Funktionsorientierte Bausteine zur Integration kontinuierlicher Medien in verteilte Anwendungen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik
der Johann Wolfgang Goethe–Universität
in Frankfurt am Main

von
Johannes Christian Fritzsche
aus Nürnberg

Frankfurt am Main
April 1997
(DF1)

vom Fachbereich Informatik der
Johann Wolfgang Goethe–Universität
als Dissertation angenommen.

Dekan: Prof. Dr. Oswald Drobnik

Gutachter: Prof. Dr. Oswald Drobnik

.....

Datum der Disputation: .....

An dieser Stelle ist es mir ein Bedürfnis, allen, die mich bei meinen Arbeiten unterstützt haben, zu danken.
Professor Dr. Oswald Drobnik für seine Aufgeschlossenheit und Großzügigkeit sowie seine geduldige Unterstützung, die meine Arbeiten erst ermöglicht haben. Die Diskussionen mit ihm und seine Fragestellungen waren immer wieder Anregungen zur Weiterentwicklung der Arbeit.

Professor Dr. Ralf Steinmetz gilt mein Dank für die intensiven Diskussionen, wertvollen Anregungen und fachlichen Hinweise, sowie die gemeinsamen Arbeiten.

Professor Dr. Kurt Rothermel danke ich für Ratschläge und Hinweise, die sehr zum Gelingen der Arbeit beigetragen haben.

Meinen Kollegen Magdalena Feldhoffer und Martin Zimmermann danke ich für die Unterstützung bei den täglichen Arbeiten, die kritischen Fragen und hilfreichen Diskussionen. Ebenso danke ich den Kollegen Christian Mönch, Jürgen Berghoff, Anselm Lingnau und Martin Hess für ihre Unterstützung. Ein besondere Dank geht an Marion Terrell.

Ameneh Alireza, Claudia Bastian, Therese Konstantinov, Hartmut Braun, Michael Danneberg, Frank Hesmert, den Studenten, ohne die eine so komplexe Arbeit nicht hätte entstehen können, gilt besonderer Dank.

Meinen Zimmergenossen Andrea Tschischka, Golamali Darvish und Andreas Heckwolf danke ich für das Ertragen meiner Launen und Selbstgespräche sowie des Chaos, das ich zeitweilig verbreitet habe.

Meinen Eltern und meinen Geschwistern, die meine Arbeiten stets aufmerksam verfolgt und kommentiert haben, danke ich sehr, auch ihre moralische Unterstützung war für mich sehr wertvoll.

Ganz herzlich danke ich meiner lieben Frau und meiner Tochter für ihre verständnisvolle Unterstützung und umfassende Fürsorge. Sie und Gottes Hilfe gaben mir die Kraft diese Arbeit zu beenden.

Inhaltsverzeichnis

1 Multimedia — Ein modernes Schlagwort 1
1.1 Formen der Tradierung 1
1.2 Intention der Tradierung 2
1.2.1 Autor, Leser und Dokument 2
1.2.2 Realisierung der Intentionen 3
1.3 Medien 4
1.3.1 Monomedia, Multimedia, Hypermedia 4
1.3.2 Umgang mit multimedialen Dokumenten 6
1.4 Integration der Medien in den Rechner 7
1.4.1 Entwicklung der Integration 7
1.4.2 Weitere Integrationsschritte 7
1.4.3 Anwendungsbeispiele 8
1.5 Zielvorstellungen und Gliederung der vorliegenden Arbeit 8

2 Multimediamodelle 11
2.1 Funktionsmodelle und Unterstützungssysteme 11
2.1.1 Erste und zweite Ebene 11
2.1.2 Dritte Ebene: Anwendungsmodellierung 14
2.1.3 Beispiele für Entwicklungsmodelle verteilter Multimediasysteme 16
2.1.4 Anforderungskatalog an Bausteine für kontinuierliche Medien 20
2.2 Datenmodelle 21
2.2.1 Kodierungen kontinuierlicher Medien 22
2.2.2 Semantische Datenmodelle 25
2.2.3 Zeitmodelle 28
2.2.4 Synchronisation 30
2.2.5 Datenübertragung 34
2.2.6 Anforderungskatalog für semantische Datenmodelle 35

3 Ein Funktionenmodell für Anwendungen kontinuierlicher Medien 37
3.1 Multimediale Anwendungen 37
3.1.1 Eine Beispielanwendung 37
3.1.2 Erstellung von Multimediaanwendungen 38
3.1.3 Verteilungsaspekte 39
3.2 Grundfunktionen kontinuierlicher Medien 40
3.2.1 Aufspaltung der Monomedien in Grundfunktionen 40
3.2.2 Basisanwendungen 41

3.2.3 Datenflußstruktur zwischen Grundfunktionen ... 42
3.2.4 Anwendungsbeschreibung im Grundfunktionenmodell ... 43
3.3 Erweiterung des Modells ... 45
3.3.1 Einordnung von Verarbeitungsfunktionen in das Funktionenmodell ... 45
3.3.2 Beschreibung der Verarbeitungsfunktionen ... 48
3.3.3 Eigenschaften und Nebenbedingungen der Verarbeitungsfunktionen ... 51
3.3.4 Anwendungsbeschreibung ... 52
3.3.5 zusammengesetzte Verarbeitungsfunktionen ... 55
3.4 Umgebungen für Funktionen ... 56
3.4.1 Einbettung der Grundfunktionen ... 57
3.4.2 Einbettung der Verarbeitungsfunktionen ... 58
3.4.3 Kopplung von Umgebungen ... 58

4 Ein Datenmodell für die Verarbeitung kontinuierlicher Medien ... 61
4.1 Datenmengen kontinuierlicher Medien ... 61
4.1.1 Integrierte Beschreibung der Eigenschaften von Einzeldaten ... 61
4.1.2 Eigenschaften von Datenmengen ... 62
4.1.3 Eigenschaften der Datenparameter in digitalen Rechnersystemen ... 72
4.1.4 Grundlagen des Zeitbezugssystems ... 74
4.1.5 Ticker und Schrittgeber ... 76
4.1.6 Ein Datentyp für kontinuierliche Medien ... 80
4.2 Strukturierung der Datenmengen kontinuierlicher Medien ... 82
4.2.1 Granularitäten aus Benutzersicht ... 83
4.2.2 Die Listenebene kontinuierlicher Medien ... 84
4.2.3 Die Sequenzebene kontinuierlicher Medien ... 85
4.2.4 Die Elementebene kontinuierlicher Medien ... 86
4.2.5 Anwendung der Modellierungsebenen ... 87
4.3 Audio und Video mit Zeitparametern ... 88
4.3.1 Modellierungsbeispiele für Audio und Video ... 88
4.3.2 Abbildung auf Listen, Sequenzen und Elemente ... 88
4.4 Integration des Datenmodells in das Funktionenmodell ... 90
4.4.1 Perzeption und Perzeptionsumgebung ... 90
4.4.2 Präsentation und Präsentationsumgebung ... 92
4.4.3 Speicherung ... 93
4.4.4 Verarbeitungsfunktionen ... 94
4.5 Beziehungen zwischen Sequenzen ... 95
4.5.1 Aspekte der Verwendung mehrerer Sequenzen ... 95
4.5.2 Intersynchronisation ... 96
4.5.3 Strategien zur Synchronisation ... 96

5 Umsetzung der Modelle auf Komponenten verteilter Systeme 99
5.1 Komponenten in verteilten Anwendungen 99
5.1.1 Schnittstellenspezifikation 100
5.1.2 Kommunikationskontext 101
5.1.3 Komponentenspezifikation 102
5.1.4 Anwendungskonfiguration 103
5.2 Umsetzungsstrategie für kontinuierliche Medien 104
5.3 Die Managementschnittstelle 105
5.4 Die Steuerschnittstelle 106
5.4.1 Eine allgemeine Steuerschnittstelle 107
5.4.2 Perzeption und Präsentation 107
5.4.3 Speicher 110
5.4.4 Verarbeitungsfunktionen 111
5.5 Die Datenschnittstelle 112
5.6 Kommunikationsanforderungen 113
5.6.1 Anforderungen der Steuerschnittstelle 114
5.6.2 Anforderungen der Datenschnittstelle 118
5.7 Beschreibung der Komponenten 121
5.7.1 Grundfunktionskomponenten 121
5.7.2 Verarbeitungskomponenten 124
5.7.3 Steuerkomponenten 125
5.8 Anwendungskonfiguration 127
5.8.1 Beispielanwendungen 127
5.8.1.1 Ein einfaches Aufzeichnungssystem 127
5.8.1.2 Ein einfaches Wiedergabesystem 129
5.8.1.3 Bearbeitungssysteme 130
5.8.2 multifunktionale und multimediale Komponenten 130
5.8.3 Auswahl von Komponenten 131
5.8.4 Umkonfiguration der Anwendung 132

6 Resümee und Ausblick 135
6.1 Entwicklung der Konzepte und Modelle 135
6.2 Implementierungen 137
6.2.1 Realisierung von Komponenten 137
6.2.2 Realisierung von Tickern, Schrittgebern und enger Kopplung 139
6.3 weitere Entwicklung 139

Literatur 143

Anhang A Schnittstellen und Kommunikationskontexte 153

A.1 management 153
A.2 data 153
A.3 control 153
A.4 Kommunikationsanforderungen 156

Anhang B Komponentenbeispiele 161

Anhang C Klassendefinitionen 169

Lebenslauf 171

Abbildungen

Abb. 1 Ein Medium vermittelt zwischen der Umwelt und dem Benutzer 4
Abb. 2 Medienaufteilung nach Grundfunktionen 40
Abb. 3 Kategorien von Anwendungsmedien 42
Abb. 4 Spaltung der Grundfunktionen in Quellen und Senken 43
Abb. 5 Anwendungsmedien der Kategorien 1 bis 4 im Grundfunktionenmodell 44
Abb. 6 Funktionseinheiten 5 bis 9 im Grundfunktionenmodell 45
Abb. 7 das Funktionenmodell mit Verarbeitungsfunktionen 46
Abb. 8 Grundfunktionen mit Mischen und Verteilen 46
Abb. 9 Mischen und Verteilen mit Verarbeitungsfunktionen 47
Abb. 10 Das Funktionenmodell 47
Abb. 11 Beispiele für einfache Verteilung $f_5°$ und einfaches Mischen f_5* 48
Abb. 12 verallgemeinerte Verteilung und verallgemeinertes Mischen 49
Abb. 13 Aufzeichnungssystem ohne und mit Aussteuerregelung (Verstärkung) 53
Abb. 14 Wiedergabesystem ohne und mit Verstärkung 53
Abb. 15 Lesersystem ohne Live–Präsentation 53
Abb. 16 Lesersystem mit Live–Präsentation 54
Abb. 17 Autorensystem 54
Abb. 18 eine Kette von Funktionen 55
Abb. 19 zusammengesetzte Verarbeitungsfunktionen 55
Abb. 20 zusammengesetztes und zerlegtes Verteilen und Mischen 56
Abb. 21 Funktionen und Umgebungen am Beispiel des Autorensystems 57
Abb. 22 Beispiele für Datenströme mit möglichen Lücken und Überschneidungen 65
Abb. 23 mögliche Überschneidungen zweier aufeinanderfolgender Werte 67
Abb. 24 Das Zeitbezugssystem 75
Abb. 25 mögliche Granularitätsstufen für Audio und Video 83
Abb. 26 Basisbausteine für die Konfiguration verteilter Anwendungen [Zimm93] 99
Abb. 27 Beispiel eines Pfadausdrucks 100
Abb. 28 Struktur der Schnittstellenspezifikation 101
Abb. 29 Struktur der Komponentenspezifikation 103
Abb. 30 Struktur der Anwendungskonfiguration 103
Abb. 31 Komponenten kontinuierlicher Medien und ihre Schnittstellen 104
Abb. 32 Eine einfache Multimediaanwendung in Komponentendarstellung 105
Abb. 33 Spezifikation der allgemeinen Managementschnittstelle 106

Abb. 34 Spezifikation der Prototypsteuerschnittstelle für Grundfunktionen ... 107
Abb. 35 Ableitungswurzeln für die Schnittstellen von Quellen und Senken ... 107
Abb. 36 Spezifikation der gemeinsamen Steuerschnittstelle für Perzeption und Präsentation ... 108
Abb. 37 Spezifikation einer allgemeinen Steuerschnittstelle für die Perzeption ... 108
Abb. 38 Erweiterung der Steuerschnittstelle für Element–Mehrschritt–Strategie ... 109
Abb. 39 Spezifikation der allgemeinen Schnittstelle für die Präsentation ... 109
Abb. 40 einfache Schreib– und einfache Lese–Steuerschnittstelle ... 110
Abb. 41 einfache Schreib/Lese–Steuerschnittstelle ... 110
Abb. 42 Schreib– und Lese–Steuerschnittstelle für Sequenzen ... 111
Abb. 43 Beispiel einer Steuerschnittstelle für Verarbeitungsfunktionen ... 112
Abb. 44 Spezifikation der allgemeinen Datenschnittstelle ... 113
Abb. 45 Kommunikationskontextpaar für Steuerschnittstellen in loser Kopplung ... 115
Abb. 46 Kommunikationskontextpaar für Steuerschnittstellen in enger Kopplung ... 117
Abb. 47 Kommunikationskontextpaar für Speicher–Steuerschnittstellen ... 117
Abb. 48 Schablone für die Kommunikationskontexte der Datenschnittstelle [Alireza94] ... 120
Abb. 49 ein Datenschnittstellenkommunikationskontextpaar ... 120
Abb. 50 graphische Darstellung der Komponenten eines Aufzeichnungssystems ... 128
Abb. 51 graphische Darstellung der Komponenten eines Wiedergabesystems ... 130
Abb. 52 Spezifikation einer zusammengesetzten Aufnahme–Komponente ... 131
Abb. 53 Konfiguration von Komponenten ... 132
Abb. 54 Elementare Objektarchitektur [Zimm93] ... 138

Tabellen

Tab. 1 Funktionseinheiten nach [SteiMey92] ... 18
Tab. 2 Multimediadaten und QoS Parameter ... 35
Tab. 3 Verhältnis der Schrittraten zum 600 Hz Ticker ... 76

1 Multimedia — Ein modernes Schlagwort

Der Begriff Multimedia wird heute in vielen verschiedenen Zusammenhängen benutz. Im Bereich der Informatik kennzeichnet er die Integration verschiedener Repräsentationen für Informationen im Rechner. Somit soll jeder Benutzer an seinem Arbeitsplatz verschiedene Medien, wie Text, Bild, Ton, Video, etc. zur Verfügung haben. Dieses Kapitel bietet eine Einführung in die Ideen und Grundlagen der Integration in den Rechner.

