Filter Types
8 nach MSDN: About DirectShow Filters [DS_3]

2.3.3 Datenfluss im Filtergraph

Ein Filtergraph besteht immer aus mindestens einem Source Filter und wenigstens einem Renderer Filter, meist kommen noch ein oder mehr Transform Filter zum Einsatz. Diese Filter sind verbunden und kommunizieren über *Pins*, welche von einem Filter genau definiert sind. Source Filter haben nur Output Pins und Render Filter nur Input Pins. Transform Filter haben mindestens einen Input und einen Output Pin.

Dabei ist zu beachten, dass nicht jeder Output Pin mit irgend einem Input Pin verbunden werden kann. Es wird vom Filter genau festgelegt, welche Art von Daten über den Pin ausgetauscht werden und auf welche Art und Weise der Austausch geschieht. Dadurch ist es zum Beispiel nicht möglich, einen Output Pin, der MPEG-1 Daten liefert, mit einem Input Pin zu verbinden, der Digital Video (DV) Daten voraussetzt. So eine Verbindung würde den Datenstrom unbrauchbar machen.[9]

Die Art der Daten, die unterstützt werden, wird in zwei Stufen angegeben, zum einen der Haupttyp (mediatype), er gibt an, ob es sich zum Beispiel um Audio- oder Videodaten handelt. Dieser Typ wird dann nochmals genauer spezifiziert (mediasubtype). Auf diese Art wird zum Beispiel bei einem Video-Haupttypen im mediasubtype das Farbformat mit angegeben, also ob es sich um 4:4:4, 4:2:2, 4:1:1 usw., handelt.

Zusätzlich dazu werden bei der Verbindung der Pins noch weitere Detaildaten übermittelt. Mit Hilfe einer BITMAPINFOHEADER-Struktur kann zum Beispiel die Bitmaske für die RGB-Farbwerte angegeben werden. Ähnlich dazu gibt es auch eine DVINFO-Struktur, über die es zum Beispiel möglich ist, herauszufinden, ob es sich um PAL oder NTSC handelt.

Die Mediadaten im Filtergraphen fließen immer vom Source zum Render Filter. Dabei werden die Daten meist über das Push-Model weitergegeben, das Pull-Model ist aber ebenso möglich.

9 nach [DS_2] Part I, Chapter 1, Abs.: Connections Between Filters

2.4 Containerformate für Videodaten

2.4.1 Allgemeines

Für die Wiedergabe von Videos auf dem PC ist es erforderlich, dass die Videodaten in einer Datei vorliegen. Bei solch einer Datei handelt es sich immer um ein Containerformat. Die Video- und Audiodaten sind im Allgemeinen in getrennten Spuren in diesem Container gespeichert und unterliegen einer Komprimierung.

Bei der Wiedergabe der Datei werden die zwei Spuren wieder verbunden. So ist es möglich, dass für ein Video mehrere Tonspuren hinterlegt sein können. Je nach Containerformat können auch Untertitel, Kapitalinformationen und Menüs, ähnlich der DVD, gespeichert werden.

Bei der Speicherung in solch einem Containerformat kommt ein Multiplexer zum Einsatz. Es werden dabei nicht nur die komprimierten Rohdaten gespeichert, sondern auch wichtige Struktur- und Zeitinformationen. Diese Informationen benötigt der Demultiplexer bzw. Splitter bei der Wiedergabe der Datei. Die Zeitdaten sind wichtig, damit Video und Ton synchron laufen können. Die Strukturdaten stellen nicht nur den Aufbau des Containers dar, sondern sind vor allem für die verwendete Komprimierung wichtig, da mit ihnen meist die Kennzeichnung des Frametypen vorgenommen wird. Somit ist es möglich zwischen I-Frame (Key-Frame), P-Frame und B-Frame (Zwischen-Frame) zu unterscheiden.

Diese Frametypen ergeben sich aus der Art der Komprimierung des Videobildes. Bei der Intra-Frame-Komprimierung wird nur das Frame selber, unabhängig von den anderen Frames, komprimiert. Dabei handelt es sich um ein I-Frame oder auch Key-Frame. Es stellt das klassische Standbild dar.

Bei der Inter-Frame-Komprimierung entstehen nun Abhängigkeiten unter den Frames. Das P-Frame ist das Differenzbild aus dem vorangegangen Frame und dem eigentlichen Bild, daher benötigt es weniger Speicherplatz als ein I-Frame. Den wenigsten Platz benötigen die so genannten B-Frames. Das B steht für Bidirektional. Bei der Berechnung eines B-Frames werden die Informationen aus vorangegangen und folgenden Frames berücksichtigt. Bekanntestes Verfahren dafür ist die MPEQ-1 Komprimierung.[10]

10 nach Wikipedia: MPEG-1 – Videokodierungsverfahren [WIKI_3]

2.4.2 AVI-Container

Einer der ältesten und meist verbreitetsten Container für Multimediadaten ist das *Audio Video Interleave* (AVI) Format. Es basiert auf dem *Resource Interchange File Format* (RIFF). Dabei werden die Daten in so genannten Chunks und Listen gespeichert.

Ein Chunk beginnt immer mit einem *four-character code* (FOURCC), gefolgt von einem 32-Bit Wert, der die Größe des Datenblocks angibt. Daraufhin folgen die Daten oder weitere Chunks oder Listen. Es kann also beliebig geschachtelt werden.

Eine Liste ist eine Ansammlung von aufeinander folgenden Chunks bzw. Listen. Eine Liste beginnt mit dem FOURCC 'LIST', gefolgt von einem 32-Bit Wert mit der Größe und einem weiterem FOURCC, welcher den Listentyp angibt. Daraufhin folgen die eigentlichen Daten der Liste. Jede AVI-Datei hat mindestens zwei Listen eine für die Kopfdaten und eine für die Videodaten.[11]

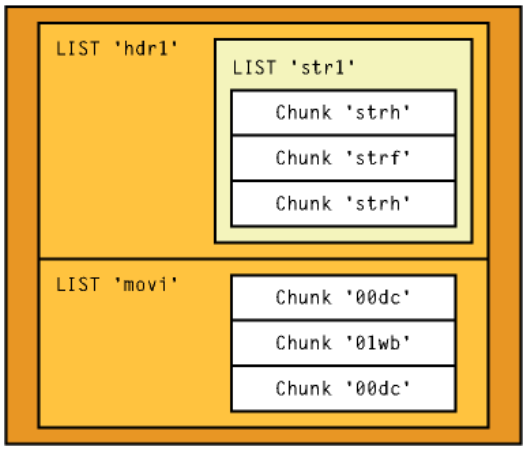

Abbildung 1: AVI-RIFF Beispiel

Eine Datei im RIFF-Format beginnt immer mit dem FOURCC 'RIFF', gefolgt von der Größe des Datenblocks und einem weiterem FOURCC, welcher den Dateitypen angibt. Der FOURCC einer AVI-Datei ist 'AVI '. Ein weiteres Beispiel für einen RIFF-Dateityp ist WAVE.

Nachteil des ursprünglichen AVI-Formates, war die Beschränkung der Dateigröße auf zwei GigaByte. Diese wird aber durch die Erweiterung OpenDML aufgehoben. Ein weiterer Nachteil besteht darin, dass die Verwendung von B-Frames nicht nativ unterstützt wird. Es gibt allerdings einige Modifizierungen die es trotzdem erlauben, Videocodecs, die B-Frames benötigen, in einer AVI-Datei zu speichern.

DirectShow bringt die benötigten Filter bereits mit, die es ermöglichen AVI-Dateien zu schreiben und zu lesen. Steht kein DirectShow zur Verfügung, gibt es noch die API „Video for Windows“, also den Vorgänger von DirectShow. Diese API ist ganz auf das AVI-Dateiformat abgestimmt. Somit ist es möglich, auf nahezu jedem Windows Betriebssystem AVI-Dateien wiederzugeben.

11 nach [DS_2]: Part IV, Chapter 14: Understanding The AVI File Format

2.4.3 Weitere multimediale Containerformate

QuickTime

Es gibt neben AVI noch eine Vielzahl anderer Containerformate. Vorbild und Hauptkonkurrent des AVI-Formats ist QuickTime, welches von Apple entwickelt wurde. Es bietet weit mehr Möglichkeiten als das AVI-Format, da es eine Vielzahl an multimedialem Inhalt unterstützt. Neben Audio und Video sind auch Animation, Text und Grafik, sowie Flash und 3D möglich.

Ähnlich dem AVI-Format erfolgt die Speicherung der Daten in sowohl sequenziell als auch hierarchisch organisierten Dateneinheiten, den so genannten *Atomen*. Es kann daher neben chronologischen Informationen, auch strukturelle und hierarchische Informationen und Zusammenhänge speichern. Durch einen speziellen Synchronisations-Layer eignet es sich zudem für das Streaming über ein Netzwerk.[12]

MPEG-4 Part 14 (MP4)

MP4 ist das Multimedia Containerformat des MPEG-4 Standards. Es wurde auf Basis von QuickTime entwickelt. Es kann neben den Audio und Videodaten im MPEG Format auch nahezu jede andere Art von Audio und Videocodec beinhalten.

Die Daten sind, wie bei QuickTime, hierarchisch und strukturell in Atomen organisiert. Es unterstützt die Speicherung von DRM (Digital Rights Management) geschützten Daten. Das MP4 Format eignet sich ebenfalls zum Streamen über das Netzwerk.[13]

Matroska

Der Matroska Container bietet von allen Containerformaten den größten Funktionsumfang. Eine Datei in diesem Format, kann eine Vielzahl von Audio- und Videospuren enthalten. Die Speicherung von Bildern und Untertiteln ist ebenso möglich. Somit kann zum Beispiel eine gesamte DVD nahezu eins zu eins in eine Matroska-Datei abgebildet werden.

Intern arbeitet der Container mit der, für dieses Format entwickelten, *Extensible Binary Meta Language* (EBML). Sie ist das binäre Äquivalent zu XML. Das komplette Matroska-Projekt ist Open Source und steht unter Public Domain Lizenz.[14]

12 nach Wikipedia: QuickTime - Containerformat [WIKI_4]
13 Nach Wikipedia: MPEG-4 Part 14 [WIKI_5]
14 Nach Wikipedia: Matroska [WIKI_6]

3 Vorbetrachtungen

3.1 Allgemeine Voraussetzungen

Nachdem die nötigen Grundlagen erläutert wurden, kann es an die Entwicklung einer Lösung gehen. Doch ehe dies geschieht, lohnt sich ein Blick auf eine bereits existierende Umsetzung.

Bei dieser Lösung handelt es sich um die, von Microsoft seit Windows XP SP1 zur Verfügung gestellte, *Stream Buffer Engine* (SBE). Sie ermöglicht die zeitversetzte Wiedergabe von Videomaterial während seiner Aufzeichnung. Im folgenden Unterkapitel wird das Konzept dieser Lösung vorgestellt. Anschließend daran wird erläutert, warum dieses System im vorliegenden Fall nicht verwendet werden kann.