1.1 Formen der Tradierung

Multimediale Systeme werden häufig als das wichtigste Mittel zur Information genannt [Brand90]. Sie sollen es erleichtern, Wissen weiterzugeben. Dazu sind auf den unterschiedlichsten Gebieten Ansätze unternommen worden, die die verwendeten Techniken in den Vordergrund stellen [GloStr90]. Hier soll zunächst auf die Weitergabe von Wissen, Erfahrung und Information eingegangen werden.

Um Wissen und Erfahrung zu dokumentieren und zu tradieren, wurden verschiedene Methoden entwickelt. Strukturierung der Information ermöglicht ein besseres Verstehen der Dinge und kann so zum Erkennen von Fehlern oder Schwächen führen. Die strukturierte Tradierung von Wissen und Erfahrung erleichtert die Verknüpfung mit bereits vorhandenem Wissen. Die Strukturierung von Wissen erleichtert auch das Wiederauffinden von Informationen in Informationssammlungen. Auf diese Art werden auch Lernbarkeit und Lehrbarkeit von Wissen und Information gefördert. Die Struktur fördert aber auch die Wiederholbarkeit, da die Struktur für das Abbrechen und Wiederanfangen geeignete Punkte liefert. Das strukturierte Beschreiben von Wissen erleichtert so dessen Weitergabe und Weiterentwicklung. Da Methoden zur Strukturierung vom Inhalt der Information beeinflußt werden, soll zunächst vom Inhalt abgesehen werden. Somit stehen vor den Strukturierungsmethoden die Strukturierungsmittel für Wissen und Erfahrung.

Erfahrung geschieht beim Menschen durch die Wahrnehmung mit den Sinnen. Für die Tradierung sensitiver Eindrücke und von Wissen müssen diese aufgezeichnet, konserviert und wiedergegeben werden können. Von den fünf Sinnen Schmecken, Riechen, Tasten, Hören und Sehen wurden besonders für die beiden letzten technische Mittel zur Aufzeichnung, Konservierung und Wiedergabe entwickelt. Die Strukturierung von Wissen und Erfahrung kann sowohl bei der Aufzeichnung als auch bei der Konservierung und ebenso bei der Wiedergabe erfolgen oder wieder neu erfolgen.

Die ersten Ansätze der Tradierung liegen in der Übertragung der erlebten dreidimensionalen Welt auf flache, zweidimensionale Bilder. Das wesentliche an Bildern oder Graphiken ist, daß sie ihre Information sofort und direkt vermitteln. Der Nachteil liegt darin, daß sich nur schwer Entwicklungen aufzeigen lassen. Abhilfe schaffen hier Folgen einzelner Bilder oder Film, die die Information im Ablauf der Zeit darstellen. Für den Film oder Bilderfolgen ist es wichtig, zu welchem Zeitpunkt sie aufgenommen werden, das heißt, welche Situation sie darstellen. Dazu kommt, daß bei diesen Medien eine Strukturierung, wie zum Beispiel in klar voneinander abgesetzten Szenen, ein wesentliches Gestaltungselement ist.

Sieht man als Objekt der Tradierung Wissen, das das Ergebnis mentaler Prozesse darstellt, so ergibt sich zunächst das Problem dieses Wissen auszudrücken. Das Mittel dazu ist die Sprache. Sie beschreibt Dinge durch die Verwendung abstrakter Begriffe. Gegenstand der Beschreibung können sowohl sensitiv wahrnehmbare Dinge als auch abstrakte, mit den Sinnen nicht wahrnehmbare Dinge sein. Da die Sprache Information in der Abfolge der Zeit wiedergibt, ist es zur Tradierung erforderlich, Sprache zu konservieren. Dazu muß das zeitabhängige akustische Signal *Sprache* in ein zeitunabhängiges Signal gewandelt werden, die Sprache ist zu kodieren.

Eine Vorschrift, ein zeitabhängiges Signal in ein zeitunabhängiges zu wandeln, nennt man Schreibvorschrift, die Umkehrung Lesevorschrift. Somit ist ein wichtiges Stichwort gefallen: schreiben. Die Schrift ist das älteste Mittel Sprache aufzuzeichnen und zu konservieren. Sprache läßt sich auch, wie jedes andere akustische Signal, mit technischen Hilfsmitteln wie zum Beispiel dem Tonband aufzeichnen, konservieren und wiedergeben. Auch das Aufnehmen auf ein Tonband ist ein Schreibvorgang.

Da die Schrift erst gelesen werden muß, um ihre Information wiederzugeben, gibt auch sie ihre Information erst im Verlauf der Zeit ab, ist aber insofern zeitunabhängig, als man jedes Wort zu jeder beliebigen Zeit lesen kann. Schrift in diesem Sinne ist nicht beschränkt auf die uns bekannten, optisch wahrnehmbaren Schriften, sondern ist zum Beispiel auch die Blindenschrift und die Knotenkordeln, die bei einigen Indianervölkern in Gebrauch waren. Bei der Blindenschrift ist der Lesevorgang die Interpretation einer Folge von Tasteindrücken.

1.2 Intention der Tradierung

1.2.1 Autor, Leser und Dokument

Die Tradierung soll Wissen oder Erfahrung einer Person einer anderen vermitteln. Man nennt die erste Person **Autor**, die zweite **Leser** und das die Tradierung beinhaltende Objekt **Dokument**. Autor und Leser haben an die Tradierung und damit an das Dokument besondere Anforderungen. Sie erwarten, daß die Tradierung eine unveränderte Wiedergabe eines realen Geschehens vermittelt, oder, daß beim Leser die selbe Vorstellung erzeugt wird, die der Autor hat. Dies ist nicht zweimal die selbe Forderung, da der Autor auch nicht reale Dinge beschreiben kann, die nur in seiner Vorstellung existieren. Die Forderungen sind in ihrer Absolutheit nicht zu erfüllen, sondern nur annähernd zu erreichen.

Der Autor möchte beim Leser einen bestimmten Eindruck erzeugen. Es hängt vom Autor ab, ob dieser Eindruck allgemein gehalten ist, oder ob ein ganz bestimmter Eindruck genau erzeugt werden soll. Je genauer der Eindruck sein soll, desto mehr Sinneseindrücke sollen dem Leser vermittelt werden und desto größer ist der Aufwand. Der Autor berücksichtigt die Erwartungen des Lesers, also dessen Anforderungen an das Dokument. Dazu wählt er eine bestimmte Form für das Dokument.

Die Erwartungen der Leser an ein Dokument lassen sich nicht in ein gemeinsames Konzept pressen, sondern beschreiben verschiedene Klassen von Dokumenten. Die verbreitetste Dokumentform sind Bücher. Sachbücher stellen Informationen zu bestimmten Themengebieten zusammen, Lehrbücher und Lehrkurse, die den Leser als Schüler ansprechen, bereiten Informationen speziell für den Wissenserwerb auf, im allgemeinen in Verbindung mit Übungen für den Leser, es gibt Nachschlagewerke und Handbücher und noch viele weitere. Für

alle Beispiele sind multimediale Rechneranwendungen vorgeschlagen worden. Nur selten werden dagegen multimediale Rechneranwendungen für Belletristik oder dramatische Werke angesprochen. Da diese aber zur wirtschaftlich wichtigen Unterhaltungssparte zählen, werden Rechneranwendungen in diesem Bereich in Zukunft wesentlich an Bedeutung gewinnen.

1.2.2 Realisierung der Intentionen

Oft erfordert die Intention der Tradierung, daß Geschehen aus der realen Welt aufgezeichnet wird. Die Aufzeichnungen können in vielfältiger Weise verwendet, analysiert und bearbeitet werden. Eine Intention kann beispielsweise sein, Rückschlüsse auf das ursprüngliche, reale Geschehen zu ziehen. Typisch dafür ist die Bildaufzeichnung von Bewegungsabläufen und deren Wiederholung in Zeitlupe oder Einzelbildern. Schwächen können festgestellt werden und das reale Geschehen kann optimiert werden. Oft vermittelt eine Aufzeichnung auch erst das Verständnis für das reale Geschehen.

Liegt ein Teil der Intention dagegen in der Verbesserung der Qualität der Aufzeichnung gegenüber dem realen Geschehen, so stehen auch verschiedene Möglichkeiten zur Bearbeitung der Aufzeichnungen zur Verfügung. Neben dem Verhältnis zum realen Geschehen oder zur Qualität der Aufzeichnung gegenüber dem realen Geschehen, kann auch die bearbeitete Wiedergabe der Aufzeichnung für sich interessieren. Bearbeitungsmöglichkeiten sind unter anderen: Schnitt, Zeitlupe (slow motion), Zeitraffer (fast motion), Ein– und Ausblendung (fade in, fade out, fade to dark, fade to white).

Moderne Literaturwissenschaft beschäftigt sich auch mit Ton– und Bilddokumenten. Für die speziellen Anforderungen der Literatur und der verschiedenen Formen der Tradierung wurden Hilfsmittel geschaffen, die Aufzeichnung, Konservierung und Wiedergabe optimieren. Solche Hilfsmittel sind zum einen von wissenschaftlich didaktischer Natur, zum anderen sind es technische Geräte und Werkzeuge. Eine weitergehende Betrachtung der Intention der Tradierung sowie der Einteilung und Klassifizierung der Dokumente bleibt der Literaturwissenschaft vorbehalten. Hier wird dagegen die technische Unterstützung betrachtet.

Die zur Zeit erlebte Weiterentwicklung der technischen Hilfsmittel erlaubt es, mit unseren Sinnen Wahrnehmbares, in der Zeit Vergängliches aufzuzeichnen, zu speichern, wiederzugeben und verschiedene Aufzeichnungsarten ineinander zu überführen. Zum Beispiel kann man einen elektronisch gespeicherten Text als Buchstaben auf einem Drucker oder Bildschirm ausgeben oder auch als Tastsignal in einem Blindenlesegerät. Neuere Entwicklungen unterstützen auch das Vorlesen von Texten durch den Rechner. Die Kombination der Tradierung der verschiedenen Sinneseindrücke mit verschiedenen Tradierungstechniken führt zum Entstehen verschiedener Medien. Die Entwicklungen der Kommunikationstechnik erlauben es, diese Medien über weite Strecken zu verbreiten.

1.3 Medien

1.3.1 Monomedia, Multimedia, Hypermedia

Die verschiedenen Mittel zur Aufzeichnung, Speicherung, Bearbeitung, Übertragung und Wiedergabe der Sinneseindrücke heißen Medien. **Medium** ist lateinisch und bedeutet das in der Mitte Befindliche, im weiteren Sinne auch das Vermittelnde. Die in den Medien gespeicherten Sinneseindrücke heißen **Mediendaten**. Ein Medium überwindet zeitliche oder räumliche Distanzen zwischen der Umwelt, aus der es die Mediendaten gewinnt, und seinem Benutzer, dem es die Mediendaten vermittelt oder präsentiert.

Medien, die genau eine Präsentationsform verwenden, heißen **Einzel-** oder **Monomedien**. Schrift, Bild und Ton sind die klassischen Einzel- oder Monomedien. Dazu kommen heute vor allem die bewegten Bilder in der Form von Film und Video; weitere Entwicklungen sind zum Beispiel taktile Systeme, die besonders von Blinden genutzt werden. Aus einzelnen dieser Monomedien können verschiedene monomediale Dokumente erstellt werden.