3.2 Konzept und Funktionsweise der SBE

Für die Arbeit mit der Stream Buffer Engine werden zwei Filtergraphen benötigt. Der Sink-Graph ist für das Aufzeichnen des Videos verantwortlich, während der Source-Graph für die Wiedergabe sorgt. Die SBE besteht also aus zwei Filtern, dem „Stream Buffer Sink Filter“ und dem „Stream Buffer Source Filter“.

Der Sink Filter schreibt in eine Sammlung von temporären Dateien die Daten, die mit dem Source Filter ausgegeben werden. Dabei ist es nicht nur möglich, dass es mehrere Source-Graphen gibt, sondern auch, die Daten permanent aufzuzeichnen.

Die Größe des Zwischenspeichers kann flexibel eingestellt werden. Standardmäßig sind es sechs Dateien, die jeweils 5 Minuten aufzeichnen, also insgesamt 30 Minuten. Nur die Daten aus dem Zwischenspeicher können wiedergegeben werden. In ihnen kann sich frei vor und zurück bewegt werden.

Bei der permanenten Aufzeichnung der Daten erhält man, nach Beendigung der Aufnahme, eine Datei vom Typ dvr-ms. Bei diesem Dateitypen handelt es sich um das *Advanced Streaming Format* (ASF), ein Containerformat für Multimediadaten von Mircosoft.

Die Stream Buffer Engine unterstützt Eingangsdaten im Format MPEG2 und DV. Hierbei steht allerdings der Einsatz mit MPEG2 klar im Vordergrund, da dies das Standardformat für Digital Video Broadcasting (DVB) ist.

3.3 Probleme der SBE

Und genau darin liegt eines der Hauptprobleme bei der Verwendung der SBE für unsere Zwecke. Microsoft gibt zwar in der Referenz der Stream Buffer Engine an, dass das System mit MPEG2 und DV zusammenarbeitet[15], doch leider musste beim Testen der SBE festgestellt werden, dass die Zusammenarbeit mit DV nicht wie angegeben funktioniert.

Für diesen Zweck wurde eine einfache Testanwendung erstellt, welche die Auswahl einer Live-Quelle ermöglicht und daraufhin die benötigten Filtergraphen erstellt und die Aufnahme und Wiedergabe startet. Im Testprogramm wurden die Pausefunktion und das Vor- sowie Zurückspringen im Video getestet. Die SBE wurde so konfiguriert, dass am Ende eine Datei mit den Videodaten vorhanden ist (permanente Aufnahme).

Bei dem Ausführen des Programms mit einer DV-Kamera als Quelle wurde festgestellt, dass die Wiedergabe zwar pausiert und fortgesetzt werden kann, ein Bewegen im Video allerdings nicht möglich ist. Es konnte nur an den Beginn der Aufnahme gesprungen werden.

Ein weiteres Problem besteht in dem verwendeten Dateiformat für die Aufnahme. Damit die dvr-ms Datei wiedergegeben werden kann, wird die Stream Buffer Engine benötigt. Dabei ist vor allem die Tatsache, dass es bei der Wiedergabe der Datei immer noch nicht möglich ist, sich frei im Video zu bewegen, am schwerwiegendsten.

Auf der beigelegten CD ist das Testprogramm im Ordner„Quellcode\DsPlayer1“ inklusive Quellcode zu finden. Das Programm ist nicht nur zum Test der SBE gedacht, sondern auch als normaler Videoplayer. Wird über das Menü „Datei“ der Punkt „Stream öffnen“ angeklickt, kann eine Kamera für die Aufnahme mit der SBE ausgewählt werden. Daraufhin startet automatisch die Aufzeichnung. Die Zieldatei wird im Verzeichnis „C:/SBETest“ unter dem Namen test.dvr-ms gespeichert.

15 nach MSDN: Using the Stream Buffer Engine [DS_4]

4 Konzeptentwurf für eine eigene Umsetzung

4.1 Allgemeiner Aufbau

Um nun die Probleme der Stream Buffer Engine zu umgehen, ist ein eigenes System von Nöten, welches sich allerdings am grundsätzlichen Konzept der SBE orientieren kann.

Die Verwendung von zwei Filtergraphen, ist die beste Variante um eine effektive Trennung von Aufnahme und Wiedergabe zu erreicht. Die Speicherung der Daten kann auf diese Art und Weise optimal gesteuert werden. Die Daten können hierbei sogar mehrfach parallel und unabhängig voneinander wiedergegeben werden. Eine Alternative mit einem Filtergraphen ist auch möglich. Diese wird im letzten Unterkapitel erläutert.

Für dieses eigene System wird ebenfalls ein anderes Containerformat zur Speicherung der Videodaten benötigt. Mit der Auswahl eines passenden Formates befasst sich das folgende Unterkapitel. Im Anschluss daran werden die benötigten Filter ermittelt.

4.2 Auswahl des Containerformates

Mit dem Beenden der Aufnahme muss eine Videodatei zur Verfügung stehen, die auch ohne dieses Aufnahmesystem wiedergegeben werden kann. Damit dies gewährleistet ist, müssen die Daten der DV-Kamera in einem Containerformat gespeichert werden, welches diese Art von Videodaten unterstützt.

Die Daten sollen so wie sie kommen gespeichert werden, also nicht nochmal neu komprimiert werden. Dabei ist es vorzuziehen, die Daten als Interleaved zu speichern und somit keine Trennung von Audio- und Videospur vorzunehmen.

Es stehen mehrere Containerformate zur Auswahl, die dies ermöglichen. Zum einen das AVI-Format, welches das Standardformat unter Windows darstellt. Ebenso ist es möglich die DV-Daten in QuickTime zu speichern. Dabei handelt es sich um das Standardformat von MacOS. MP4 und Matroska sind ebenfalls möglich. Die Verwendung von Mircosofts Advanced Streaming Format ist nicht möglich, da es DV-Daten nicht direkt speichern kann, die Daten müssten erst konvertiert und neu kodiert werden.

Die Formate QuickTime, MP4 und Matroska haben den Vorteil, dass sie zum Streamen geeignet sind. Das ist für das angestrebte Ziel, des gleichzeitigen Aufnehmens und Wie-

dergebens eine wichtige Eigenschaft. Streamen bedeutet, dass die Daten wiedergegeben werden können, ohne dass die gesamte Datei vorhanden ist, was vor allem bei der Übertragung von Videos über ein Netzwerk essentiell ist. Beim AVI-Format ist dies normalerweise nicht möglich. Durch die Verwendung eines Hilfskonstruktes kann dennoch das AVI-Format genutzt werden.

Größter Nachteil bei den drei genannten Containerformaten ist ihre Abhängigkeit zu weiteren Komponenten. Damit ein normaler Windows-PC diese Formate abspielen kann, muss auf ihm zusätzlich Software installiert werden. Bei QuickTime ist dies der QuickTime Media Player. Dateien in diesem Format können dann nur in diesem Player wiedergegeben werden und nicht im Standard Media Player von Windows, was ein sehr entscheidender Nachteil ist.

Bei MP4 und Matroska verhält es sich ähnlich. Mit der Installation entsprechender Software, wie zum Beispiel FFMPEQ, können diese allerdings im Windows Mediaplayer Abgespielt werden. In diesem Fall werden entsprechende Filter im System installiert, die es jeder DirectShow Anwendung erlauben diese Formate zu lesen bzw. schreiben.

Hier kann das AVI-Format voll Punkten, da es fest in Windows integriert ist und vom System entsprechende DirecShow Filter bereitgestellt werden. Es ist somit unproblematisch Videos in diesem Format weiterzugeben, da selbst ein Windows95 diese Dateien wiedergeben kann.

Ein weiterer Vorteil des AVI-Formates liegt in seiner Struktur. Das RIFF-Format ist sehr einfach aufgebaut. Die Implementierung eines Datei-Parsers ist daher, wenn nötig, recht unkompliziert umzusetzen. Die Daten können so direkt aus der Datei gelesen werden, ohne erst den Umweg über eine zusätzliche API zu gehen.

Wichtigstes Auswahlkriterium ist die Möglichkeit, die entstanden Dateien weiterzugeben und auf anderen Rechner Abzuspielen. Hier ist, trotz einiger Nachteile, das AVI-Format am besten geeignet, da es neben Windows auch von vielen anderen Betriebssystemen nativ unterstützt wird.

Unter anderen Bedingungen wäre wohl, trotz der großen Auswahl an Containerformaten, ASF vorzuziehen. Es ist, wie das AVI-Format, in heutigen Windows Systemen integriert und, wie der Name schon sagt, zum Streamen seines Inhaltes geeignet. Microsoft bietet über DirectShow zur Zeit leider nur die Möglichkeit Windows Media Video (WMV) Daten darin zu speichern.

4.3 Benötigte Filter

4.3.1 Von DirectShow zur Verfügung gestellte Filter

Da nun feststeht, in welchem Format die DV-Daten gespeichert werden, kann über die Auswahl der benötigten DirectShow Filter nachgedacht werden.

DirectShow stellt Universalfilter für das Lesen und Schreiben von Dateien zur Verfügung. Dabei handelt es sich zum einen um den Filter „File Source“ und zum anderen um den Filter „FileWriter“.

„File Source“ ist, wie der Name schon sagt, ein Source Filter. Er arbeitet nach dem Pull-Prinzip. Um ihn für die Wiedergabe von AVI-Dateien einzusetzen wird ein weiterer Filter benötigt. Dieser findet sich unter dem Namen „AVI-Splitter“. Dabei handelt es sich nicht nur um einen Parse Filter sondern auch um einen Demultiplexer. Dieser gibt, je nach AVI-Typ, einen Video- und einen Audiostream (Typ 2) oder einen Interleaved Videostream (Typ 1) aus.

Der „File Writer“ ist ein Render Filter, welcher die Daten, die er empfängt, in eine Datei schreibt. Für die Aufzeichnung einer AVI-Datei tritt er in Verbindung mit dem Filter „AVI Mux“ auf. Dabei handelt es sich um einen Mux Filter, welcher seine Eingangsdatenströme verbindet und ausgibt. Er sorgt dafür, dass im „File Writer“ eine Datei im AVI-Format erzeugt wird.

4.3.2 Probleme

Nun ergeben sich allerdings einige Probleme mit diesen Filtern. Die Filter „File Writer“ und „File Source“ können nicht gleichzeitig auf ein und dieselbe Datei zugreifen! Die Filter öffnen eine Datei mit exklusivem Zugriffsrecht, so dass kein anderer Prozess Zugang zu dieser Datei bekommen kann. Daraus folgernd können diese Filter nicht eingesetzt werden, um das Ziel zu erreichen, da die Datei für die Videodaten parallel gelesen und geschrieben werden muss.

Ein weiteres Problem ergibt sich aus der Tatsache, dass das AVI-Format nicht für Streaming vorgesehen ist. Es muss immer erst die vollständige Datei vorhanden sein, ehe die Daten ausgegeben werden können. Somit kann der Filter „AVI Splitter“ ebenfalls nicht verwendet werden.

4.3.3 Lösung – Eigene Filter

Um diese Probleme zu umgehen, müssen eigene Filter erstellt und genutzt werden. Diese Filter müssen sich an den Funktionen der vorhandenen DirectShow Filter orientieren und die Probleme dieser umgehen.

Damit das AVI-Format als Container für die Videodaten genutzt werden kann, und diese Daten parallel auch ausgegeben werden können, muss eine weitere Bedingung gestellt werden. Der AVI-Container darf intern nur einen Datenstrom haben. Nur wenn das der Fall ist, kann gewährleistet werden, dass schon Daten ausgegeben werden können, obwohl noch keine abgeschlossene AVI-Datei vorhanden ist. Daraus folgernd muss der Eingangsdatenstrom vom Typ Interleaved Video sein. Bei der AVI-Datei handelt sich nach dem Ende der Aufnahme also um einen Typ 1.

Ist diese Bedingung erfüllt, kann ohne Probleme der Filter „AVI Mux“ genutzt werden. Der Filter empfängt dann einen DV-Frame und gibt diesen eingepackt in einen AVI-Chunk an den nachfolgenden Filter weiter. Eine Besonderheit des „AVI Mux“ liegt darin, dass er mit dem Stopp des Filtergraphen, weiter Daten schreibt. Bei den Daten handelt es sich um die AVI-Header Informationen. Erst mit diesen, kann die erzeugte AVI-Datei später wiedergegeben werden. Damit der Filter diese Daten schreiben kann, muss der Input-Pin des folgenden Filters das Interface *IStream* unterstützen. Es ist von großem Vorteil, diesen Filter nutzen zu können, da sonst selbst die Strukturdaten für die AVI-Datei geschrieben werden müssten.

Der Filter „File Writer“ muss nun also durch einen eigenen Filter ersetzt werden, der an seinem Input Pin Daten vom Typ *Mediatyp_Stream* entgegen nimmt. Diese Daten werden nun in die zu speichernde AVI-Datei geschrieben. Diese Datei darf allerdings nicht exklusiv geöffnet wurden sein. Des Weiteren muss der Input Pin das besagte IStream Interface unterstützen. Der Filter wird, äquivalent zu dem Filter in der SBE, als Sink-Filter bezeichnet.

Passend dazu wird ein Source-Filter benötigt, der die unvollständige AVI-Datei öffnen und dessen Inhalt ausgeben kann. Die Daten werden nach dem Push-Prinzip an den folgenden Filter weitergegeben. Bei den Ausgabedaten handelt es sich um DV-Frames, wie sie von der Kamera kommen. Um dies zu ermöglichen, wird der „AVI Splitter“ nicht benötigt, da die Verwaltung der Daten in einem Hilfsobjekt erfolgt und somit relativ einfach aus der AVI-Datei gelesen werden kann.

4.4 Synchronisation

Da der Source-Filter die DV-Daten aus der AVI-Datei nur ausgeben kann, wenn die Header-Informationen vorhanden sind, muss zwischen ihm und dem Sink-Filter eine Kommunikation erfolgen.

Für diesen Zweck wird ein COM-Objekt eingesetzt, welches den Lese- und Schreibzugriff auf die AVI-Datei ermöglicht. In diesem Objekt erfolgt dann die Verwaltung der bereits geschriebenen Frames. Darüber hinaus können in diesem Objekt zusätzliche Informationen, wie etwa die allgemeinen Frame-Eigenschaften, gespeichert werden. Für dieses COM-Objekt werden die Interfaces ISharedFile und ISharedDVInfo eingeführt.

Das COM-Objekt stellt für den Lese- und Schreibzugriff zwei weitere Objekte zur Verfügung. Das Objekt für den Schreibzugriff implementiert das Interface ISharedFileWriter, sowie das Interface IStream. Von diesem Objekt kann es nur eine Instanz geben. Auf diese Weise wird sichergestellt, dass nur an einer Stelle in die Datei geschrieben wird und der Schreibzugriff für andere Prozesse blockiert werden kann.

Das Objekt für den Lesezugriff implementiert das Interface ISharedFileReader. Von diesem Objekt kann es beliebig viele Instanzen geben, da das Lesen einer Datei unproblematisch ist.

Jeder dieser zwei Objekttypen hat einen eigenen FileHandler auf die entsprechende Datei. Dies ermöglicht es, in der Datei beliebig zu springen. Es können also zwei lesende Objekte, die Daten von unterschiedlichen Positionen in der Datei ausgeben, während das schreibende Objekt Daten an die Datei anfügt.

Ein schematischer Aufbau des ganzen Systems, findet sich in der Abbildung auf der nachfolgenden Seite. Als Namenskonvention für die Objekte wurde das Kürzel 'TS', wie TimeShift, gefolgt vom Namen des Hauptinterfaces, ohne das 'I' allerdings, genutzt. Für die Filter gilt ähnliches, nur dass hier, äquivalent zur Stream Buffer Engine, die Funktion als Namensgeber dient.

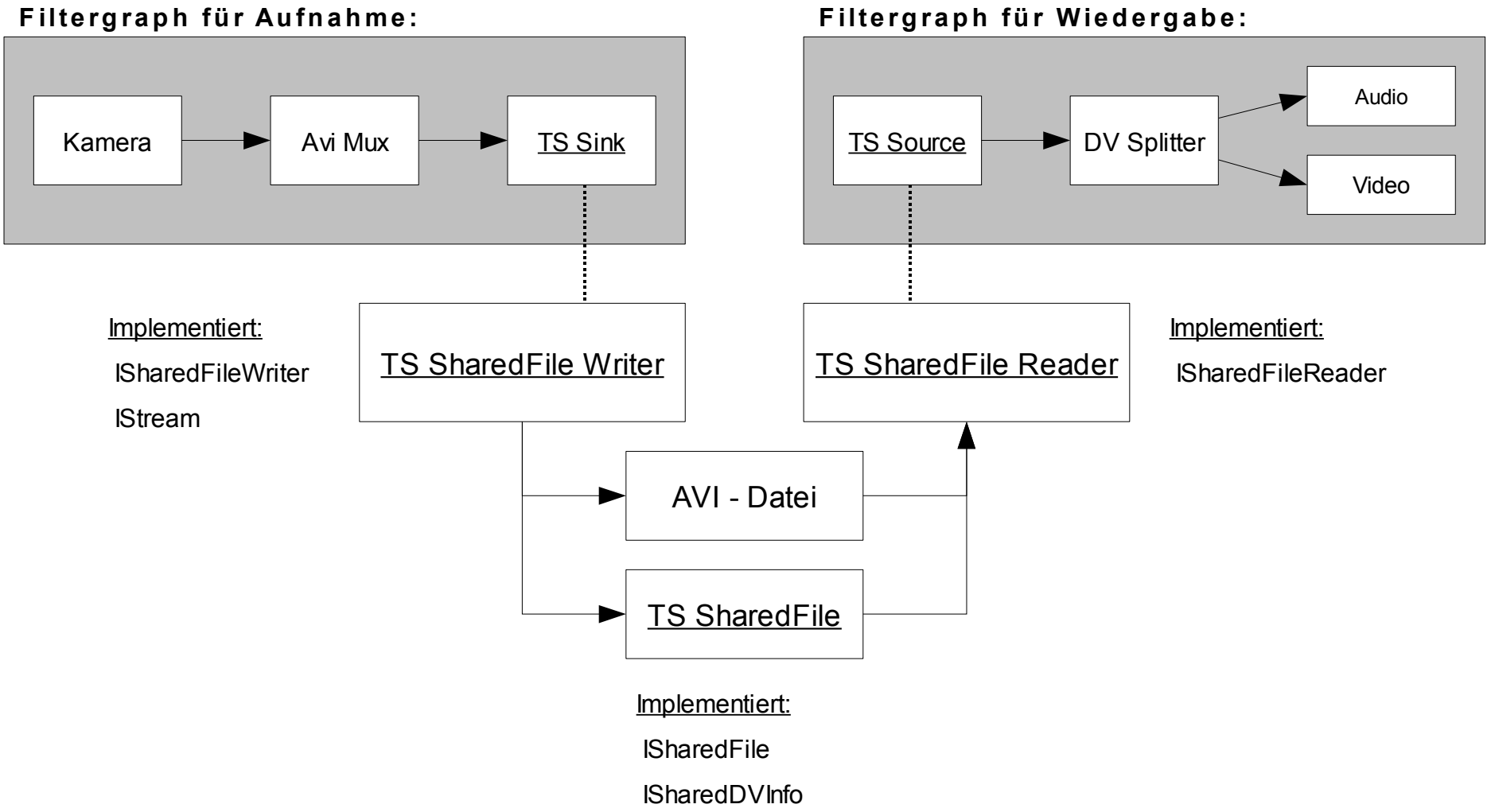

Abbildung 2: Schematische Darstellung des Konzeptes

4.5 Alternative mit einem Filtergraphen

Neben dem gerade vorgestellten Konzeptentwurf mit zwei Filtergraphen, ist auch eine alternative Variante mit nur einem Filtergraphen denkbar.

Aus den zweit Filtern, TS Sink und TS Source, wird dann ein einziger Filter. Dabei handelt es sich dann, theoretisch gesehen, um einen Transform Filter, obwohl die Daten nicht direkt hindurch fließen oder geändert werden.

Bei diesem Filter übernimmt der Input Pin die Aufgabe des Sink Filters und der Output Pin die Aufgabe des Source Filters. Die Funktionsweise bleibt also gleich.

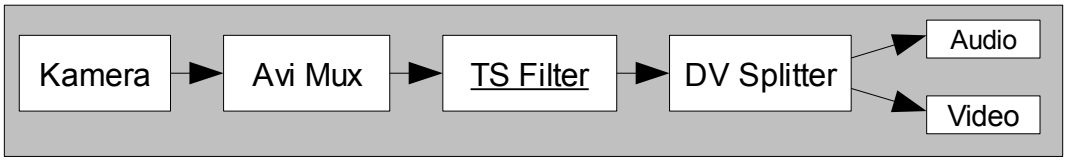

Abbildung 3: Filtergraph bei Verwendung nur eines Filters

Die Steuerung des Ausgabeteiles erfolgt ganz normal, wie gewohnt, über den Filtergraphen, die Steuerung des Eingabestromes, von der Kamera bis zum TS Filter, muss jedoch davon losgelöst sein. Der TS Filter muss dafür sorgen, dass Seek-Kommandos sowie die Start/Stop/Pause Kommandos nicht an den Aufnahmeteil des Filtergraphen weitergegeben werden. Er muss darüber hinaus ein Interface zur Verfügung stellen, welches es ermöglicht, den Aufnahmeteil zu steuern.

Wie man bereits erkennen kann, ist das Ganze nicht nur sehr unsauber, sondern beinhaltet auch einen erheblichen Mehraufwand. Des Weiteren ist es bei dieser Variante nicht möglich, mehrere unabhängige Videofenster anzuzeigen. Bei der Verwendung eines separaten Filtergraphen für die Wiedergabe ist dies jedoch, durch Nutzung mehrerer Render-Graphen, Problemlos machbar.

Folgernd aus diesen Einschränkungen, ist davon abzuraten, diese Variante als Lösung anzustreben.