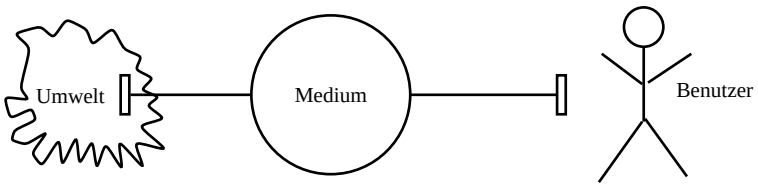


Abb. 1 Ein Medium vermittelt zwischen der Umwelt und dem Benutzer

Die Bezeichnung Monomedium weist schon darauf hin, daß es neben diesen auch kombinierte Medien gibt. Ganz einfache Beispiele für kombinierte Medien sind illustrierte Bücher, der Stummfilm mit seinen Texttafeln oder der Tonfilm. Bei der Kombination der Monomedien werden verschiedene Beziehungen zwischen diesen erstellt. Die wesentliche Eigenschaft der genannten Dokumente ist, daß die Beziehungen nach Fertigstellung des Dokuments unveränderlich fest liegen. Eine Veränderung der Beziehungen bewirkt eine Neuerstellung des Dokuments. In dieser Form kombinierte Medien heißen abhängige Medien vgl. auch [Stein90]. Als Beispiel sei hier ein Lichttonfilm genannt, da sich bei diesem Bild und Ton weder trennen noch gegeneinander verschieben lassen.

Die Beziehungen zwischen den abhängigen Medien werden vom Autor festgelegt und ihre Unveränderlichkeit ist zunächst durchaus erwünscht, da der Leser sie nicht beliebig ändern können soll. Auch ist der Leser eines Dokuments nicht in jedem Fall daran interessiert, daß die Beziehungen verändert werden (können). Zum Beispiel soll in einem Videospielfilm die Sprache zu den Lippenbewegungen des Schauspielers passen. Ein anderer Aspekt ergibt sich für den Autor bei der Erstellung des Dokuments, insbesondere bei der Komposition der Monomedien. Meist liegen diese zunächst in unabhängiger Form vor, müssen bearbeitet werden, werden zusammengesetzt und wieder getrennt, bis das Gesamtergebnis vom Autor akzeptiert wird. Für diese Schritte der Kombination ist es daher nicht sinnvoll abhängige Medien zu verwenden.

Insgesamt besteht die Forderung an Multimediasysteme nach mehr Flexibilität als von Dokumenten mit abhängigen Medien geboten wird [Niels90]. Daher beläßt man die Monomedien unabhängig voneinander und integriert sie, indem man Beschreibungen der zeitlichen, räumlichen und inhaltlichen Beziehungen zwischen ihnen hinzufügt [Stein90]. Ein Dokument besteht dann aus den Daten der Monomedien und der Beschreibung der Beziehungen zwischen den Monomedien.

Der Begriff Multimedia ist in der Informatik nicht neu erfunden worden, sondern war schon früher bekannt. Die größte Verbreitung fand er als Begriff Multimediashow, womit meist eine Diavorführung mit begleitender Musik und gesprochenem Text gemeint war; die zeitunabhängigen Dias werden mit der zeitabhängigen Musik verbunden, ohne sie fest miteinander zu verkoppeln. Eine feste Verkopplung wäre ein auf dem Diarahmen aufgeklebter Tonträger mit dem gesprochenen Text. Weitere Beispiele multimedialer Systeme sind Bücher, denen Schallplatten oder Tonbandkassetten beiliegen. Die bekanntesten Vertreter dieser Richtung sind Fremdsprachenkurse oder Vogelstimmenlehrsysteme. Alle diese Systeme setzen eine direkte Einflußnahme des Menschen bei der Lenkung des Zusammenspiels der einzelnen Medien voraus. Zum Beispiel muß der Leser des Fremdsprachenkursdokuments das Tonband zur richtigen Lektion starten und dann eventuell den wiedergegebenen Text im Buch mitlesen.

Eine weitere Forderung an Multimediasysteme ist die freie Gestaltung von Zusammenhängen innerhalb eines Dokuments. Dabei kann die sequentielle Struktur, wie zum Beispiel der Bücher, aufgehoben werden [GloStr90] [DürNes90]. Die Dokumente bestehen dann aus einem Netz von Seiten, welche selbst aus sogenannten Partikeln aufgebaut sind [HoCoLa89] [SülCor90]. Diese Dokumente werden auch als Hyperdokumente bezeichnet. Werden für Partikel nur zeitunabhängige Medien, wie zum Beispiel Text, Graphik, Tabellen oder Standbild verwendet, so spricht man von **Hypertext** [Conkli87]. Dagegen spricht an von **Hypermedia**-Systemen, wenn auch kontinuierliche Medien, wie Film/Video oder Audio, für Partikel verwendet werden.

Da die Medien eines Hyperdokuments voneinander unabhängig sind, werden Hyperdokumentsysteme häufig auch als Multimediasysteme bezeichnet. Mit der Verbreitung des Begriffs *Multimedia* als Schlagwort, haben sich unterschiedliche Vorstellungen davon eingestellt, was dieser Begriff bedeutet. In [Stein93] wird daher eine restriktive Definition für Multimediasysteme vorgestellt.

«Ein Multimediasystem ist durch die rechnergestützte, integrierte Erzeugung, Manipulation, Darstellung, Speicherung und Kommunikation von unabhängigen Informationen gekennzeichnet, die in mindestens einem kontinuierlichen (zeitabhängigen) und einem diskreten (zeitunabhängigen) Medium kodiert sind.»

Statt von kontinuierlichen Medien wird auch von zeitabhängigen Medien gesprochen, um die mögliche ständige Veränderung der Umweltparameter zu betonen. Der Begriff kontinuierliches Medium rückt die Aufeinanderfolge der Elemente bei der Aufnahme oder der Präsentation in den Vordergrund. Diese Kontinuität wird im Rechner aufgebrochen und durch diskrete zeitabhängige Daten ersetzt. Daher wird in dieser Arbeit von den zeitabhängigen Daten der kontinuierlichen Medien gesprochen. An den Stellen, wo Medien miteinander verglichen werden, wird auf das eingängige Gegensatzpaar zeitabhängige Medien und zeitunabhängige Medien zurückgegriffen.

1.3.2 Umgang mit multimedialen Dokumenten

Der Umgang mit multimedialen Dokumenten beinhaltet das Aufnehmen, Speichern, Bearbeiten, Wiedergeben, Ordnen und Wiederfinden von Einzeldokumenten in großen Dokumentmengen. Das sind klassische Gebiete der Informatik, die zu einem Teil in den Bereich der Informationssysteme, insbesondere der Datenbank- und Dokumentnachweissysteme fallen. Entwicklungen in diesem Bereich beginnen auch mit der Integration von Bild- und Tondokumenten durch formalisierte Beschreibung des Dokuments und Verweis auf den Speicherort. Ebenso gibt es bereits verschiedene Werkzeuge zur Bearbeitung und Manipulation der einzelnen Medien. Ein Teil der derzeitigen Entwicklung auf dem Gebiet der Informationssysteme ist die Integration dieser Werkzeuge. Dabei ergeben sich verschiedene Schwierigkeiten. Zum einen verwalten Informationssysteme nur Informationen über die Bild- oder Tondokumente, anstatt den Inhalt der Dokumente. Dafür wären Bild– oder Ton–Analyseverfahren zu integrieren, die Anfragen nach "dem Bild mit Onkel Otto" oder der Musik mit der Tonfolge "bach" erlauben. Für diese Zwecke existieren noch keine geeigneten Werkzeuge. Desweiteren fehlt Informationssystemen die Fähigkeit der Interpretation der Beziehungsinformation zwischen den Dokumenten. Insbesondere ist hier die Strukturinformation von Hypermediadokumenten zu nennen. Ein weiterer wichtiger Punkt ist die Integration von kontinuierlichen Medien, die ja erst zu den multimedialen Systemen führen. Die Zeitabhängigkeit der kontinuierlichen Medien ist in Datenbanksystemen nur schwer realisierbar, da Datenbanken kein Modellierungsmittel für die real ablaufende Zeit kennen. Dazu muß die Zeitabhängigkeit erst in den Daten beschrieben werden. Die Interpretation der Zeitabhängigkeit ist dann vor allem beim Zugriff für die Präsentation der Daten nötig.

Die Forschung im Bereich der multimedialen Systeme beschäftigt sich damit, dem Anwender die Medien und Operationen auf ihnen integriert so anzubieten, daß Beziehungen zwischen den Monomedien hergestellt werden können. Der Rechner hat sich bei der Be– und Verarbeitung der gespeicherten Form der Medieninhalte, den Mediendaten, als praktisches und leistungsfähiges Werkzeug erwiesen. Technische Schwierigkeiten beim Rechnereinsatz bestehen noch bei der Aufzeichnung und Wiedergabe von einzelnen Medien im Vergleich mit der Qualität und Effizienz der vorhandenen Alternativen, meist analogen Verfahren. Beispiele für hochqualitative analoge Verfahren sind die hohen Informationsdichten von Photographie, Holographie sowie Film und Multispur–Audioaufnahmen.

Die Vorteile der Verwendung von Rechnern für multimediale Anwendungen liegen in der Möglichkeit der Beschreibung und Verarbeitung einer Vielzahl von Beziehungen zwischen den Monomedien und der Integration aller Monomedien in einem Gerät. Die Digitalisierung im Rechner bringt als weiteren Vorteil mit sich, daß alle Daten beliebig oft ohne Informationsverlust vervielfältigt werden können. Alle Kopien sind mit dem Original identisch. Die Analyse, Beschreibung und Klassifikation der semantischen Beziehungen zwischen den Medien ist Aufgabe der Forschung im Bereich der Hypermediasysteme. Im multimedialen Bereich werden vor allem die zeitlichen Beziehungen zwischen den kontinuierlichen Medien behandelt.

1.4 Integration der Medien in den Rechner

1.4.1 Entwicklung der Integration

Bei der Integration der Medien in den Rechner können verschiedene Stadien unterschieden werden. Der erste Schritt zur Integration war, herkömmliche Geräte, wie Kameras, Tonbandgeräte oder Schnitt– und Mischgeräte, mit einer digitalen Steuerschnittstelle zu versehen und so über einen Rechner bedienbar zu machen. Daraus ergibt sich, daß bei der Integration der Medien in den Rechner stets sowohl die Hardware als auch die Steuerung, das heißt die Software, zu beachten sind [SteiHe91]. Darüber hinaus sollten die Daten sowie ihre Übertragung und Speicherung betrachtet werden.

Auf der Basis der digitalen Steuerung durch Rechner verschiedener Leistungsklassen wurden große Multimediaanwendungen im kommerziellen Bereich aufgebaut. Dabei werden analoge Daten weiterhin außerhalb des Rechners zwischen ebenfalls externen Geräten ausgetauscht. Man spricht von hybriden Systemen, wie zum Beispiel IMAL (**I**ntegrated **M**edia **A**rchitecture **L**aboratory) [IMAL]. In diesen Systemen existieren auch digitale Komponenten, wie Vermittlungsstellen, die allerdings analoge Daten vermitteln.

Waren in den ersten Systemen die Geräte für kontinuierliche Medien und die Rechner noch streng getrennt, so wurde in der Weiterentwicklung die Präsentation der kontinuierlichen Medien in den Rechner integriert; zunächst geschah das zum Beispiel analog, über eine Videomischung um Fernsehbilder auf dem Monitor des Rechners darstellbar zu machen. Gleichzeitig wurden Verfahren zur Digitalisierung der Analogsignale entwickelt, die es erlauben, die kontinuierlichen Medien über digitale Netze zu übertragen. Diese digitalen Daten haben die Eigenschaft, daß sie keine direkte Korrelation mit den Umweltdaten haben, die sie repräsentieren. Eine Veränderung der digitalen Daten kann diese Werte vernichten. Zur Darstellung müssen die digitalen Daten erst wieder in analoge Daten gewandelt werden, die dann nach analogen Verfahren interpretierbar sind. Die Integration beschränkt sich hier auf das gleiche Transportverfahren.

Diese Technik der Verwendung digitaler Netze zum Transport der Daten, die Reanalogisierung und anschließende analoge Mischung der kontinuierlichen Medien zu den rechnerinternen diskreten Medien ist eine weit verbreitete Realisierung von multimedialen Systemen.

1.4.2 Weitere Integrationsschritte

Die Entwicklung in der Multimediatechnik geht dahin, die Daten an der Quelle zu digitalisieren. Beispiele dafür sind die Kodierungen Audio-CD, DVI, JPEG, MPEG [Stein93]. Mit diesem Schritt können sowohl die Daten als auch die Verarbeitung in den Rechner integriert werden. Die zur Zeit dafür verwendeten Modelle vernachlässigen aber, daß die Daten der kontinuierlichen Medien, verglichen mit den bisherigen Daten, wesentlich neue Semantik in den Rechner bringen.