5 Konzeptdetails

5.1 Synchronisationsobjekt

5.1.1 ISharedFile und seine Hilfsinterfaces

Für die Synchronisation und Kommunikation zwischen den zwei benötigten Filtern „TS Sink“ und „TS Source“, wird ein COM-Objekt eingesetzt, welches von beiden angesprochen werden kann. Im Grunde genommen sind es sogar drei COM-Objekte, die genutzt werden. Das Hauptaugenmerk liegt dabei auf dem Objekt TSSharedFile, welches das *ISharedFile*-Interface implementiert.

Über dieses Interface wird nicht nur der Dateiname bekannt gegeben, sondern auch der Zugang zu zwei Objekten hergestellt, die für das Lesen und Schreiben der Videodatei nützlich sind. Das Interface ist von IPropertyBag abgeleitet und bietet somit für zukünftige Zwecke eine Möglichkeit, weitere wichtige Informationen auszutauschen, die nicht in die Datei geschrieben werden. Auf diese Weise wäre es zum Beispiel möglich, auch ohne das ISharedDVInfo-Interface auszukommen.

Über die Funktion SetFileName wird der Dateiname gesetzt. Mit GetCurFile kann dieser wieder ausgelesen werden. Mit Hilfe der Funktion GetWriter, wird das Interface ISharedFileWriter eines weiteren COM-Objektes bereitgestellt. Dieses Objekt ist für den Schreibzugriff auf die Datei, welche mit SetFileName gesetzt wurde, zuständig. Von diesem Objekt kann es allerdings nur eine aktive Instanz geben. Da so immer nur ein Prozess das Schreibrecht für die Datei erhält, werden Kollisionen beim Beschreiben der Datei vermieden.

Das dritte COM-Objekt implementiert das Interface ISharedFileReader, welches über die Funktion GetReader erhalten werden kann. Von diesem Objekt kann es beliebig viele Instanzen geben. Daher ist es zum Beispiel möglich, mehrere Videofenster für die gleiche Datei anzuzeigen.

Die zwei Interfaces *ISharedFileWriter* und *ISharedFileReader* sind sich sehr ähnlich. Beide stellen die aus ISharedFile bekannte Funktion GetCurFile bereit, mit dessen Hilfe der Name der zu Grunde liegenden Datei abgerufen werden kann. Ebenfalls in beiden Interfaces vorhanden ist die Funktion SeekToPos, mit ihr kann der Dateizeiger gesetzt werden. Hierbei muss sich keine Sorgen um eventuelle Abhängigkeiten gemacht wer-

den, da jedes der beiden Objekte einen eigenen Dateihandler besitzen und die Dateizeiger damit unabhängig voneinander sind.

Darüber hinaus beinhaltet ISharedFileWriter die Funktion WriteData und äquivalent dazu ISharedFileReader die Funktion ReadData, die für das Schreiben bzw. Lesen von Daten in bzw. aus der Datei zuständig sind. Ihnen wird ein Pointer auf einen Pufferspeicher übergeben. Aus dem Pufferspeichern werden die Daten dann in die Datei geschrieben bzw. werden die Daten aus der Datei ausgelesen und in den Puffer kopiert.

5.1.2 ISharedDVInfo

Zusätzlich zu ISharedFile wird von TSSharedFile das Interface ISharedDVInfo implementiert. Da für die Arbeit der beiden Filter das Interface IPropertyBag zu unflexibel ist, um wichtige Informationen auszutauschen, wurde das Interface ISharedDVInfo definiert und implementiert.

Der Name des Interfaces leitet sich von der Datenstruktur ab, die mit Hilfe dieses Interfaces ausgetauscht werden kann, und zwar die DVINFO Struktur. Diese Struktur wird von der Kamera an den AVI-Mux Filter übergeben und erst mit dem Beenden der Aufnahme in den AVI-Header geschrieben. Da der Inhalt dieser Datenstruktur aber essentiell für die Wiedergabe der Videodaten ist, wird ihr Inhalt über ISharedDVInfo dem Sink Filter bekannt gegeben. Für diesen Zweck stellt das Interface die Funktionen GetDVInfo und SetDVInfo zur Verfügung.

Das ist allerdings nicht das einzige Einsatzgebiet dieses Interfaces. Damit der Source Filter korrekt arbeiten kann, werden weitere Daten benötigt. Dabei handelt es sich zum Einen um die Framelänge und -größe und zum Anderen muss dem Source Filter bekannt sein, wie viele Frames schon gespeichert wurden und an welcher Stelle sich diese in der Datei befinden.

Die Framelänge und -größe kann mit Hilfe der Funktion IsPal ermittelt werden. Da es sich bei diesen Werten um Standardwerte handelt, die sich aus der Fernsehnorm ergeben, sind sie davon abhängig, ob das Videobild in PAL oder NTSC Norm anliegt. Um die Nutzung allerdings zu vereinfachen, stellt das Interface ebenfalls die Funktionen GetFrameLength und GetSampleSize zur Verfügung, welche die entsprechenden Werte, je nach Fernsehnorm, ausgeben.

Der Sink Filter benutzt, neben der Funktion zum setzten der DVINFO Struktur, nur eine einzige Funktion des Interfaces, und zwar die Funktion SetSamplePos. Diese wird aufgerufen, wenn ein Frame in die AVI-Datei geschrieben wurde. Ihr wird die Dateiposition des Frames als Parameter übergeben. Diese Funktion löst ein Event aus, welches anzeigt, dass ein neues Sample verfügbar ist. Der Name dieses Events kann über die Funktion GetNewSampleEventName abgefragt werden.

Der Source Filter kann diese Informationen über eine Reihe von Funktionen abrufen. Zum Einen kann die Position des ersten Frames abgerufen werden, was besonders für den Wiedergabestart des Filters wichtig ist, da die Wiedergabe erst beginnen kann, wenn auch Daten vorhanden sind. Zum Anderen kann aber auch die Position des zuletzt geschriebenen Frames abgerufen werden, was für später Zwecke sinnvoll ist, da somit annähernd ein Livebild gezeigt werden kann. Die Namen der Funktionen entsprechen im Grunde dem, was sie zurückgeben, GetFirstSamplePos und GetActualSamplePos.

Mit Hilfe der Funktion GetSampleCout kann die Anzahl der bereits geschriebenen Frames ermittelt werden. So ist es möglich, in Verbindung mit der Framelänge, die momentane Gesamtlänge des Videos zu ermitteln.

Damit sich der Source Filter in dem schon aufgezeichneten Videomaterial bewegen kann, stellt das Interface die Funktionen GetPosForTime und GetPosForFrame zur Verfügung. Wird diesen Funktionen die gewünschte Zeit bzw. Framenummer übergeben, liefern sie die entsprechende Position des Frames in der Datei zurück. Darüber hinaus gibt die erste Funktion bei Bedarf auch die Framenummer mit aus.

5.2 Filter

5.2.1 Allgemein

Nachdem das Konzept für das Synchronisationsobjekt und seine Interfaces erarbeitet wurde, kann nun das Konzept für die zwei benötigten Filter definiert werden. Erster wichtiger Punkt hierbei, ist das Herstellen der Kommunikation zwischen den zwei Filtern. Die Kommunikation erfolgt, wie bereits erwähnt, über das Objekt TSSharedFile.

Um dieses Objekt beiden Filtern bekannt zu geben, muss es von einem der Filter erstellt werden und an den anderen Filter ein Pointer vom ISharedFile Interface übergeben werden. Da im ersten Schritt der Aufnahmegraph erzeugt wird, erfolgt die Erstellung einer Instanz von TSShardFile im Sink Filter. Der Filter gibt über sein QueryInterface das ISharedFile Interface dieses COM-Objektes aus.

Dem Source Filter muss nun ein Interface zur Verfügung gestellt werden, welches solch einen ISharedFile-Interfacepointer entgegennehmen kann. Zu diesem Zweck wird das Interface *ITSSourceFilter* eingeführt. Es bietet mit seiner Funktion SetSharedFile genau dies. Des Weiteren wird dieses Interface um die Funktion Live bereichert. Mit dieser wird es später möglich sein, zu dem (Beinahe-)Livebild zu springen.

Das Konzept für die beiden Filter wird in den nächsten zwei Kapiteln näher erläutert. Im Allgemeinen wird dabei die meiste Arbeit im Pin erledigt. DirectShow stellt eine Reihe von Basisklassen für die Erstellung eigener Filter, sowie diverse Hilfsklassen und Konstrukte zur Verfügung. Auf diese wird im 6. Kapitel näher eingegangen.

5.2.2 Sink-Filter

Der Sink Filter ist ein Render Filter. Er ist der letzte Filter im Datenfluss des Filtergraphen. Er nimmt die Daten des ihm vorgeschalteten AVI-Mux entgegen und speichert diese in einer Datei.

Der Filter

Für diese Art von Filter, ein Renderfilter der in eine Datei schreibt, bietet es sich an, das DirectShow Interface *IFileSinkFilter* zu implementieren. Dieses Interface stellt Funktionen für das Setzen bzw. Ausgeben des Namens der Zieldatei zur Verfügung. Bei diesen Funktionen handelt es sich um die beiden bereits aus ISharedFile bekannten SetFileName und GetCurFile. Besonderheit hierbei ist, dass der Input Pin erst erstellt werden kann, wenn ein Dateiname gesetzt wird. Näheres dazu im Abschnitt des Pins.

Wie bereits erwähnt, erstellt der Filter eine Instanz von TSSharedFile. Dies geschieht bereits mit dem Erstellen des Filters. Daher kann jederzeit über QueryInterface auf das ISharedFile Interfaces zugegriffen werden.

Der Input Pin

Mit dem Setzen des Dateinamens kann nun endlich auch der Input Pin erstellt werden. Damit er arbeiten kann, benötigt er allerdings Pointer auf das Interface ISharedFile und ISharedFileWriter. Den ersten bekommt er über seinen Filter und den zweite kann er sich über den ersten per GetWriter holen. Allerdings nur, wenn bereits ein Dateiname gesetzt wurde, daher kann auch jetzt erst der Pin erstellt werden.

Damit der Input Pin mit dem Output Pin des Filters AVI Mux verbunden werden kann, muss er den entsprechenden MediaType unterstützen. Dabei handelt es sich um MEDIATYPE_Stream und als Untertyp MEDIASUBTYPE_Avi.

Die eigentliche Arbeit des Pins erfolgt schließlich in der Funktion Receive. Ihr wird ein MediaSample übergeben, welches in die Datei geschrieben wird. Hier kommt nun das Interface ISharedDVInfo zum Einsatzt. Ihm wird nach jedem Schreibvorgang die Position des Samples/Frames in der Datei mitgeteilt.

Das Interface ISharedDVInfo kommt noch ein zweites Mal zum Einsatz, und zwar muss die DVINFO Strukur ermittelt und gespeichert werden. Da der Pin diese Strukur leider nicht mit übergeben bekommt, muss sie auf einem anderen Weg ermittelt werden.

Ermittlung der DVINFO Struktur

Jeder Filter eines Filtergraphen hat einen Pointer auf diesen. Über diesen Pointer kann jeder Filter im Graph angesprochen werden.