Mit der Digitalisierung der Daten kontinuierlicher Medien entwickelte sich auch die Hardware der Geräte weiter, die jetzt als eigenes System von Prozessoren angesehen werden kann. Es ergibt sich somit für jedes Multimediasystem die Betrachtungsmöglichkeit als verteiltes System mit verschiedenen Prozessoren, die durch Datenübertragung miteinander verbunden sind. Die Prozessoren können sich eng gekoppelt innerhalb eines Rechners befinden oder lose gekoppelt über verschiedene Rechner verteilt sein. Für diesen Integrationsschritt

verteilter Verarbeitung müssen noch geeignete Beschreibungsmittel gefunden werden. Die Voraussetzung dafür ist, daß die Systemteile klare Funktionalitäten an definierten Schnittstellen anbieten, so daß die Verteilung für den Benutzer transparent wird. Die Verteilung darf auf die Modellierung der Anwendung keinen Einfluß haben. Aber die Anwendung muß die Lokalisierung der Systemteile steuern können. Die letzte Forderung ergibt sich aus der Ortsgebundenheit der Benutzer von Multimediasystemen.

1.4.3 Anwendungsbeispiele

Existierende Multimediasysteme sind häufig noch dediziert erstellte Programmsammlungen zur Unterstützung genau einer Anwendung. Einige Multimediasysteme unterstützen die Erstellung einer bestimmten Klasse von Anwendungen wie z. B. Telekonferenzen oder Ausbildungskurse. Weiterhin gibt es die sogenannten Autorensysteme, die die Erstellung und Manipulation multimedialer Dokumente erlauben [CACM89]. Schließlich gibt es noch die Anwendungssysteme, die im wesentlichen zum Lesen der multimedialen Dokumente dienen. Solche Systeme verarbeiten meist gleichzeitig eine Hypertextstruktur der Dokumente. Einen ausführlichen Überblick über Multimediaanwendungen bietet [Kaul95].

Neben den Anwendungen, die die Verwendung von kontinuierlichen Medien in den Vordergrund stellen, gibt es zahlreiche, häufig verteilte Anwendungen, die um kontinuierliche Medien erweitert werden. Beispiele hierfür sind Multimediamailsysteme wie MIME (Multipurpose Internet Mail Extensions) [MIME] oder dem BERKOM Multimedia Mail Teleservice [Blum93] [AlSchTh94] sowie die Erweiterung von Gruppenarbeitssystemen um Bild und Ton der Teilnehmer beziehungsweise einblendbares Video wie in JVTOS (Joint Viewing and Tele–Operation Service) [GuAL93], CoDRAFT [KLLMW93] und TEATIME [SemRo93].

Gemeinsam ist diesen Arbeiten, daß bestehende Anwendungen um Multimediafunktionen erweitert werden. Dabei werden jeweils ähnliche Funktionen für jede Anwendung neu erstellt. Besser wäre ein Konzept, das eine allgemeine Spezifikation und Realisierung von Multimediafunktionen einmal universal verwendbar erstellt. Dieser Satz von Funktionen ist dann in beliebigen Anwendungen verwendbar.

1.5 Zielvorstellungen und Gliederung der vorliegenden Arbeit

Multimediale Kommunikation besteht heute bereits in verschiedenen Ausprägungen und allgemeinen Anwendungen. Als Beispiel sei vor allen anderen das World Wide Web (WWW) genannt. Darin werden Daten und Programme bereitgestellt, die von einem Benutzer zu einer beliebigen Zeit ausgewählt und benutzt werden können. Wesentliches Element des Arbeiten im WWW ist, wie in seiner Grundlage, dem Internet, der wahlfreie, dezentrale und asynchrone Zugriff auf die bereitgestellten Daten und Programme. Jeder Anwender steuert selbst, was er wann benutzen möchte. Einzelne Daten können aber aus Gruppen bestehen, mit der exakten Beschreibung von Zusammenhängen zwischen den Gruppen und den Elementen dieser Gruppen. Zum Beispiel kann in einer Textseite ein Bildbereich für eine Animation vorgesehen sein, wobei die Animation aus einer Folge von Einzelbildern besteht.

Im Gegensatz zu Anwendungen, wie dem WWW, die nur durch lokale Interaktion der einzelnen Anwender gesteuert werden, stehen Anwendungen, die eine zentralisierte Steuerung benötigen, damit an verschiedenen, verteilten Orten eine Kooperation ermöglicht wird. Ein

Beispiel hierfür sind Video-Konferenzsysteme. Verallgemeinert sind das Systeme, bei denen zur Erlangung eines gemeinsamen Ziels verschiedene verteilte Anwendungskomponenten durch eine gemeinsame Steuerung koordiniert zusammenarbeiten. Diese Art der verteilten Systeme soll mit multimedialen Funktionen ausgestattet werden. Wesentliche Gesichtspunkte sind dabei die Kooperation der verteilten Komponenten und die dazu notwendige Steuerung.

Bei der Erstellung multimedialer Anwendungen ergeben sich für den Entwerfer oder Programmierer besondere Problemstellungen bei der Modellierung der Anwendung und ihrer Strukturierung in Bezug auf die Verwendung kontinuierlicher Medien. Diese Problemstellungen sind gerade für kontinuierliche Medien immer wieder die gleichen oder zumindest sehr ähnlich.

Die Entwicklung von Multimediaanwendungen vollzieht sich in einer dreistufigen logischen Struktur, wobei auf jeder Stufe mehrere Verfeinerungen und Neukonzeptionen auftreten können. Die erste Stufe ist die Modellierung des Multimediadatenflusses von Gerät zu Gerät. Die zweite Stufe ist die Gruppierung der Geräte zu einheitlich gesteuerten komplexen Geräten. In der dritten Stufe werden die Zeitbedingungen zwischen den Geräten geregelt. Ein Ergebnis auf einer Stufe kann zu einer Revision der Konzeption auf einer früheren Stufe führen.

Bisherige Ansätze zur Unterstützung der Erstellung multimedialer Anwendungen stellen die Optimierung der Leistung in den Vordergrund. Da es sich bei Multimediaanwendungen zunehmend um große, verteilte Anwendungen handelt, gewinnen aber die Aspekte der Wiederverwendbarkeit und der Struktur der Anwendungen an Bedeutung. Daher soll eine Anwendungsmodellierung einerseits die drei Entwicklungsstufen widerspiegeln, andererseits aber auch ein einfaches Wechseln zwischen den Ebenen ermöglichen.

Derzeitigen Ansätzen fehlt meist die allgemeine Interoperabilität der von ihnen realisierten Komponenten. Diese kann nur durch eine umfassende, formale Spezifikation der Komponentenschnittstellen erreicht werden. Neben den Schnittstellen zur Datenweiterleitung sind auch Schnittstellen zum Management verteilter Anwendungen vorzusehen. Vor allem aber müssen die Komponenten mit Schnittstellen zur Komponentensteuerung versehen werden. Bei der Spezifikation der Steuerschnittstellen müssen sowohl daten– als auch gerätespezifische Operationen berücksichtigt werden. Steuerschnittstellen, die diesen Voraussetzungen genügen, sind bisher in keinem anderen Ansatz vorgestellt worden.

Es sollen daher Bausteine entwickelt werden, die bei der Modellierung und Programmierung von verteilten Anwendungen, die zeitkontinuierliche Medien verwenden, eingesetzt werden können. Diese Bausteine sollen einfach und unabhängig voneinander sein, sowie sich wiederverwendbar implementieren lassen. Die Bausteinstruktur einer Anwendung soll die drei Ebenen der Multimediaanwendungsentwicklung widerspiegeln. Daher muß sowohl die Datenflußstruktur als auch die Steuerungsstruktur für die Bausteine sichtbar werden. Weiterhin ist die Integration einer Zeitsteuerung erforderlich.

Dazu wird ein Modell für die Funktionen auf kontinuierlichen Medien benötigt. Aus diesem Funktionenmodell sollen sich die Bausteine für die Anwendungen ergeben. Zur Realisierung der einzelnen Bausteine ist eine Integration der kontinuierlichen Medien in die Programmierwelt erforderlich. Da die kontinuierlichen Medien durch ihre zeitabhängigen Ei-

genschaften wesentlich neue Semantik in den Rechner bringen, wird ein Ansatz vorgestellt, der die Semantik der kontinuierlichen Medien erfaßt und verschiedene andere Ansätze sowohl aus der Datenverarbeitung als auch der Medienarbeit berücksichtigt.

Zur Spezifikation der Bausteine wird eine allgemeine Beschreibung verteilter Anwendungen herangezogen. Die Integration in die Entwicklung allgemeiner verteilter Anwendungen ermöglicht die Erweiterung bestehender Anwendungen, stellt die Besonderheiten von multimedialen Komponenten heraus, und beugt einer Doppelentwickung, von Konzepten vor. Für die zu entwickelnden Komponenten wird ein minimalistischer Ansatz verwendet, indem nur solche Funktionen eingebracht werden, die nicht durch eine Kombination anderer Funktionen realisiert werden können. An einigen Stellen werden weitere Funktionen hinzugefügt, die die Praktikabilität im Einsatz erhöhen.

Kapitel 2 der Arbeit schafft einen Überblick über bestehende Multimediamodelle für Anwendungen, Unterstützungssysteme sowie für Multimediadaten. In Kapitel 3 wird ein Funktionenmodell für Anwendungen kontinuierlicher Medien entwickelt, das es erlaubt, den Fluß der Multimediadaten von einer Verarbeitungsfunktion zur nächsten graphisch darzustellen. Das in Kapitel 4 hergeleitete Datenmodell liefert die entscheidende Grundlage für die in Kapitel 5 entwickelten Komponenten. Diese Komponenten stellen auf Grund ihrer Schnittstellenspezifikationen funktionsorientierte Bausteine zur Integration kontinuierlicher Medien in verteilte Anwendungen dar. Kapitel 6 zieht ein Resümee aus den durchgeführten Arbeiten, gibt einen Überblick über die Implementierungen und stellt einige Punkte vor, an denen die Arbeit weitergeführt werden kann.

2 Multimediamodelle

In diesem Kapitel werden verschiedene, existierende Modelle für Multimediaanwendungen, für die Unterstützung von Multimediaanwendungen, für Komponenten von Multimediaanwendungen und für Multimediadaten vorgestellt. Um der Fülle der existierenden Modelle und deren Einsatzgebieten gerecht zu werden, werden sie in drei Ebenen eingeteilt. Bei den Datenmodellen wird neben der Untersuchung der Medienrepräsentation auch kurz auf die Synchronisation und den Datentransport eingegangen.

2.1 Funktionsmodelle und Unterstützungssysteme

Die Funktionsmodelle und Unterstützungssysteme spannen einen weiten Bogen von einfachen Modellen, die hardware-nah mit Einsteckkarten zur Steuerung von Audio– und/oder Video–Geräten mit dem Rechner arbeiten, bis zu komplexen Objektmodellen. Die Unterschiede liegen dabei sowohl im Grad der Abstraktion der Funktionen als auch in ihrer Behandlung, entweder eigenständig oder als Methode eines Objekts. Eigenständige Funktionen müssen die Abstraktionen auch auf ihre Datenmodelle beziehen; Objektmodelle beziehen die Abstraktionen dagegen auf ihre Funktionalität und die Menge ihrer Methoden.

Die Anwendungsunterstützung für multimediale Systeme kann in drei Ebenen eingeteilt werden. Die erste Ebene stellt die Grundvoraussetzungen durch Hardwareintegration mit der Digitalisierung der kontinuierlichen Medien zur Verfügung. Die zweite Ebene kennzeichnet die Programmierunterstützung von Multimediaanwendungen durch Funktionsabstraktion in Quellen, Filter und Senken. Die dritte Ebene bilden Modelle zur Entwicklung verteilter Multimediaanwendungen, sie stellen Anwendungskonzept bereit. Da viele Multimediaanwendungen von sich aus schon verteilt sind, wie zum Beispiel eine Videokonferenz, enthalten die meisten Modelle Aspekte sowohl der zweiten als auch der dritten Ebene. Die Unterschiede liegen dabei in der Flexibilität der Entwicklung der verteilten Anwendungen.

2.1.1 Erste und zweite Ebene

Die erste Ebene basiert auf Einsteckplatinen mit realisierten Gerätetreibern und einer Bibliothek von Treiberaufrufen auf Betriebssystemebene, die für jede Karte spezifisch sind. Anfangs wurden auf dieser Ebene die kontinuierlichen Medien analog verarbeitet, später setzte sich die Digitalisierung durch. Beispiele sind das Einblenden einer Videoszene in den Bildschirm oder das Abspielen von Audio–CD's über den im Rechner eingebauten CD–Player. Mit der Verfügbarkeit der Realzeitdigitalisierung können die Mediendaten im Rechner gespeichert und verarbeitet werden. Systeme dieser Art verwenden häufig proprietäre Datenformate, so daß eine Nutzung oder Portierung auf das Vorhandensein der entsprechenden Hard– und Software angewiesen ist. Beispiele für diese Systeme sind DVI [Ripley89] [Harney91] [Stein93a], CD-I [PhSo88] [Meißner91], und QuickTime [Ortiz91] [Burg93].

Da die Portabilität der auf erster Ebene entwickelten Software unzureichend ist, wurden Betriebssystemerweiterungen auf der Basis von Funktionssammlungen oder Toolkits entwickelt. Diese stellen mit ihren Programmiermodellen die zweite Ebene der Unterstützung

dar. Die Funktionssammlungen versuchen eine einheitliche Schnittstelle zu verschiedenen Geräten zu liefern. Sie bilden damit einen bottom–up Ansatz, der von Spezialfunktionen und Eigenheiten der Hardware und der Treiber abstrahiert. Entsprechende Erweiterungen wurden im Rahmen von Anwendungssoftware und Produkten auch für die Systeme der ersten Ebene entwickelt.

In [SteiFri91] werden verschiedene Abstraktionen zur Programmierung kontinuierlicher Medien untersucht. Grundlage hierfür bilden Abstraktionsmechanismen, die allgemein in Programmiersprachen verfügbar sind. Mittels eines objektorientierten Ansatzes werden in [AndCh91] die Funktionen gleichartiger Geräte zu C^{++}-Klassen zusammengefaßt, die so Abstraktionen für Mikrophone, Lautsprecher, Dateien für kontinuierliche Medien und Audiomixer beschreiben. Darüber hinaus werden als Programmiermodell Klassen zur Verwaltung von Benutzerprogrammen für Quellen, Senken oder Filter bereitgestellt. Jedes dieser Objekte, die hier gemeinsam als CM–Node bezeichnet werden, wird mit einer Schnittstelle ausgestattet, die vier Aufgaben hat. Sie bestimmt die geforderte Verzögerung, die Datenrepräsentation, die Nachrichtenmenge an der Schnittstelle und den Verbindungsaufbau. Das System COMET [AndCh91], in dem dieser Ansatz verwendet wird, kann verteilte Anwendungen konfigurieren und damit weit mehr, als auf der zweiten Ebene definiert wird. Daher wird die Anwendungsbeschreibung mit COMET im nächsten Abschnitt behandelt.

Ein Multimediaprogrammiermodell wird in [SteiSir91] vorgestellt. Audio–, Video– und Bild–Daten werden darin mit Objekten verbunden, die als Quellen, Filter oder Senken dienen. Diese Objekte können beliebig zu komplexen Objekten zusammengefaßt werden. Im Programmiermodell bilden die Objekte Komponenten, die durch Nachrichtenwarteschlangen miteinander verbunden werden. In das Modell wurden unterschiedliche Synchronisationsmechanismen eingebunden. Da die Komponenten nicht mit speziellen Datentypen korrespondieren, kann es sein, daß eine Audiokomponente, z. B. ein Filter, Videodaten erhält. Es wird in dem Modell angenommen, daß die Komponente das selbständig aus der Datenrepräsentation erkennt und die Daten in diesem Fall unverändert weitergibt. Dieses Modell erleichtert die schnelle Entwicklung von Programmprototypen, bietet aber keine direkte Unterstützung der Anwendungsentwicklung. Die Anwendungsunterstützung liegt in der Allokation der von den Komponenten benötigten Ressourcen. Speziell fehlt die Möglichkeit der Abstraktion von Details der Datenmanipulation. Weiterhin ist dieses Modell auf ein einziges System oder maximal ein LAN beschränkt.

An der Universität in Genf wurde eine Multimedia–Framework–Architektur, die Betriebssystem, Netzwerk, Dateisysteme und Benutzerprogramme einschließt, entwickelt, die in [Gibbs91] und [Gibbs92] vorgestellt wird. Die Architektur unterscheidet für die Anwendungsunterstützung die Programmierebene (framework level) von der Anwendungsebene (script level). Im folgenden wird näher auf die Programmierebene eingegangen, die Anwendungsebene wird im nächsten Abschnitt behandelt.

Auf der Programmierebene werden in der Multimedia–Framework–Architektur zwei verschiedene, aber aufeinander bezogene Objekthierarchien unterschieden. Die eine beschreibt die Multimediadaten (MediaValue) die andere (MediaObject) bezieht sich auf Geräte oder Prozesse, die Multimediadaten produzieren, transformieren oder verbrauchen [Gibbs92]. Objekte der Klasse MediaObject sind aktive Objekte im Sinne von [Tsich91], damit haben sie sowohl interne Zustände und ein bestimmtes Verhalten (Methoden) wie gewöhnliche passive Objekte, sind aber auch mit einem Prozeß assoziiert, der selbst dann laufen kann,

wenn keine Nachricht an das Objekt gesandt wurde. Jedes MediaObject kann als eine Sammlung von Ports angesehen werden. Abhängig von der Anzahl und den Ein-Ausgabe–Eigenschaften ihrer Ports stellen MediaObjects Quellen, Filter oder Senken dar [Gibbs91].

Eine aktives Objekt ist in genau einem von den drei Zuständen IDLE, RUNNING oder SUSPENDED. Die Zustandsübergänge werden durch die Methoden START, STOP, PAUSE und RESUME angestoßen. Es ergeben sich bei der Verbindung zweier oder mehrerer solcher MediaObjects Probleme, falls nicht garantiert wird, daß die gleichen Methoden gleichzeitig bei allen verbundenen Geräten ausgeführt werden.

Jedes MediaObject beinhaltet zwei Zeitkoordinatensysteme, eins in einer Weltzeit und eins in seiner Objektzeit. Der Ursprung der Weltzeit wird von der Anwendung zu deren Anfang gesetzt, der Ursprung der Objektzeit wird vom MediaObject festgelegt; die Laufrichtung der Zeitachsen kann entgegengesetzt sein [Gibbs91]. Beschreibt ein MediaObject eine Quelle oder eine Senke in Form eines Speichers, hat die Objektzeit dort allerdings keinen Sinn.

Allen Programmiermodellen auf der zweiten Ebene ist gemeinsam, daß sie geeignete Abstraktionen mit Gruppen von Funktionen für Geräte, wie Mikrophon, Kamera oder Lautsprecher, anbieten. Diese Abstraktionen erreichen verschiedene Stufen und sind "von unten nach oben" entwickelt. Daneben wird einheitlich zur Programmstrukturierung ein Datenflußmodell bestehend aus Quelle, Filter und Senke vorgestellt. Aus diesen Ansätzen geht weiterhin hervor, daß es von Vorteil ist, parallel zur Geräteabstraktion ein korrespondierendes abstraktes Datenmodell anzubieten.

Der Nachteil, der dem Toolkit–Ansatz anhaftet, ist, daß es sich nur um Funktionensammlungen einzelner Geräte oder Gerätegruppen handelt. Zwischen den verschiedenen Geräten können auf dieser Ebene keine Beziehungen angegeben werden, sondern diese werden explizit programmiert. Zusätzlich muß die Struktur der verarbeiteten Daten vom Aufrufer zur Funktion passend bereitgestellt werden. Diese Funktionensammlungen unterstützen kein Bausteinprinzip mit klar definierten Schnittstellen sowie freier Verbindbarkeit der Geräte und bieten so außer der Strukturierung keine weitere Erleichterung für eine Verteilung der Anwendung.

Die Erweiterung des Toolkit–Ansatzes um aktive Komponenten, wie Quellen, Senken und Filter, weist schon auf Bausteine für die Verteilung hin, liefert aber hier weder eine Verbindungsstruktur noch eine Anwendungsmodellierung. Dagegen ist die damit verbundene Einführung abstrakter Geräte ein wichtiger Schritt für den Top–Down–Entwurf der Anwendung, der sich aus diesem Ansatz ableitet. Eine Erweiterung der Systeme, die auf der Toolkit–Ebene liegen, um Elemente der Anwendungsmodellierung, wie sie in [AndCh91] und [Gibbs91] [Gibbs92] vorgenommen wird, zeigt, daß die auf der zweiten Ebene noch fehlende Anwendungsunterstützung erst auf einer anderen Abstraktionsebene angeboten werden kann.

Sowohl auf der ersten als auch auf der zweiten Ebene wurden Anwendungssysteme mit hoch elaborierten Benutzerschnittstellen entwickelt [Burg93] [Stein93a]. Diese haben vor allem durch die Verwendung kompatibler Hardware weite Verbreitung gefunden. Viele dieser Systeme sind jedoch auf eine spezielle Anwendungsklasse oder Hardware ausgerichtet.

2.1.2 Dritte Ebene: Anwendungsmodellierung

Die dritte Ebene der Unterstützung liefern verschiedene Modelle, die von verteilten Multimediaanwendungen ausgehen. Sie können in zwei Gruppen eingeteilt werden. Die eine Gruppe bilden die verteilten Multimediamodelle, die für eine bestimmte Anwendungsklasse entwickelt wurden. Diese Anwendungsklassen sind zum Beispiel: Videokonferenz; Lehr– und Lernsysteme; Autorensysteme; verschiedene Lesersysteme wie Video-on-Demand oder Hypermediasysteme, elektronische Zeitung. Einen guten Überblick über die Anwendungsgebiete und die dort entwickelten Systeme bietet [WiBla94].

Die andere Gruppe bilden die verteilten Multimediamodelle, die allgemeine verteilte Multimediasysteme beschreiben. Dabei überwiegt der datenflußorientierte Ansatz in den Beschreibungsmethoden mit der Einführung von Quellen–, Senken–, und Filterkomponenten. In [BlaCou92] und [FreyIn91] werden dagegen datenflußorientiert Geräte und Medienströme modelliert.

Das Anwendungsmodell COMET [AndCh91] besteht aus einem Graphen, dessen Knoten eine Quelle oder eine Senke, oder beides in einem, für Multimediadaten darstellen. Jeder Knoten hat einen oder mehrere Ports zum Verbinden der Knoten untereinander. COMET berechnet durch Interpretation des Graphen eine feste Anwendung in einer Art Kompiliervorgang. Dabei werden zeitliche Restriktionen, unterschiedliche Datentypen an den Schnittstellen, Nachrichtenlängen und der explizite Verbindungsaufbau berücksichtigt. Dies setzt eine zutreffende zentrale und globale Sicht auf alle Ressourcen voraus, die lediglich in kleinen verteilten Umgebungen, wie einem Workstation–Cluster, gegeben ist. Die einmal erstellte Anwendung ist sowohl in ihren Komponenten als auch in der Verteilung und in den Datentypen festgelegt und kann nicht mehr verändert werden. Änderungen sind nur auf dem beschreibenden Graphen möglich und müssen nochmals neu interpretiert werden [AndCh91].

Neben den Geräten und Medienströmen, die von Multimedia–Service–Objekten verwaltet werden, gehören zum Modell von [FreyIn91] auch MediaViews. Diese enthalten eine Benutzerschnittstelle und definieren im Gegensatz zu den übrigen Ansätzen damit neben den Datenschnittstellen eine zweite Klassen von Schnittstelle in der Multimediaanwendung. Allerdings wird weder eine Beziehung zwischen den Schnittstellen noch eine Anwendungsmodellierung beschrieben.

In [Baker92] stellen die Autoren gleichzeitig auf verschiedenen Abstraktionsebenen Modelle für Funktionen und Daten untereinander vermischt vor. Auf der untersten Ebene kennen sie *Streams*, die Daten repräsentieren, und *Filter* und Verteiler (*switches*), durch die die Daten geleitet werden. Das ist insofern interessant, als hier zum erstenmal deutlich zwischen diesen beiden Funktionen (*filter vs. switch*) unterschieden wird. Allerdings werden weder Quellen noch Senken von Streams definiert. Eine andere Erweiterung wird dagegen in [SteHaHo92] vorgenommen. Dort werden Abstraktionen für Mischen und Trennen zu den Prozeßstationen *digital data source, digital data sink, conversion* und *transport* hinzugenommen.

Auf der Präsentationsebene werden in [Baker92] verschiedene Abstraktionen und Mechanismen unterschieden. Dazu gehören ein logisches Zeitbezugssystem mit einer Synchronisationsbeschreibung, eine Mischfunktion (*mixer)* und eine dreistufige Hierarchie (track, chanel, presentation) von Datenabstraktionen, die eine mehrfache Enthaltensbeziehung beschreiben. So enthält eine *Presentation* mehrere *Chanel*, welche wiederum jeweils mehrere

Tracks enthalten können. Das darüber liegende Hyperpresentation–Layer beschreibt den Präsentationsablauf in einer Art Hypertextstruktur. Eine Anwendungsmodellierung beschreiben die Autoren nicht.

Die Touring Machine [ArBaGo92] stellt ein Softwaresystem dar, das die Verbindung von Audio– und Video–Geräten über verschiedene Verteiler und spezialisierte Hardware steuert. Das System unterstützt eine Vielzahl von Anwendungen. Besonders in den ersten Versionen [BateSega90], [AraClay90], war dieser Ansatz vom Hintergrund der Telekommunikationsgesellschaften geprägt, was sich insbesondere in der strikten Unterscheidung zwischen den Ressourcen der Netzwerkbetreiber (call object) und den Ressourcen der Benutzer (station object) zeigt [RustBCD91]. Dieses Modell wurde um eine Programmieroberfläche erweitert, die als Abstraktionen *client, session, connector, endpoint, port* und *mapping* kennt. Damit bleibt das Modell eng an die Telekommunikationsdienste angelehnt.