Da die DVINFO Strukur direkt von der Kamera ausgegeben wird, muss auf deren Filter im Graphen zugegriffen werden. Die Struktur findet sich in der MediaType-Struktur, mit der der Filter mit seinem Nachfolger verbunden ist.

Es müssen nun alle Pins des Kamera-Filters durchgegangen werden und überprüft werden, ob sie verbunden sind und ob es sich um Output Pins handelt. Ist dies der Fall, kann über die Funktion ConnectionMediaType des Interfaces IPin, der MediaType der Verbindung geholt werden.

Der richtige Zeitpunkt für das Ermitteln der DVINFO Struktur ist das Verbinden des Input Pins des Sink Filters mit dem Output Pin des AVI Mux. Nach einer erfolgreichen Verbindung wird die Funktion CompleteConnect ausgeführt. In ihr kann nun nach der Struktur gesucht werden.

IStream

Wird der Filtergraph gestoppt und die Aufnahme damit beendet, muss der Input Pin trotzdem noch arbeiten. Wie bereits erwähnt, muss dieser das Interface IStream bereitstellen, damit die AVI-Headerdaten nach dem Stoppen des Filtergraphen geschrieben werden können.

Dieses Interface wird von dem COM-Objekt TSSharedFileWriter implementiert. Das Objekt ist dem Filter über das ISharedFileWriter Interface bekannt. Die Anfrage nach dem IStream Interface kann somit direkt an dieses COM-Objekt weitergegeben werden.

5.2.3 Source-Filter

Der Filter

Der Filter implementiert, wie bereits erwähnt, das Interface ITSSourceFilter. Es handelt sich hierbei um einen Filter vom Typ Source Filter, d.h. von ihm geht der Datenfluss im Filtergraphen aus. Er besitzt einen Output Pin über den die Daten ausgegeben werden.

Der Output Pin des Filters wird, ähnlich wie bei dem Sink Filter, erst erzeugt, wenn mittels der ITSSourceFilter-Funktion SetSharedFile der Pointer auf das ISharedFile Interface gesetzt wird. Wenn über dieses Interface ein Reader (ISharedFileReader) geholt werden kann, wird der Pin erzeugt.

Die Funktion Live des Interfaces ITSSourceFilter sorgt für einen Seek am Output Pin auf das aktuelle Livebild. Dabei handelt es sich um das letzte geschrieben Frame in der AVI-Datei.

Der Output Pin

Der Output Pin des Sink Filters übernimmt also die Hauptarbeit. Er ist nicht nur für das Senden der Daten verantwortlich, sondern er empfängt auch das Seek-Kommando. Er muss also dafür sorgen, dass sich in den bereits aufgezeichneten Daten vor und zurück bewegt werden kann.

Doch erst einmal muss eine Verbindung zum Filter „DV Splitter“ hergestellt werden. Damit dies geschehen kann, muss der Filter die MediaType Struktur der Daten, die er liefert, ausgeben. In dieser Struktur ist als Haupttyp MEDIATYPE_Interleaved, also Interleaved DV Daten, vermerkt. Als Untertyp wird MEDIASUBTYPE_dvsd angegeben. Das ist zur Zeit der einzige Untertyp der direkt von DirectShow und dem Filter „DV Splitter“ unterstützt wird. Damit der Splitter Filter richtig arbeiten kann, benötigt er, zusätzlich zu diesen Angaben, die DVINFO Struktur. Aus dieser Struktur kann er herauslesen, wie die Daten in Video und Audio aufgeteilt werden müssen.

Ebenfalls Wichtig für den Aufbau der Verbindung zum Splitter Filter ist die Funktion DecideBufferSize. Mit ihrer Hilfe wird die Größe des Zwischenspeichers für die MediaSamples festgelegt. Hier ist nun die Größe eines Frames entscheidend. Diese kann mittels GetSampleSize über das ISharedDVInfo Interface abgefragt werden.

Ähnlich wie die Receive Funktion des Sink Filter Input Pins, besitzt der Output Pin eine Funktion, die für die Übertragung der Daten zuständig ist. Dabei handelt es sich um die Funktion FillBuffer. Sie füllt einen Zwischenspeicher mit den Daten eines Frames aus der AVI-Datei und übergibt den entsprechenden TimeStamp für diesen.

Beim Lesen der Daten aus der AVI-Datei ist darauf zu achten, dass an der angegebenen Stelle der AVI-Chunk-Header gefolgt vom eigentlichen Frame steht. Es muss also erst der Chunk-Header ausgelesen werden, ehe die Framedaten geladen werden können.

Wenn keine Daten mehr zum lesen bereit sind, wird auf das Event „NewSample“ gewartet. Dies ist zum Beispiel der Fall, wenn die Wiedergabe schneller läuft als die Aufnahme oder aber die Aufnahme beendet wurde. Wird das Event nicht innerhalb einer Sekunde gesetzt, wird die Wiedergabe gestoppt.

Seeking und Wiedergabegeschwindigkeit

Damit sich in den bereits aufgezeichneten Daten vor und zurück bewegt werden kann, muss der Pin die Funktionen für das Seeking implementieren. Wichtig hierbei ist unter anderem die Funktion OnThreadStartPlay, da sie immer aufgerufen wird, wenn die Wiedergabe gestartet wird und wenn der Pin einen Seek ausgeführt hat. In dieser Funktion muss die angegebene Startzeit umgewandelt werden in die entsprechende Framenummer und somit der Dateizeiger auf den entsprechenden Frame gesetzt werden.

Aber nicht nur Seeking ist möglich, die Geschwindigkeit der Wiedergabe lässt sich ebenfalls ändern. Mit Hilfe der Funktion SetRate kann diese gesetzt werden. Dabei ist allerdings Vorsicht geboten, wenn die Wiedergabegeschwindigkeit größer als eins ist. In diesem Fall, kann es zu Stockungen kommen, da die Daten mit einfacher Geschwindigkeit geschrieben werden.

5.3 Testanwendung

Um das System der beiden Filter zu testen, wird ein Testprogramm benötigt. DirectShow bietet mit dem Programm GraphEdit zwar eine Möglichkeit, Filtergraphen zu erstellen und zu testen, da aber mehrere Filtergraphen erstellt werden müssen und zwischen den Filtern das Objekt TSSharedFile ausgetauscht werden muss, kann dieses Programm nicht mehr genutzt werden.

Einen Zweck erfüllt es dennoch. Im Testprogramm werden die beiden erzeugten Filtergraphen in die Running Object Table des Systems eingetragen. Mit dem Programm GraphEdit ist es nun möglich zu so einem Remotegraphen eine Verbindung aufzubauen. Auf diese Weise ist es möglich die erzeugten Graphen zu analysieren.

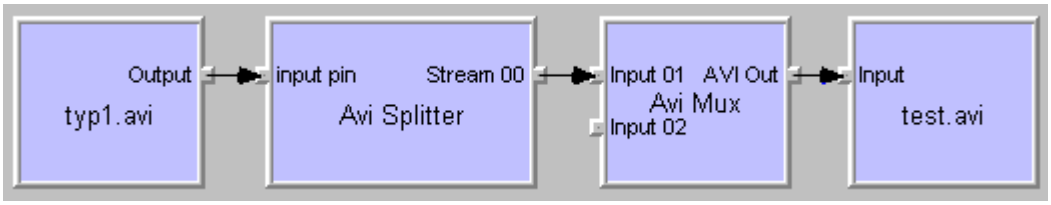

Abbildung 4: Sink-Graph mit Datei als Quelle

Damit zum Testen des Systems nicht immer eine Kamera an den Rechner angeschlossen sein muss, bietet das Testprogramm eine Auswahl für die Datenquelle. Zum einen kann eine angeschlossene Kamera ausgewählt werden und zum anderen eine AVI-Datei als Quelle für den Sink Graphen.

Damit die Auswahl einer AVI-Datei als Quelle möglich ist, muss der Input Pin des Sink Filters erweitert werden. Da nun kein Filter für die Kamera mehr vorhanden ist, muss die DVINFO Struktur am Output Pin des Filters „AVI Splitter“ abgegriffen werden.

Neben der Auswahl der Quelle muss natürlich auch angegeben werden, in welcher Datei die Daten gespeichert werden sollen.

Das Testprogramm ist so aufgebaut, das es komplett über das Menü gesteuert wird. Hierbei wird für einige Menüpunkte ein ShortCut vergeben, damit alternativ eine Steuerung über die Tastatur möglich ist. Im Clientbereich des Programmes wird während der Wiedergabe das Video angezeigt.

Das Menü beinhaltet hierbei zwei Untermenüs, eines für den Aufnahmegraphen (Sink) und eines für den Wiedergabegraphen (Source). Das Menü „Aufnahme“ enthält die zwei Punkte Start und Stopp, mit ihnen kann die Aufnahme der Daten gestartet und gestoppt werden.

Das Menü „Wiedergabe“ stellt diese zwei Punkte ebenfalls zur Verfügung, allerdings wird mit ihnen die Wiedergabe gestartet und gestoppt. Zusätzlich dazu enthält das Wiedergabemenü den Punkt Pause, für das Pausieren der Wiedergabe und zwei Punkte für das Vor- und Zurückspringen im Video. Ebenfalls enthalten sind zwei Punkte für das Erhöhen bzw. Verringern der Wiedergabegeschwindigkeit. Zusätzlich dazu enthält das Menü einen Punkt „Live“, mit ihm wird die Live-Funktion des Source Filters aufgerufen und das aktuelle Livebild angezeigt.

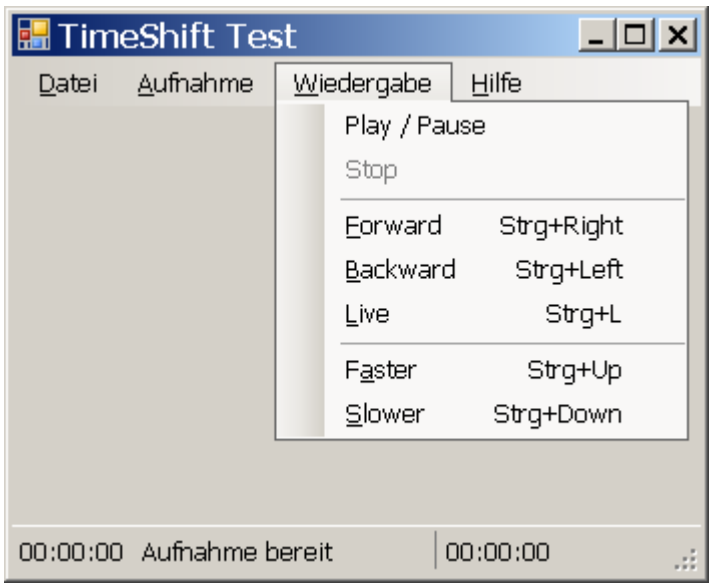

Abbildung 5: TSTest - Menü Wiedergabe

6 Implementierung

6.1 Allgemeine Vorbemerkungen zur Implementierung

6.1.1 Entwicklungsumgebung

Die Implementierung des Konzeptes erfolgt hauptsächlich in der Programmiersprache C++. Für die Definition der Interfaces wird zum Teil MIDL genutzt. Das Testprogramm wird in C# geschrieben und verwendet .Net 2.0. Dies dient zum Einen Demonstrationszwecken, also zum Beweis für die Interoperabilität von COM, und zum Anderen dazu den Programmieraufwand für die GUI möglichst einfach und kurz zu halten.

Das Konzept wird mithilfe der IDE Microsoft Visual Studio 2005 umgesetzt. Für Testzwecke wird das im Plattform SDK enthaltene Programm „GraphEdit“ benutzt. Für die Analyse von AVI-Dateien kommen die zwei Programme „DVDate 6.4.4“ von Paul Glagla[16] und „FileAlyzer 1.5.5.0“ von Safer Networking Limited[17] zum Einsatz.

Als erstes wird das Objekt für die Synchronisation der Filter implementiert, da die Filter erst erstellt werden können, wenn die Interfacedefinition von ISharedFile und Co vorhanden ist. Danach werden die zwei Filter mit ihren Pins implementiert. Als letztes folgt die Entwicklung einer Testumgebung für das System.

Für die Entwicklung der Filter wird das aktuelle Mircosoft Plattform SDK benötigt. In diesem sind die Headerdateien von DirectShow, sowie eine Sammlung von Hilfsklassen für das Erstellen von Filtern, enthalten.

Die Implementierung wird an ausgewählten Beispielen demonstriert. Bei den Beispielen handelt es sich meist um besonders kritische oder schwierige Stellen im Code. Der vollständige Quellcode, sowie die entsprechenden Binaries sind auf der beiliegenden CD enthalten.

16 http://paul.glagla.free.fr/dvdate_en.htm
17 http://www.safer-networking.org/de/filealyzer/

6.1.2 Verhaltensmuster für den Quellcode

Um die Lesbarkeit des Quellcodes zu gewährleisten, wird der Code im *Allman-Stiel*[18] formatiert, d.h. es wird eine Einrückung von 4 Leerzeichen genutzt und die öffnende geschweifte Klammer befindet sich auf einer neuen Zeile.

Um mögliche Fehler in Kontrollstrukturen zu vermeiden, wird die *Left-Hand Methode* genutzt, d.h. dass Konstanten auf der linken Seite eines Vergleiches stehen. Dadurch wird sichergestellt, dass es nicht zu ungewollten Zuordnungen kommt. Da einer Konstanten kein Wert zugewiesen werden kann, bringt der Compiler einen Fehler. Bei der Right-Hand Methode kommt es oft zu schwer auffindbaren Fehlern.

```
int f(int x, int y, int z)
{
    if (foo(y, z) > x )
    {
        haha = bar[4] + 5;
    }
    else
    {
        while (z)
        {
            haha += foo(z, z);
            z--;
        }
        return ++x + bar();
    }
}
```

Damit im Programm die Entstehung von Speicherlöchern vermieden wird, kommt das Hilfsobjekt *CComPtr* zum Einsatz. Dabei handelt es sich um eine SmartPointer-Implementierung für COM-Pointer. Das Objekt hält den eigentlichen Pointer und sorgt für die nötigen Aufrufe von AddRef und Release. Wird das Objekt entladen, wird zum Beispiel automatisch die Release-Methode aufgerufen.

Damit im Quellcode schnell zu erkennen ist, um was für eine Variable es sich handelt, orientiert sich die Namenskonvention für diese an der *ungarischen Notation*[19]. Für Membervariablen der Klasse wird als Präfix „m_“ genutzt, Konstanten werden komplett groß geschrieben und wenn es sich bei der Variable um einen Pointer handelt, wird vor den Namen ein kleines „p“ gesetzt. Gleiches gilt für Pointer auf Pointer, so lautet der Variablenname mit dem Typ int** zum Beispiel „ppVar“.

18 siehe http://en.wikipedia.org/wiki/Indent_style#Allman_style_.28bsd_in_Emacs.29
19 siehe http://de.wikipedia.org/wiki/Ungarische_Notation

6.2 Synchronisationsobjekt

6.2.1 ISharedFile

Um die Arbeit der Filter zu vereinfachen und die Kommunikation sicher zu stellen, werden drei Objekte benötigt. Das Hauptobjekt TSSharedFile ist Ausgangspunkt für die beiden anderen Objekte TSSharedFileReader und TSSharedFileWriter, daher stellt auch nur dieses die Funktion CreateInstance zur Verfügung.

Um an die anderen beiden Objekte zu gelangen werden die Funktionen GetWriter und GetReader genutzt. Hierbei ist besonders auf den Referenzcount der drei Objekte zu achten, da die ReadWrite Objekte einen Pointer auf das Hauptobjekt haben.

Referenzcount

Wenn in der Funktion GetWriter ein neues TSSharedFileWriter-Objekt mittels new erzeugt wird, muss der Referenzcount dieses per AddRef erhöht werden. Ebenso muss der Referenzcount von TSSharedFile erhöht werden, da ja das eben erzeugte Objekt einen Pointer auf diese hat. Gleiches gilt für GetReader.

Wird das TSSharedFileWriter Objekt wieder freigegeben und sein Referenzcount ist 0, wird das Objekt zerstört. In diesem Moment wird auch der Pointer auf TSSharedFile freigegeben. Sollte der Writer als einziges Objekt noch einen Pointer auf dieses gehabt haben, wird der Referenzcount von TSSharedFile ebenfalls 0 betragen und somit dieses auch zerstört werden.

IPropertyBag

Da das Interface ISharedFile von IPropertyBag abgeleitet ist, muss es dieses ebenfalls implementieren. Dabei wird über die Funktionen Read und Write ein Schlüssel-Wert-Paar gelesen bzw. gespeichert. Für die Speicherung dieser Paare wird die Klasse CSimpleMap genutzt. Als Schlüssel kommt dabei eine Zeichenkette zum Einsatz. Der Wert wird mit dem Typ VARIANT gespeichert. Somit ist es möglich eine Vielzahl an unterschiedlichen Informationen und Datentypen zu speichern.

Beim Einsatz dieser Speicherung kommen wieder zwei Hilfsklassen aus COM zum Tragen. Zum einen CComBSTR, mit dessen Hilfe die Zeichenkette gespeichert wird, und zum anderen CComVariant, welches die Arbeit mit dem VARIANT-Typen vereinfacht und sich um die nötigen Kopiervorgänge kümmert.

6.2.2 ISharedDVInfo

Das ISharedDVInfo Interface ist das umfangreichste Interface von TSSharedFile. Bei der Implementierung dieses Interface kommt es vor allem auf eine konsistente Datenhaltung an.

Da zwei oder mehr Threads gleichzeitig auf dieses Interface zugreifen und dabei Daten bzw. Variablen verändern, müssen diese Zugriffe synchronisiert werden. Schreiben zwei Threads gleichzeitig in eine Variable, ist der Inhalt dieser inkonsistent, da hier der pure Zufall entscheidet, was die Variable letztendlich enthält.

Um kritische Abschnitte bezüglich ISharedDVInfo abzusichern, kommt wieder eine Hilfsklasse von COM zum Einsatz. Die Klasse *CCritSec* stellt einen Mutex dar, welcher mit Lock gesperrt und mit Unlock wieder freigegeben werden kann.

TSSharedFile enthält eine Membervariable vom Typ dieser Klasse. Immer wenn eine Funktion lesend oder schreibend auf die Variablen die ISharedDVInfo betreffen zugreifen, wird das Objekt gelockt. Wenn gleichzeitig ein weiterer Funktionsaufruf versucht, auf eine Variable zuzugreifen, warte dieser, bis der Mutex wieder freigegeben wird.

```
int foo()
{
    // do something
    ...
    {
        CAutoLock lock(m_mutex);
        // do something special
        ...
    }
    // do something
    ...
}
```

Damit es bei der Verwendung des Mutex nicht zu Deadlocks kommt, weil zum Beispiel vergessen wurde, diesen wieder freizugeben, kommt die Hilfsklasse *CAutoLock* zum Einsatz. Dieser wird beim Erzeugen einer Instanz, der Mutex übergeben und automatisch gelockt. Wird die Instanz zerstört, weil zum Beispiel die Sichtbarkeit (der Scope) verlassen wird, wird der Mutex wieder freigegeben.

Wenn man zum Beispiel den Zugriff auf eine gefährdete Variable nochmal in geschweifte Klammern schreibt und im Anfang dieses Blockes ein CAutoLock Objekt erzeugt, ist der Mutex solange gesperrt, bis der Block wieder verlassen wird.

Eine der Variablen, die so geschützt werden muss, ist die Liste mit den Pointern zu den Frames in der AVI-Datei. Beim Aufruf von SetSamplePos aus dem Sink Filter heraus, wird in diese Liste geschrieben, während gleichzeitig zum Beispiel der Source Filter GetSampleCount aufruft und damit die Anzahl an Pointern in der Liste abfragt.

6.3 Filter

6.3.1 Sink-Filter

Bei der Implementierung des Sink Filters ist vor allem sein Pin wichtig, da dieser die Daten empfängt und speichert. Außerdem ist der Pin dafür verantwortlich, dass in TSSharedFile die DVINFO Struktur des Videosignals gespeichert wird.

Für die Beschaffung dieser Struktur kommen zwei Hilfsfunktionen zum Einsatz. Als erstes wird die Funktion FindSourceFilter aufgerufen. Diese Funktion sucht in dem Graphen, in dem sich der Sink Filter befindet, den Filter mit der DVINFO Struktur. Dabei handelt es sich im Normalfall um den Filter der Kamera. Wird dieser nicht gefunden, wird nach einem „AVI Splitter“ gesucht. Dieser Fall wurde mit implementiert, damit man das ganze System auch ohne Kamera testen kann.

Filter finden

Um den entsprechenden Filter zu finden, wird ein Pointer auf den Graphen benötigt. Dieser wird am Filter gesetzt, sobald er einem Graphen hinzugefügt wird. Der Graph stellt eine Funktion EnumFilters bereit, mit ihr können alle Filter im Graphen durchgegangen werden.

Um die Filter zu identifizieren, werden zwei Methoden genutzt. Da IBaseFilter, das Interface, welches alle DirectShow Filter implementieren, von IPersist abgeleitet ist, stellt dieses eine Funktion GetClassID zur Verfügung. Damit lässt sich die CLSID des Filters ausgeben. Stimmt diese mit der CLSID des „AVI Splitter“ Filters überein, ist der Filter eindeutig identifiziert wurden.

Für den Filter der Kamera muss allerdings anders vorgegangen werden, da für diesen keine CLSID vorhanden ist. Um den Filter dennoch zu finden, wird auf das Interface IAMExtDevice getestet. Implementiert der Filter dieses Interface, handelt es sich wahrscheinlich um den Filter der Kamera. Es könnte allerdings auch der Filter eines Videorekorders sein. In diesem Fall würde aber sowieso das ganze System nicht funktionieren, deshalb kann diese Ausnahme unbeachtet bleiben.

Wenn die Funktion einen Kamera-Filter findet, wird dieser zurückgegeben. Wurde keiner gefunden, wird wenn möglich der „AVI Splitter“ Filter zurückgegeben. Wenn dieser ebenfalls nicht gefunden wurde, ist die Rückgabe der Funktion E_FAIL.