An der Universität von Lancaster wurde eine umfassende Plattform für offene verteilte Multimediaanwendungen entwickelt. Die wesentlichen Abstraktionen in diesem Ansatz sind *Geräte, Endpunkte* von Verbindungen und *streams*. Dazu kommt ein umfassender Synchronisationsmanager [BlaCou92]. Somit teilt sich der Ansatz in Basisdienste, die der Verteilung der Multimediaanwendungen dienen, und eine zusätzliche Unterstützung für die Synchronisation. Zu den Basisdiensten wurde ein Konfigurationsdienst entwickelt, der der Erstellung von verteilten Anwendungen dient [Coulson93]. Dieser Ansatz ist insofern hervorzuheben, als er deutlich macht, daß eine Plattform für offene verteilte Multimediaanwendungen im Zusammenhang mit allgemeinen offenen verteilten System behandelt werden soll [CoulBla93]. Der Vorteil diese Ansatzes liegt darin, daß bei der Erzeugung der Multimediaanwendung deren gesamten Charakteristika berücksichtigt werden. Daraus resultiert aber gleichzeitig auch der Nachteil einer hohen Komplexität, da unterschiedliche Abstraktionsebenen gemeinsam beschrieben werden.

Die Multimedia–Framework–Architektur aus [Gibbs91] und [Gibbs92] wurde schon ausführlich beschrieben. Neben den bereits vorgestellten Konzepten der Programmierunterstützung, wurden auch verschiedene Konzepte der Anwendungsmodellierung entwickelt. In [Gibbs91] wird ein *scripting level* beschrieben, das es ermöglicht verschiedene Funktionalitäten oder MediaObjects zusammenzufassen. Ein *script* definiert eine Gruppe von MediaObjects, die miteinander verbunden werden können. Dazu enthält ein *script* Bedingungen über die Typen von MediaObject, die verbunden werden dürfen. Ein *script* wird durch eine Script-Sprache spezifiziert. Im Script-Modell von [Gibbs91] werden folgende Elemente unterschieden: Multimediahierarchien über MediaValue und MediaObjects, Verbindungstypen, Ports, Objektschnittstellen, Mitgliedschafts– und Konfigurationsbedingungen. Eine andere Unterstützung der Anwendungskonfiguration der Multimedia–Framework–Architektur wird in [Gibbs92] und [MeyGib93] vorgestellt. Dabei handelt es sich um ein Graphenmodell, in dem die Knoten die MediaObjects oder Komponenten, die Kanten deren Verbindungen repräsentieren.

Bei den Systemen auf der dritten Ebene stehen die anwendungsbezogenen Funktionalitäten im Vordergrund, die eine Grundlage für die Verteilung bilden können. Durch die Möglichkeit diese Funktionen hierarchisch zu speziellen Geräten zu verfeinern, wird hier ein top-down Entwurf unterstützt. Modelle auf dieser Ebene beschreiben das Zusammenspiel der einzelnen Anwendungselemente. Gemeinsam ist diesen Modellen weiterhin, daß sie für abgeschlossene Multimediaanwendungen entwickelt wurden und keinen Bezug zu anderen

verteilten Anwendungen herstellen oder anbieten. In ihrer Untersuchung der Systemunterstützung für Multimediaanwendungen kommen die Autoren in [BlaCoDa94] zu dem Ergebnis, daß zwar einige Plattformen für verteilte Multimediasysteme existieren, aber noch keine Übereinstimmung in den Abstraktionen für die kontinuierlichen Medien bestehen. Weiterhin bieten die existierenden Plattformen nicht die für verteilte Anwendungen benötigte Unterstützungsleistung.

Ein etwas anderer Ansatz als die bisher auf der dritten Ebene beschriebenen wird in [Blako93] gewählt. Einerseits wird eine abstrakte Medienbeschreibung mit einer Einteilung der Medien in Informations–, Transport– und Darstellungsobjekte vorgestellt, andererseits wird als zentrales Planungs–, Ausführungs– und Prüfmittel ein Flußgraphenmodell definiert, das Synchronisationsspezifikationen, und Beschreibungen für die Anwendungsmedien, die Arbeitsstationen und das Netzwerkmodell zur Durchführung seiner Aufgaben heranzieht. Durch die expliziten Transportobjekte verliert die Anwendung ihre Verteilungstransparenz. Die Beschreibung durch den Flußgraphen verdeckt die möglichen Benutzerinteraktionen, die nur im Medienmodell spezifiziert werden können.

Insgesamt bieten alle Modelle auf der dritten Ebene Grundlagen für die Verteilung von Multimediaanwendungen durch die Einführung von Komponenten deren Verbindung. ‚Im allgemeinen werden aber nur die Datenverbindungen beschrieben; die Steuerung der Komponenten oder Komponentengruppen ist dagegen für sich abgeschlossen, ohne einen Bezug zu allgemeinen verteilten Anwendungen und kann so die Integration von kontinuierlichen Medien dort nicht leisten.

2.1.3 Beispiele für Entwicklungsmodelle verteilter Multimediasysteme

Die Anwendungsmodellierung der dritten Ebene bezog sich jeweils auf die Elemente der darunterliegenden Programmierebene. Dabei wurde zwar die Einteilung in Quellen–, Filter– und Senkenfunktionalität genutzt, eine Anwendungsentwicklung auf einer abstrakten Stufe mit nachfolgenden Verfeinerungen aber nicht unterstützt. Modelle die dieses Ziel verfolgen, werden Entwicklungsmodelle genannt. Dabei stehen allgemeine Multimediakomponenten im Vordergrund, die zu einer Anwendungsstruktur verknüpft werden, bevor ihre Funktionalität verfeinert wird. Dieser Ansatz wird in den im folgenden vorgestellten Arbeiten verfolgt.

Im Y–System [Popescu91], das mit dem Projekt BERKOM [BERKOM91] [BERKOM92] verbunden ist, wird ausgehend von der Modellierung verteilter Anwendungen, multimediale Funktionalität integriert. Hierbei ist hervorzuheben, daß auf der Anwendungsebene die Interaktionen zwischen den Anwendungsobjekten in zwei Phasen ablaufen. In der ersten Phase werden die Bindungen zwischen den Anwendungsobjekten realisiert, in der zweiten Phase erst die Operationen der Anwendungsobjekte ausgeführt. Der Schwerpunkt der Integration multimedialer Funktionalität liegt aber im Bereich der Kommunikation [ButHeLu91]. Darauf aufbauend wurden verschiedene spezielle, in sich abgeschlossene Multimediadienste wie der BERKOM *Multimedia Mail Service* [Blum93] und der BERKOM *Multimedia Collaboration Service* [BERMMC93] entwickelt.

Grundlegende Arbeiten für die Entwicklung verteilter Multimediasysteme wurden von der *Multimedia and Hypermedia information coding Expert Group* (*MHEG*) vorgelegt [MHEG90]. Dort wird gefordert, daß der Begriff Medium, nicht ohne genauere Präzisierung gebraucht werden sollte, da er zu unterschiedliche Verwendungen hat. Es werden fünf verschiedene Medien unterschieden:

1. Perzeptionsmedium,
   Information, wie sie von einer Person natürlich aufgenommen wird.
   (Sprache, Geräusch, Musik, Text, Zeichnung, Film ...)
2. Repräsentationsmedium,
   Kodierung, in der die Information in einem Datentyp, enthalten ist.
   (ASCII, EBCDIC, CGM, JPEG, MPEG, ...)
3. Präsentationsmedium,
   Mittel, mit dem Information zur Eingabe oder Ausgabe dargestellt wird.
   (Tastatur, Maus, Mikrophon, Kamera, ...)
   (Bildschirm, Drucker, Lautsprecher, ...)
4. Speichermedium,
   das physikalische Mittel, das die Information speichert.
   (Speicherbausteine, Festplatte, Diskette, Magnetband, CD, ...)
5. Übertragungsmedium,
   das physikalische Mittel zur Datenübertragung.
   (Kupferadern, Koaxialkabel, Glasfaser, Funk, ...)

Dabei beschränkt sich MHEG im weiteren auf die Normierung der kodierten Repräsentation, die als nicht mehr änderbare Einheit ausgetauscht werden soll. Daraus ergibt sich, daß die weiteren Arbeiten von MHEG nicht als Medienmodellierungstechnik in einem Unterstützungssystem geeignet sind [Blako93].

Das Modell von [SteiSir91] wurde bereits bei den Programmiermodellen besprochen, da es keine Unterstützung für die Beschreibung der Anwendungsstruktur enthält. Es wird hier aber erwähnt, weil es zum erstenmal die generische Sichtweise auf die Komponenten in den Vordergrund stellt und deren Vorzüge herausarbeitet. Diese liegen vor allem in der Typunabhängigkeit der Komponenten von den verarbeiteten Daten.

In [FrBöBr92] werden als Abstraktionsgrundlage Funktionen auf kontinuierlichen Medien herangezogen, die in ihrer Verbindung untereinander den Datenfluß der kontinuierlichen Daten beschreiben. Zu den Funktionen auf kontinuierlichen Medien wird eine Objekthierarchie angegeben, die entsprechend der generischen Sicht von [SteiSir91] die Funktionen zu Geräten verfeinert. Komplexe Geräte lassen sich durch Mehrfachvererbung herleiten. Die in [FrBöBr92] vorgestellten Ansätze stellen den Ausgangspunkt der vorliegenden Arbeit dar.

Ein Ansatz zur Strukturierung multimedialer Anwendungen wird in [SteiMey92] vorgestellt. Ausgehend von den vorgegebenen Basisfunktionen der persistenten und nicht persistenten Erzeugung, der Präsentation und der Speicherung von Multimediadaten werden verschiedene Funktionseinheiten (FU) entwickelt, die die Struktur multimedialer Anwendungen beschreiben können. Durch die Kombination der Datenquellen und –senken ergeben sich 9 verschiedene Funktionseinheiten.

Quellen / Senken	Perzeption	Lesen	Perzeption und Lesen
Präsentation	Funktionseinheit 1	Funktionseinheit 3	Funktionseinheit 6
Schreiben	Funktionseinheit 2	Funktionseinheit 4	Funktionseinheit 7
Präsentation und Schreiben	Funktionseinheit 5	Funktionseinheit 8	Funktionseinheit 9

Tab. 1 *Funktionseinheiten nach [SteiMey92]*

Mittels dieser Funktionseinheiten läßt sich der Datenfluß in Multimediaanwendungen beschreiben. Das Modell liefert genau die Anwendungskonfiguration für kontinuierliche Medien, beschreibt aber lediglich einzelne Elemente und deren Zusammensetzung zu ausgewählten, repräsentativen Anwendungstypen. Es eignet sich daher gut für die Repräsentation deren Grundstruktur. Wie in [Börger92] gezeigt wird, unterstützen die Funktionseinheiten aber nicht die Anwendungsentwicklung, da sie keine Kombinationsfunktion untereinander anbieten. Eine Kombination von Funktionseinheiten oder auch eine Spezialisierung einzelner Funktionen erfordert ein Aufbrechen deren internen Struktur. Insbesondere bieten sie so keine Möglichkeit für eine generische, objektorientierte Erzeugung multimedialer Anwendungen.

Das *Cinema*–Modell [BarDer93] geht von einer abstrakten Komponente zur Manipulation multimedialer Daten aus. Im Vordergrund steht die Trennung der Funktionen zur Bearbeitung multimedialer Daten von Problemen der Ressourcenverwaltung, der Transportmechanismen und von zeitlichen Aspekten. Allerdings wird zunächst kein Anwendungsmodell entwickelt, sondern es wird gezeigt, wie sich die verschiedenen notwendigen Interaktionen von Komponenten unterschiedlichen Schnittstellen zuordnen lassen. Die abstrakte Komponente kann zu Komponenten spezieller Funktionalität verfeinert werden. Durch Verschachtelung können aus bestehenden Komponenten neue Komponenten mit komplexerer Funktionalität gebildet werden. Die komplexen Komponenten beschreiben dann einen Teil der Anwendungsstruktur. Die gesamte Anwendungsstruktur entsteht erst beim Verbinden der einzelnen Komponenten. Beispielhafte Anwendungskonfigurationen werden nicht vorgestellt.

Die einfachsten Komponenten werden in [BarDer93] Basiskomponenten genannt. Eine Basiskomponenten hat Schnittstellen zu anderen Basiskomponenten sowie zur Ressourcen–Verwaltung. Es existieren dedizierte Schnittstellen für Datenempfang und -Übertragung (DATA), Ressourcen–Reservierung (SRP), Ressourcen–Anforderung (RM), für die Dienstgüteanforderungen an die Komponente (QoS), die Ereignissverarbeitung (Event) und für das Konfigurationsmanagement (Config).

Anwendungen spezifizieren die gewünschte Funktionalität und Kommunikationstopologie, indem sie Funktionalitätskomponenten zu "End-to-End" Kommunikationspfaden verknüpfen. Alle Interaktionen der Komponenten werden erfaßt, wobei Anwendungsfunktionalität und Managementfunktionen über sechs verschiedenen Schnittstellen verteilt sind.