MediaType der Verbindung

Wurde ein passender Filter gefunden, kann die Funktion GetConnectionMediaType verwendet werden, um den MediaType der ausgehenden Verbindung des Filters zu ermitteln. In dieser sollte dann die DVINFO Struktur enthalten sein.

Um den MediaType der Verbindung zu erhalten, werden alle Pins des Filters dahingehend überprüft, ob es sich um Output Pins handelt. Ist dies der Fall, wird mit Hilfe der Funktion ConnectionMediaType, welche von IPin bereitgestellt wird, der MediaType der Verbindung abgefragt und ausgegeben.

Schreiben der Daten

Die Hauptaufgabe des Pins besteht allerdings darin, die empfangenen Daten in die AVI-Datei zu schreiben. Dies geschieht in der Funktion Receive.

Der Funktion wird ein MediaSample übergeben, welches an eine bestimmte Stelle geschrieben werden soll. Die genaue Position an der dies erfolgen soll, steht in der Startzeit des MediaSamples. Bei dieser handelt es sich nicht wie sonst, um die Angabe einer Referenzzeit, sondern um die Adresse in der AVI-Datei.

Der Pin erledigt das Schreiben dieser Daten über das Objekt TSSharedFileWriter oder besser gesagt über das Interface ISharedFileWriter. Über SeekToPos wird an die angegebene Stelle gesprungen, an der nun mittels WriteData der Zwischenspeicher aus dem MediaSample geschrieben wird.

Ist das Schreiben erfolgreich verlaufen, wird die Position des eben geschriebenen Frames, in TSSharedFile, per SetSamplePos, gespeichert.

6.3.2 Source-Filter

Der Source Filter ist, wie auch der Sink Filter, eigentlich nur dafür da, um seinen Pin zu erzeugen. Im Output Pin dieses Filters wird die ganze Arbeit erledigt. Damit der Pin erzeugt wird, muss dem Filter ein Pointer auf ein ISharedFile Interface übergeben werden. Da der Filter dieses nicht benötigt, gibt er es, ohne eine Referenz darauf zu behalten, direkt an den Pin weiter.

Der Pin holt sich nun über das Interface ein TSSharedFileReader Objekt, mit dem er die Daten aus der AVI-Datei lesen kann. Zusätzlich dazu beschafft er sich noch das ISharedDVInfo Interface.

Mit dem Start der Wiedergabe wird die Funktion OnThreadStartPlay aufgerufen. In ihr muss wird zu der angegeben Startzeit die Frameposition und -nummer geholt. Schlägt dies fehl, weil zum Beispiel noch keine Daten vom Sink Filter geschrieben wurden, wird der Thread in den Wartemodus versetzt. Es wird solange gewartet, bis das NewSample-Event eintritt. Tritt dieses nicht innerhalb von fünf Sekunden ein, wird der Thread beendet.

Lesen der Daten

Wurde eine Position gefunden, nimmt der Pin seine Arbeit auf und liefert über FillBuffer seine Daten. Dieser Funktion wird ein MediaSample übergeben, in welches die Daten des aktuellen Frames geschrieben werden. Zusätzlich dazu wird im MediaSample noch die Start- und Stoppzeit des Frames angegeben.

Ehe dies allerdings geschieht, wird bei jedem Aufruf der Funktion überprüft, ob Daten für die aktuelle Position vorhanden sind. Da die beiden Threads für das Schreiben und Lesen nicht synchron laufen, kann es vorkommen, dass der Pin versucht Daten zu lesen, die noch nicht geschrieben wurden. Wenn festgestellt wird, dass noch keine Daten vorhanden sind, wird der Thread in den Wartemodus geschalltet. Wenn das Event „NewSample“ eintritt beginnt der Thread wieder zu arbeiten. Sind nach einer Sekunde immer noch keine Daten da, wird abgebrochen und die Funktion liefert S_FALSE zurück. Dies sorgt dafür, dass ein EndOfStream gesendet wird.

Um die Framedaten aus der AVI-Datei zu laden, müssen als erstes 8 Byte gelesen werden. Hierbei handelt es sich um die Chunk-Header Daten. Sind diese gelesen, befindet sich der Dateizeiger an der richtigen Position. Nun können die eigentlich Daten, mittels ReadData, gelesen werden.

Live-Bild

Um das aktuelle Livebild anzuzeigen, implementiert der Filter die Funktion Live. Die Funktion führt einen Seek-Aufruf auf das Livebild aus. Für diesen Zweck wird das IMediaSeeking Interface des Filtergraphen benötigt. Über die Funktion LivePos am Output Pin erhält der Filter die Zeit des letzten Frames in der AVI-Datei. Diese Zeit wird beim Aufruf von SetPosition übergeben und sorgt dafür, dass die Wiedergabe abbricht und an dieser Stelle neu beginnt.

6.4 Testanwendung

Das Testprogramm für die zwei Filter nutzt die freie Bibliothek DirectShowNET. Diese stellt eine Art Wrapper für DirectShow, für die Verwendung in managed Code, dar.[20]

Um die Aufnahme und die Wiedergabe besser steuern zu können, werden die dafür nötigen Funktion jeweils in eine Klasse zusammengefasst. Die Aufnahme wird durch die Klasse TSRecord und die Wiedergabe durch TSPlay gekapselt.

TSRecord

TSRecord erzeugt den Filtergraphen für die Aufnahme entsprechend der Quell-Auswahl. Wenn eine Datei als Quelle gewählt ist, muss ein anderer Filtergraph erzeugt werden, als bei einer Kamera als Quelle.

Bei einer Kamera als Quelle kommt zusätzlich zu dem GrapheBuilder der CaptureGraphBuilder zum Einsatz. Wird diesem eine Referenz zur Kamera übergeben, erstellt er automatisch den benötigten Filter für diese.

Bei einer Datei als Quelle muss es sich um eine AVI-Datei handeln. Daher beinhaltet der Filtergraph in diesem Fall die Filter „File Source“ und „AVI Splitter“.

Über die Funktion GetISharedFile gibt das Objekt das Interface ISharedFile des Sink Filters bekannt.

Die Aufnahme kann über die beiden Funktionen Start und Stop gesteuert werden.

TSPlay

TSPlay wird beim Erzeugen ein Verweis auf TSRecord übergeben, damit der Filtergraph für die Wiedergabe erzeugt werden kann. Es wird der Source Filter erzeugt und in den Filter eingefügt.

Vom Source Filter wird das Interface ITSSourceFilter abgerufen. Diesem wird das ISharedFile Interface aus TSRecord übergeben. Für die Erzeugung der restlichen Filter wird die Funktion Render des Filtergraphen genutzt. Dieser Funktion wird der Output Pin des Source Filters übergeben.

Neben den Funktionen für die Steuerung der Wiedergabe, also Start, Stop, Forward, Backward, stellt die Klasse auch Funktionen für das Steuern der Wiedergabegeschwin-

20 nach: DirectShowNet Library [DS_5]

digkeit zur Verfügung. Dabei werden die Aufrufe dieser Funktionen immer an den Filtergraphen, über das Interface IMediaSeeking, weitergeleitet.

Der Aufruf der Funktion Live wird direkt an den Source Filter durchgegeben. Dieser sorgt dafür, dass das Live-Bild angezeigt wird.

Die Oberfläche

Die Oberfläche des Programms bietet mit seinen Menüs die Möglichkeit, die Funktionen von TSRecord und TSPlay aufzurufen. Diese Menüs werden erst freigegeben, wenn die beiden Instanzen erzeugt wurden.

Damit nicht ständig das Menü benutzt werden muss, sind für die wichtigsten Funktionen Tastenkombinationen vergeben.

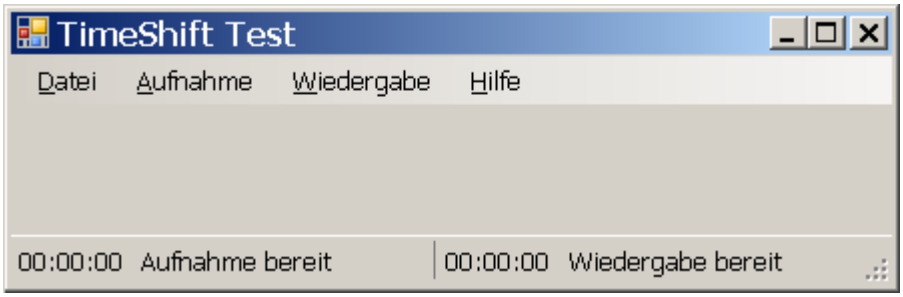

Abbildung 6: Oberfläche des Testprogramms

7 Test und Optimierung

7.1 Synchronisation

Zum Testen der Funktionen des Systems wurde ein Testprogramm entwickelt. Um den internen Programmablauf verfolgen zu können, ohne in die Ausführung des Programmes hinein zu debuggen, schreiben die Filter in eine Log-Datei. Dies geschieht nur wenn die Filter im Debug-Modus erstellt wurden.

Jede wichtige Funktion in den Filtern schreibt eine Zeile mit ihrem Namen und wichtigen Variablenwerten in die Log-Datei. Der Ort dieser Datei kann an der Registry angegeben werden.[21]

In dieser Log-Datei lässt sich sehr gut erkennen, wie die Schreib und Lesethreads arbeiten. Es wird abwechselnd geschrieben und gelesen. Leider ist dies nicht die ganze Zeit so. Es kann vorkommen, dass mehrmals hintereinander geschrieben und dann mehrmals gelesen wird.

Damit es dabei nicht zu einem ungewollten Programmabbruch kommt, ist dafür gesorgt wurden, dass nur gelesen wird, wenn auch Daten vorhanden sind (siehe 6.3.2). Dies ist vor allem wichtig, wenn die Wiedergabeposition sehr nah an der Aufnahmeposition ist, also wenn zum Beispiel die Funktion Live ausgeführt wurde.

Ein Auszug aus solch einer Log-Datei ist im Anhang zu finden.

21 HKEY_LOCAL_MACHINE\SOFTWARE\Debug\CCC-TS-Filter.dll\LogToFile

7.2 Test der erzeugten AVI-Datei

Ein weiteres Muss-Kriterium des Systems war, dass die erzeugte Datei auch ohne Einsatz dieser Filter eingesetzt werden kann. Um dies zu überprüfen, genügt es, die erzeugte Datei in einem beliebigen Player wiederzugeben.

Wenn man die Datei zum Beispiel mit dem Programm „FileAlyzer“ öffnet, kann man sehr gut den Aufbau der Datei erkennen.