Der *Cinema*–Ansatz wurde parallel zur vorliegenden Arbeit aber davon unabhängig um einige Elemente erweitert. So wird in [RoBaHe94] die Beschreibung einer Komponente auf zwei Datenports, eine Komponentensteuerung und einen Zeitbezug umgestellt. Für

Komponenten wird eine Unterscheidung in Quellen, Senken und Zwischenkomponenten vorgenommen. In der vorgestellten objektorientierten Beschreibung der Zwischenkomponenten kann nur eine Schnittstelle sichtbar gemacht werden. In diesem Fall ist das die Steuerschnittstelle, die aber in [RoBaHe94] nicht weiter betrachtet wird. Die Komponenten in diesem Modell können hierarchisch zu komplexen Komponenten zusammengesetzt werden. Eine Synchronisation zwischen den Operationen der Elemente einer zusammengesetzten Komponente wird über die Integration eines Uhrenmodells erreicht.

In [Helbig94] wird eine Zusammenfassung der Konzepte von *Cinema* vorgestellt. Darin werden als Hauptthemen für *Cinema* die Reservierung von Ressourcen und das Konfigurationsmanagement sowie die Synchronisation mittels Uhren herausgestellt. Das dabei vorgestellte Modell einer Uhrenhierarchie wird in [RotHelb96] eingehend erläutert. Die Uhrenhierarchie in *Cinema* wird im Abschnitt über die Zeitmodelle gemeinsam mit anderen, verwandten Modellen beschrieben. Die Implementierung multimedialer Systemdienste in *Cinema* wird in [BaHeRo95] erläutert.

Cinema liefert eine komplette Anwendungsentwicklungsumgebung mit und bietet damit ein geeignetes Werkzeuge zur Entwicklung verteilter multimedialer Anwendungen. In der Idee stellt *Cinema* eine Parallelentwicklung zur vorliegenden Arbeit dar. Somit ergeben sich viele Gemeinsamkeiten. Im Bereich der Zielsetzung unterscheidet sich *Cinema* durch seine Entwicklungsorientierung gegenüber der Modellierungssicht der vorliegenden Arbeit. Zu dieser Modellierungssicht gehört auch die Entwicklung eines Datenmodells, das für *Cinema* nicht zwingend erforderlich ist.

Die Modellierungssicht ist auch in den Arbeiten der *Interactive Multimedia Association* (IMA) ein zentraler Punkt. In ihrem Architekturreferenzmodell werden verschiedene Aufgaben von Multimediasystemen definiert und drei Schichten zugeordnet [IMA92]. Diese Referenzmodell soll als Brücke zu einer plattformübergreifenden Kompatibilität dienen. Es beschreibt, was alles zu unternehmen ist, und dient damit auch der Bewertung der Vollständigkeit eines Ansatzes, der plattformübergreifend verwendbar sein soll. Es fordert die Trennung der Daten von der Steuerung, des Daten- vom Dateiformat, des Daten- oder Dateiformats von Geräten und die Trennung von Anwendungsanforderungen von Benutzeranforderungen und von Systemanforderungen.

Das IMA Architekturreferenzmodell kennt drei Schichten der Plattform-zu-Plattform Kommunikation. *Content* ist der Name für die digitale Datenstromschnittstelle; darunter liegt die Anwendungskompatibilitätsschicht, darunter die Systemkompatibilitätsschicht, die die eigentlichen Interoperabilitätsdienste bereitstellt, die darunterliegende Hardwareschicht ist die Schnittstelle zu den Geräten, Platten, zu CD-ROM oder Netzwerk.

Die Anwendungskompatibilitätsschicht behandelt Programmierschnittstellen, Skriptsprachen oder spezialisierte Dienste wie zum Beispiel für Videokonferenzen; diese Schicht beschreibt somit auch die Schnittstellen einer Anwendungsentwicklungsumgebung. Die Systemkompatibilitätsschicht soll die Interoperabilität zwischen verschiedenen Anwendungen ermöglichen. Dafür stellt sie ein gemeinsames Protokoll zur Anwendungssteuerung von Multimediadatentypen und zur Gerätesteuerung sowie zur Synchronisation dieser Datentypen bei der Darstellung beim Benutzer zur Verfügung. Diese Anforderungen an die Schichten müssen von entsprechenden Systemen realisiert werden.

Die Systemkompatibilitätsschicht wurde von der IMA als *Multimedia System Services* (MSSRP) in einem *Recommended Practice* näher spezifiziert [IMA94]. Mit dem MSSRP hat die IMA eine Infrastruktur zum Aufbau von Multimediaplattformen entwickelt, die interaktive Multimediaanwendungen in heterogenen, verteilten Umgebungen ermöglichen sollen. Diese Systemdienste stellen den Entwicklungsplattformen einheitliche Schnittstellen bereit. Schnittstellen gibt es sowohl für Geräte als auch für synchronisierte, zeitabhängige Medien.

Zeitabhängige Medien werden im MSSRP als Medienströme übertragen. Aufeinander bezogene Medienströme können zu Gruppen zusammengefaßt werden; es sind Gruppen von Gruppen möglich. Die Gruppen stellen ein mächtiges Werkzeug zur Steuerung von Medienströmen dar, beinhalten aber keine Synchronisation.

Im MSSRP werden sehr eingehend die Vorgänge zur Erzeugung, Nutzung und Vernichtung von virtuellen Geräten und Verbindungen auf Systemseite betrachtet. Bei der Definition der virtuellen Geräte werden Gerätehierarchien nur indirekt auf Schnittstellenhierarchien abgebildet. Die Abbildung auf Schnittstellenhierarchien ist aber Voraussetzung zur Entwicklung von komplexen, zusammengesetzten Geräten oder Komponenten. Somit kann die Systemunterstützung nur einfache Geräte anbieten. Weiterhin verbindet die Definition der Geräte des MSSRP Daten und Steuerung miteinander auf einer recht hohen Ebene. Es werden da zum Beispiel Audio und Video direkt in die Gerätesteuerungshierarchie aufgenommen.

Insgesamt betrachtet bietet die IMA mit dem MSSRP eine Systemschnittstelle an, auf die Unterstützungssysteme zur Anwendungsentwicklung aufsetzten können. Damit werden Anwendungsentwicklungssysteme davon entbunden, Geräteallokation, Objektlebenszeiten, Datenübergabe an Systemschnittstellen oder ähnliches zu verwalten. Eine Unterstützung der Anwendungsentwicklung direkt wird von der IMA im MSSRP nicht geleistet. Damit bildet der MSSRP eine Art erweitertes Toolkit und somit zum Beispiel auch eine geeignete Grundlage für die Realisierung der in dieser Arbeit vorgestellten Konzepte zur Anwendungsentwicklung.

Ein weiteres Modell für verteilte multimediale Anwendungen mit dem Schwerpunkt auf der Entwicklungs- und Laufzeitumgebung bietet [KaHeSch95]. Die Anwendungen werden objektorientiert modelliert, wobei ein *request-broker* den Zugriff zu den Anwendungsteilen vermittelt. Ähnlich wie in der vorliegenden Arbeit werden Anwendungsteile durch Schnittstellen spezifiziert, die in einer eigenen Sprache, der Interface Definition Language (IDL), erstellt werden. Andere, den vorgestellten Arbeiten ähnliche Konzepte, werden in [Eventor94], [BatBac94] [MEDUSA94] [Tenn94] vorgestellt. Diese Ansätze stellen jeweils nur einen bestimmten Aspekt in den Vordergrund.

2.1.4 Anforderungskatalog an Bausteine für kontinuierliche Medien

Aus den untersuchten Ansätzen zur Anwendungsunterstützung läßt sich ein Anforderungskatalog an die Bausteine für kontinuierliche Medien zusammenstellen:

- Integrierbarkeit in beliebige Anwendungen durch explizite Schnittstellenspezifikation auf Anwendungsebene, das entspricht der Anwendungskompatibilitätsschicht von IMA.
- Anwendungsgliederung entsprechend den folgenden Gesichtspunkten:
  - Multimediadatenfluß (z. B. [SteiMey92])
  - Steuerungshierarchie der Komponenten

  - Zeitbezug der Komponenten
  - Bindungsstruktur aller Komponenten
- Bausteinstruktur
  - generische Komponenten [SteiSir91] [FrBöBr92]
  - funktionsorientierte Schnittstellen
  - Trennung von Management, Daten und Steuerung

Die überwiegende Zahl der bisherigen Ansätze zur Anwendungsunterstützung hat sich, wie oben dargelegt, nur mit Teilen dieses Katalogs befaßt. Zusammengefaßt ergeben sich als Kritikpunkte an diesen Ansätzen die Vermischung von Anwendungsfunktionen und Realisierungserfordernissen, indem Komponentenfunktionalität, Synchronisationsrealisierung und Ressourcenbelegung auf einer Ebene und zum Teil miteinander vermischt beschrieben werden. Weiterhin liegt ein uneinheitlicher Komponentenbegriff vor, bei dem verschiedene Aspekte der Geräte, Gerätetreiber und der Betriebssystemressourcen mit der Komponente als Element der Verteilung gleichgesetzt werden. Am weitesten entwickelt ist *Cinema*, das fast alle Punkte des Anforderungskatalogs erfüllt. Wie auch vielen anderen Ansätzen fehlt ein Bezug zu einem abstrakten Datenmodell.

In die vorliegende Arbeit sind aus den beschriebenen Ansätzen verschiedene Aspekte eingeflossen. Eine besondere Stellung nehmen dabei die beschriebenen Vorarbeiten ein, sowie die Anwendungsmodelle von [SteiMey92], die eine Grundlage für den Nachweis der Vollständigkeit der Anwendungsmodellierung bieten.

2.2 Datenmodelle

Mit der Integration der kontinuierlichen Medien in den Rechner wurden verschiedene Datenmodelle entwickelt. Ein Teil dieser Modelle bezieht sich auf Dokumentarchitekturen zum Beispiel HyTime (Hypermedia and time–based Structuring Language) auf SGML (Standard Generalized Markup Language) [HyTime91] [Gold91] [Gold91a], oder verschiedene Erweiterungen, die zur *Open Document Architecture (ODA)* [ISO8613] vorgeschlagen werden [Stein93a]. Ein anderer Teil bezieht sich auf physikalische Speicherung oder günstige Übertragungseigenschaften und befaßt sich daher mit Komprimierung und Dekomprimierung von Daten. Aus diesen Modellen haben sich verschiedene Kodierungen entwickelt, die zum Teil standardisiert wurden, zum Teil de facto Standards sind. Auf einige dieser Kodierungen wird im folgenden näher eingegangen.

Ein dritter Teil der Arbeiten beschäftigt sich mit der Speicherung in Datenbanken. Dabei wurde als Modellierungsmittel für die kontinuierlichen Daten ein BLOB (binary large object) eingeführt [CarDeWi89], [LehLind89], [Rengar92]. Obwohl die Verwaltung der sehr großen Datenwerte von kontinuierliche Medien eine Voraussetzung für Multimediadatenbanken ist, ist sie dafür aber nicht ausreichend [Gibbs94]. Das Datenbanksystem sollte in der Lage sein, diese Daten zu interpretieren, oder zumindest ihre Struktur kennen. Daher wird ein Datenmodell benötigt, das die Semantik der kontinuierlichen Medien beschreibt. Die Forschung auf diesem Gebiet ist noch neu, erste Ergebnisse werden in [Gibbs94] und [Fritzsche95] dargestellt.

2.2.1 Kodierungen kontinuierlicher Medien

Audiodaten werden meist direkt durch die von der PCM–Technik definierten Samples dargestellt. Eine Kompression der Samples ist durch die μ-law-Kodierung möglich [Skritek88]. Eine neuere Darstellung ist die Spektraldarstellung, wie sie bei der MPEG-Audiokodierung [MPEG93] verwendet wird. Die Audiokodierungen werden hierbei mit den Videokodierungen gemeinsam beschrieben.

Ein eigener Ansatz im Audiobereich ist der MIDI–Standard (Musical Instrument Digital Interface). Dieser repräsentiert ausschließlich Musik und ist ein Protokoll zur Kommunikation zwischen elektronischen Musikinstrumenten [Stein93a] [Hesme93]. Im Gegensatz zu den bisher angesprochenen Verfahren wird hier nicht direkt eine meßbare physikalische Größe gespeichert, sondern auf eine abstrakte Beschreibung zurückgegriffen. Statt die Schwingung eines Tons anzugeben, wird seine Nummer auf der Klaviatur verwendet. Dazu kommen die Anschlaghärte statt der Amplitude und die Dauer des Tons oder äquivalent dazu seinen Anfang und sein Ende. Eine Folge dieser Werte wird als Kanal bezeichnet. Die Zuordnung eines Kanals zu einem bestimmten Klang wird durch das Ausgabegerät realisiert. Es hat für jede Klangfarbe einen Satz von Samplefolgen, aus welchen mittels einer Funktion über die Parameter Nummer und Anschlaghärte die zu spielenden Samplefolgen berechnet werden. Die Klangqualität wird durch die Anzahl der verschiedenen Samplefolgen und die Anschlagsfunktionen des Ausgabegerätes bestimmt. Für MIDI–Daten kann es daher keine originalen Klänge geben. Aus Sicht der Anwendung wird für die Datenbeschreibung eine andere Abstraktionsebene als die der technischen Samples benutzt. Das MIDI–Beispiel zeigt somit, daß die Benutzer von Multimediadaten weniger mit der technischen Repräsentation der Daten arbeiten, als vielmehr mit deren abstrakter Beschreibung.