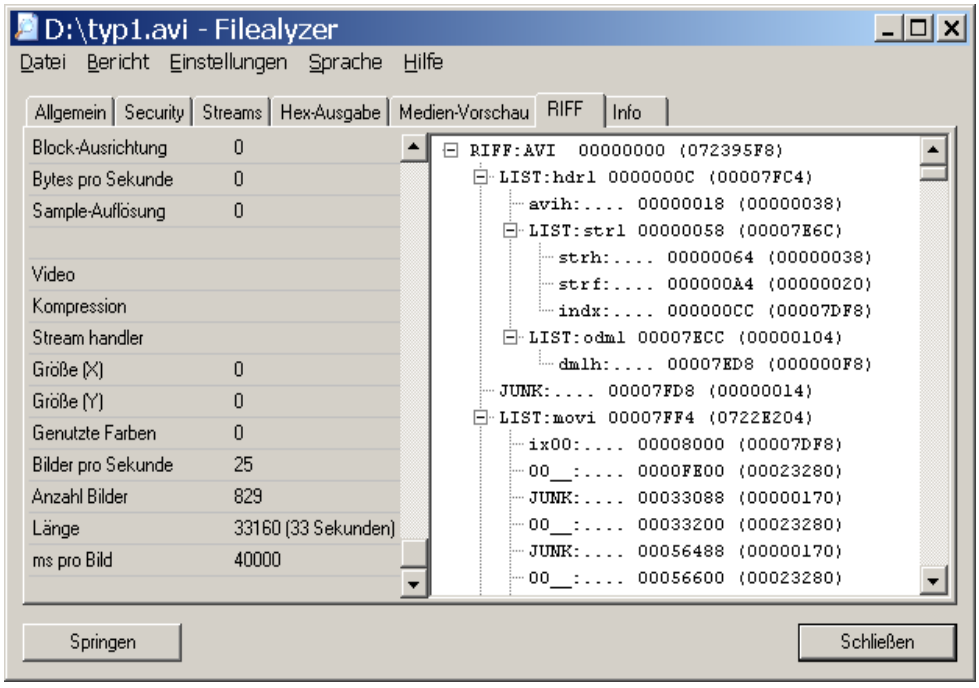

Abbildung 7: AVI-Datei in FileAlyzer

Während die Aufnahme läuft, werden nur die Daten in „LIST:movi“ geschrieben. Wird die Aufnahme beendet, werden über das IStream Interface des Input Pins des Sink Filters die Headerdaten der Datei geschrieben.

Als Beweis, dass die erzeugte Datei ohne Probleme weitergegeben werden kann, wurde sie auf mehreren Systemen erfolgreich getestet. Bei den Systemen handelt es sich um ein „Windows XP SP2“ ohne das entwickelte Filtersystem, ein „Ubuntu 7.10“ mit dem Programm „Totem“ (gstreamer), sowie ein „Suse 10.2“ mit „Kaffeine“ (xine).

8 Ergebnis und Ausblick

Es ist ein bereits funktionierendes System entwickelt wurden, welches es ermöglicht Videodaten aufzuzeichnen und gleichzeitig Wiederzugeben. Während der Wiedergabe kann sich in den bereits aufgezeichneten Daten bewegt werden, sowie die Wiedergabe Pausiert oder ihre Geschwindigkeit geändert werden. In Abschließenden Tests ist bewiesen wurden, das die Erzeugten Videodateien weitergeben werden können. Das Ziel dieser Diplomarbeit ist damit erreicht.

Als nächster Schritt folgt der Test des Filtersystems auf anderen Rechner als dem Entwicklungsrechner. Da es sich bei dem entwickelten System um eine Sammlung von COM-Objekten handelt sollte dies ohne Probleme möglich sein.

Wichtig bei dem Test der Filter ist die Zusammenarbeit mit anderen Filtern. Dies trifft vor allem auf den Wiedergabe-Graphen zu, da hier eine Vielzahl an Transform Filtern zum Einsatz kommen kann.

Sind diese Tests erfolgreich verlaufen, steht als letzter Punkt die Einbindung des Systems in die Software utilius VS. Daraufhin ist dann das in der Einleitung erläuterte Auswerten von Sportereignissen zeitgleich, während ihrer Aufnahme, möglich.

Abkürzungsverzeichnis

ASF	Advanced Streaming Format
AVI	Audio Video Interleave
CLSID	Class Identifier
COM	Component Object Model
DCT	Diskrete Kosinustransformation
DRM	Digital Rights Management
DV	Digital Video
DVB	Digital Video Broadcasting
EBML	Extensible Binary Meta Language
FOURCC	Four-Character Code
GUID	Globally Unique Identifier
IDL	interface description language
IEEE	Institute of Electrical and Electronics Engineers
IID	Interface Identifier
MP3	MPEG-1 Audio Layer 3
MP4	MPEG-4 Part 14
RIFF	Resource Interchange File Format
RPC	Remote Procedure Call
SBE	Stream Buffer Engine
SDK	Software Development Kit
TS	TimeShift
WDM	Windows Driver Model
WMV	Windows Media Video
XML	Extensible Markup Language

Literaturverzeichnis

WIKI_1 Wikipedia: DV; 30.05.2008 18:41;
http://en.wikipedia.org/wiki/DV

WIKI_2 Wikipedia: DirectShow; 10.05.2008 10:29;
http://en.wikipedia.org/wiki/DirectShow

WIKI_3 Wikipedia: MPEG-1 – Videokodierungsverfahren; 14.05.2008 08:26;
http://de.wikipedia.org/wiki/MPEG-1#Videokodierungsverfahren

WIKI_4 Wikipedia: QuickTime – Containerformat; 23.05.2008 13:15;
http://de.wikipedia.org/wiki/QuickTime#Containerformat

WIKI_5 Wikipedia: MPEG-4 Part 14; 15.05.2008 02:02
http://en.wikipedia.org/wiki/MPEG-4_Part_14

WIKI_6 Wikipedia: Matroska; 26.05.2008 23:46;
http://en.wikipedia.org/wiki/Matroska

DS_1 MSDN: Introduction to DirectShow; 10.05.2008
http://msdn.microsoft.com/en-us/library/ms786508(VS.85).aspx

DS_2 Pesce, Mark D.; Programming Microsoft DirectShow for Digital Video and Television; Microsoft Press; 2003

DS_3 MSDN: About DirectShow Filters; 10.05.2008;
http://msdn.microsoft.com/en-us/library/ms778825(VS.85).aspx

DS_4 MSDN: Using the Stream Buffer Engine; 12.05.2008;
http://msdn.microsoft.com/en-us/library/ms787869(VS.85).aspx

DS_5 DirectShowNet library; 20.05.2008;
http://directshownet.sourceforge.net/about.html

COM_1 MSDN: The Component Object Model: A Technical Overview; 01.10.1994;
http://msdn.microsoft.com/en-us/library/ms809980.aspx

COM_2 MSDN: Microsoft Interface Definition Language; 01.11.2007;
http://msdn.microsoft.com/en-us/library/aa367091.aspx

Abbildungsverzeichnis

Anlagen

IDL für ISharedFile

```
[object, uuid("CD535286-B45D-4e2e-8B7D-75EF49F44F11")]
interface ISharedFileWriter : IUnknown
{
        HRESULT SeekToPos ([in] LONGLONG llPos);
        HRESULT WriteData ([in] byte const* pv, [in] ULONG cb);
        HRESULT GetCurFile ([out, retval, string] LPOLESTR* ppszFileName);
};


[object, uuid("90F2409D-F59F-4474-BA77-8F9A6168C3A1")]
interface ISharedFileReader : IUnknown
{
        HRESULT SeekToPos ([in] LONGLONG llPos);
        HRESULT ReadData ([out] byte* pv, [in] ULONG cb);
        HRESULT GetCurFile ([out, retval, string] LPOLESTR* ppszFileName);
};

[object, uuid("E1FC34BB-D54E-47ac-9DB1-5BD481500556")]
interface ISharedFile : IPropertyBag
{
        HRESULT GetWriter ([out] ISharedFileWriter** ppWriter, [in] LPUNKNOWN pUnk);
        HRESULT GetReader ([out] ISharedFileReader** ppWriter, [in] LPUNKNOWN pUnk);
        HRESULT GetCurFile ([out, retval, string] LPOLESTR* ppszFileName);
        HRESULT SetFileName ([in, string] LPCOLESTR pszFileName);
};


[uuid("88B735F4-EA32-4895-83F1-090A2F9E013F"),
 helpstring("SharedFile Type Library"),  lcid(0x0407), // germany
 version(1.0)
]
library SharedFile
{
        interface ISharedFile;
        interface ISharedFileWriter;
        interface ISharedFileReader;
};
```

Auszug aus der Log-Datei der Filter

```
CCC-TS-Filter.dll(tid d24)      30 : Sink Pin erzeugt
CCC-TS-Filter.dll(tid d24)     180 : Sink Pin erfolgreich verbunden
CCC-TS-Filter.dll(tid d24)     180 : Externes Capture Device gefunden
CCC-TS-Filter.dll(tid d24)     180 : MediaType geholt
CCC-TS-Filter.dll(tid d24)     180 : DVInfo Format
CCC-TS-Filter.dll(tid d24)     250 : Source Pin erzeugt
CCC-TS-Filter.dll(tid d24)     270 : DecideBufferSize
CCC-TS-Filter.dll(tid dc4)    2311 : ActualSamplePos (0x00000FC24, 144008 Bytes)
...
CCC-TS-Filter.dll(tid a94)    3267 : OnThreadCreate
CCC-TS-Filter.dll(tid a94)    3268 : OnThreadStartPlay 0 (0x00000FC24)
CCC-TS-Filter.dll(tid a94)    3268 : FillBufferSeekPos (0x00000FC24)
CCC-TS-Filter.dll(tid dc4)    3271 : ActualSamplePos (0x00035B8E4, 144008 Bytes)
CCC-TS-Filter.dll(tid a94)    3300 : FillBufferSeekPos (0x000032EAC)
CCC-TS-Filter.dll(tid dc4)    3311 : ActualSamplePos (0x00037EB6C, 144008 Bytes)
CCC-TS-Filter.dll(tid a94)    3321 : FillBufferSeekPos (0x000056134)
CCC-TS-Filter.dll(tid dc4)    3350 : ActualSamplePos (0x0003A1DF4, 144008 Bytes)
CCC-TS-Filter.dll(tid a94)    3361 : FillBufferSeekPos (0x0000793BC)
CCC-TS-Filter.dll(tid dc4)    3390 : ActualSamplePos (0x0003C507C, 144008 Bytes)
...
CCC-TS-Filter.dll(tid d24)  108137 : Live
CCC-TS-Filter.dll(tid d24)  108138 : ChangeStart
CCC-TS-Filter.dll(tid d24)  108138 : UpdateFromSeek
CCC-TS-Filter.dll(tid a94)  108138 : OnThreadStartPlay 1054000000 (0x0169CE774)
CCC-TS-Filter.dll(tid a94)  108138 : FillBufferSeekPos (0x0169CE774)
CCC-TS-Filter.dll(tid a94)  108160 : Warte auf neue Samples
CCC-TS-Filter.dll(tid dc4)  108178 : ActualSamplePos (0x0169F19FC, 144008 Bytes)
CCC-TS-Filter.dll(tid dc4)  108191 : ActualSamplePos (0x016A14C84, 144008 Bytes)
CCC-TS-Filter.dll(tid a94)  108200 : FillBufferSeekPos (0x0169F19FC)
CCC-TS-Filter.dll(tid a94)  108209 : FillBufferSeekPos (0x016A14C84)
...
```