Ein Einzelbildkodierungsstandard ist JPEG (Joint Photographic Experts Group) [JPEG92]. JPEG ist für farbige und grauskalierte Standbilder entwickelt worden, läßt sich aber auch über geeignete Hardwareunterstützung auf Bewegtbildsequenzen anwenden [Stein93a]. Dabei werden in jedem einzelnen Bild Redundanzen reduziert, so daß jedes Bild einer Bildfolge einzeln für sich wiederhergestellt und interpretiert werden kann. Bei der Wiederherstellung ist die verlustfreie Kodierung von verlustbehafteter Kodierung zu unterscheiden. Die Qualität der mit Verlust kodierten Bilder ist von dem Verhältnis der Anzahl der in kodierter Form enthaltenen Bits und der Anzahl der im Bild enthaltenen Pixel abhängig.

Ein Bewegtbildstandard für verschiedene Datenübertragungsleistungen ist die CCITT Empfehlung H.261 *Video Codec for Audiovisual Services at p × 64 Kbit/s* [H.261]. Diese Empfehlung wurde vor dem Hintergrund von ISDN und dem Bildtelephon entwickelt. Daher kommt die Rate von 64 Kbit/s pro zu Verfügung stehendem B-Kanal im ISDN. Das Kodierungsverfahren selbst ist von den harten Zeitbedingungen des Videotelephons geprägt. Nach CCITT Anforderungen dürfen Kompression und Dekompression zusammen nicht mehr als 150 ms Signalverzögerung ausmachen. Daher werden Bildausschnitte unterschiedlicher Grösse gemeinsam kodiert, so daß pro Bildausschnitt stets die gleiche Information übertragen wird. Das Verfahren wird dadurch effizient, daß es von einem Bild zum nächsten nur die Differenzen in den Bildausschnitten überträgt. Falls sich in einem Bildausschnitt wenig verändert (bewegt), können nach wenigen Bildern sehr feine Strukturen dargestellt werden; bei extrem vielen Veränderungen (viel Bewegung im gesamten Bild) zerfällt das Bild vorüber-

gehend in einfarbig bunte Rechtecke. Durch die geforderte gleichmäßige Datenrate und die Kodierung von Differenzen von einem Bild zum nächsten, wird daher nur eine sehr ungleichmäßige Bildqualität erreicht.

Die *Moving Pictures Experts Group* (MPEG) der ISO hat zwei Standards, MPEG I und MPEG II, zur Audio– und Video–Kodierung und Speicherung entwickelt [MPEG93]. Beide kodieren gemeinsam Bild und Ton ähnlich dem Fernsehen. Die beiden Methoden können auch unabhängig voneinander verarbeitet werden. MPEG I reduziert Video in VHS Qualität auf eine Datenrate von etwa 1Mbit/s [LeGall91]. MPEG II soll mit bis zu 10 Mbit/s etwa die Qualität von kommerziell ausgestrahltem Fernsehen erreichen. Neben der Datenreduktion für die Übertragung ist bei MPEG auch die Speicherung der Daten berücksichtigt worden. Daher werden bei der Kompression der Bilder Kompromisse über die Datenreduktion zugunsten des wahlfreien Zugriffs auf die Bilder eingegangen. Neben der speicher– und übertragungsgünstigen Kodierung der Differenzen zwischen aufeinanderfolgenden Bildern, wie sie ähnlich auch bei H.261 benutzt wird, werden daher in einzelnen Bildern nur deren eigene Redundanzen eliminiert, entsprechend dem Verfahren von JPEG.

MPEG definiert für die Speicherung ein hierarchisches Datenmodell, das als wichtige Gliederung Elemente definiert, die unabhängig von allen übrigen Daten vollständig dekodiert werden können. Bei Audio ist das die *Audio Access Unit (AAU)*, bei Video die *Group of Pictures (GOP)*. MPEG–Audiodaten gliedern sich in Frames, diese in Audio Access Units und letztere in Slots. Die Slots sind dabei keine Einteilung des Speicherzugriffs, sondern des Kodierungsverfahrens.

MPEG Videodaten gliedern sich in sechs Ebenen [MPEG93]:

1. Sequence Layer
   Eine Sequenz enthält zunächst einige Verwaltungsinformationen, die den benötigten Speicherplatz der dekodierten Sequenz sowie die Verzögerungszeit durch das Dekodieren beschreiben, sowie die Quantisierungsmatrizen für die folgenden kodierten Bilder. Anschließend enthält eine Sequenz genau eine *Group of Pictures* (GOPs) oder eine Folge von GOPs. Unmittelbar vor jeder weiteren Bildgruppe kann ein Header eingefügt werden, der neue Quantisierungsmatrizen definiert. So lassen sich große Unterschiede, zum Beispiel in den Farben, zwischen aufeinanderfolgenden Bildgruppen effizient beschreiben, und bei der Darstellung sofort im folgenden Bild realisieren.

2. Group of Pictures Layer (GOP Layer)
   Eine GOP besteht aus mehreren Einzelbildern in unterschiedlichen Kompressionsstufen, die mit I, P und B bezeichnet werden. In einer GOP muß mindestens ein I–Bild enthalten sein, dazu können beliebig viele P– und B–Bilder oder auch weitere I–Bilder folgen. Die Reihenfolge der Bilder innerhalb einer GOP verändert sich bei der Kodierung und der Dekodierung gegenüber der Darstellungsreihenfolge. Die Reihenfolge in der die Einzelbilder gespeichert werden ist damit nicht die Darstellungsreihenfolge, sondern vielmehr die für die Dekodierung benötigte Reihenfolge. Ein ausführliches Beispiel dazu findet sich in [Braun92].

3. Picture Layer
   Auf der Bildebene sind zu jedem Bild verschiedene Parameter definiert, von denen einigen noch keine semantische Bedeutung zugemessen wurde. Der wichtigste Parameter ist die Bildnummer, über den sowohl die Darstellungsreihenfolge, als auch

unter Verwendung der Bildrate ein zeitlicher Bezug definiert werden. Bezogen auf die Rot–Grün–Blau–Signale (RGB–Farben) eines Bildes werden zu jedem Bild eine Luminanz– und zwei Chrominanz–Matrizen bestimmt. Diese Matrizen bilden die Grundlage für die Kompression der Bilder. Dafür sind folgende Bildtypen definiert:

a. I–Bilder (intra–coded picture):
   Sie sind ähnlich dem JPEG Verfahren komprimiert, wobei nur Farbdaten herausgefiltert werden, die das menschliche Auge nicht wahrnehmen kann. Die Redundanzreduktion bezieht sich nur auf die eigenen Bildelemente. Damit kann zwar nur ein relativ geringer Kompressionsgrad erreicht werden, dafür stellen die I–Bilder aber Zugriffspunkte für den wahlfreien Speicherzugriff dar. Falls die I–Bilder periodisch im Datenstrom auftreten, können sie zur Realisierung von schnellem Suchlauf vorwärts oder rückwärts benutzt werden.
b. P–Bilder (predictive–coded picture):
   Sie sind ähnlich dem Verfahren von H.261 kodiert, indem sie abhängig von vorhergehenden Bildern nur noch die Differenzen zum aktuellen Bild kodieren. Die Referenzbilder der P–Bilder können sowohl I–Bilder als auch P–Bilder sein.
c. B–Bilder (bidirecionally predictive coded picture):
   Sie besitzen nicht nur Referenzen auf vorangehende Bilder sondern auch auf nachfolgende. Diese Kodierungsart erreicht den höchsten Kompressionsgrad, kann aber nur bei einer vollständig vorliegenden Bildfolge angewandt werden. Referenzen der B–Bilder können sowohl I– als auch P–Bilder sein. B–Bilder werden dagegen nie als Referenzen für andere Bilder verwendet.
d. D–Bilder (dc coded Picture):
   Sie sind wie die I–Bilder unabhängig von anderen Bildern kodiert, aber mit eingeschränkter Qualität. Sie sollen der Realisation von schnellen Suchläufen dienen; nach [LeGall91] aber nicht in Sequenzen mit anderen Bildtypen gemeinsam vorkommen.

4. Slice Layer
   Das Slice Layer teilt ein einzelnes Bild in mehrere Makroblöcke auf. Diese Aufteilung kann sich von Bild zu Bild ändern. Das Slice Layer ist nur noch auf die Bildkodierung bezogen und läßt sich nicht mehr anwendungsbezogen interpretieren.
5. Makroblock Layer und
6. Block Layer
   die beiden untersten Ebenen sind wie schon das Slice Layer nur noch kodierungsbezogen und von einer Anwendung nicht interpretierbar. Sie sind der Anwendung im allgemeinen auch nicht zugänglich.

Das Produkt *Digital Video Interactive* DVI [Ripley89] [Harney91] [Stein93a], das von der Firma Intel stammt, stellt eine Hardware/Software Umgebung bereit, die für PCs entwickelt wurde. Als Datenmodell kennt DVI nur eine Folge von Einzelwerten, die mit einer definierten Rate verarbeitet werden sollen. Es gibt für DVI unterschiedliche Kodierungen, wobei einerseits zwischen *Production Level Video* (PLV) und *Real Time Video* (RTV) unterschieden wird. Zum anderen wird RTV in Intra–Frame–Kodierung, vergleichbar JPEG, oder Inter–Frame–Kodierung, vergleichbar H.261, angewandt. Dabei stellt RTV ein Verfahren dar, bei dem der Benutzer sowohl die Komprimierung als auch die Dekomprimierung selbst in Realzeit durchführen kann, bei entsprechen eingeschränkter aber im allgemeinen ausreichender

Qualität. PLV dagegen ist ein sogenanntes asymetrisches Verfahren, das für hochqualitative Kodierung einen langen Kompressionsprozess benötigt, dafür aber die Darstellung mit sehr guter Qualität in Realzeit leistet. Aufgrund des einfachen Datenmodells, der bildweisen Kodierung, aber der festgelegten Hardware/Software Umgebung von DIV, kann der einzelne Benutzer zwar auf die Bilder zugreifen, sie aber nicht direkt interpretieren oder verarbeiten.

Das System *Compact Disc Interactive* (CD-I) [PhSo88] [Meißner91], eine gemeinsame Entwicklung von Philips und Sony, spezifiziert ein Datenformat, das es erlaubt, verschiedene Audio– und Video– Daten auf einer Compact Disc (CD) zu speichern. Das System ist in einer eigenen Hardware/Software–Umgebung realisiert, und als Ergänzung zum Video–Heimbereich gedacht. Es wird aber auch als Rechnerzusatzgerät verwendet [Meißner91]. Die CD–I Kodierungen und Datenformate sind von einer Anwendung aus nicht erreichbar.

Die Hardware/Software–Umgebung von QuickTime [Ortiz91] [Burg93] ist nicht so abgeschlossen, wie die von CD–I. Daher ermöglicht sie auch den Zugriff auf QuickTime Daten. Diese sind als Audio– oder Video–*Tracks* definiert [Hoffert92]. Ein Track ist lediglich eine Folge von Audio–Samples oder Video–Bildern. Aus diesen Tracks werden *movies*, indem sie zueinander in zeitliche Relation gebracht werden.

2.2.2 Semantische Datenmodelle

Unter dem Abschnitt semantische Datenmodelle werden die Ansätze beschrieben, die sich mit den spezielle Eigenschaften der Daten kontinuierlicher Medien beschäftigen und zu den Daten Funktionen definieren. Ansätze dazu sind bereits in den hierarchischen Strukturen von QuickTime und MPEG zu finden. Der entscheidende Punkt der semantischen Datenmodelle ist aber die Berücksichtigung der Kontinuität und damit die Integration von Zeitattributen und Zeitoperationen in das Datenmodell. Dieser Punkt fehlt aber in den genannten Ansätzen. Eine semantische Verfeinerung bietet das Datenmodell für Datenbanken, das in [KaMeMe94] erläutert wird. Allerdings konzentriert sich dieses Modell stark auf die Integration in Datenbanken und ist in dieser Form nicht direkt für Anwendungen einsetzbar.

In [Herrtw90] [Herrtw91] werden zur Beschreibung der Zeitparameter *Zeitkapseln* eingeführt, um die Speicherung, den Transport und den Abruf zeitkritischer Daten programmtechnisch handhabbar zu machen. In dieser Arbeit werden grundlegende Begriffe der Zeitinformation vorgestellt, die sich aus verschiedenen Bereichen der Informatik ableiten lassen. *Zeit (time)* repräsentiert Werte, die man von einer Uhr ablesen kann; *Dauer (duration)* gibt Differenzen zwischen diesen Uhrenwerten als Zeitspannen an. Damit ist der *Gültigkeitszeitraum* eines Datums definiert, der von der *Ausführungszeit* unterschieden werden muß. Alle Zeitangaben sind nur innerhalb eines bestimmten *Zeitbezugssystems* gültig, wobei verschiedene Zeitbezugssysteme zueinander in Beziehung gesetzt und kombiniert werden können.

Die grundlegende Definition für Daten kontinuierlicher Medien wird zeitkritischer Datenstrom genannt [Herrtw90]. Er besteht aus einer Folge von zeitkritischen Daten *m*, die sich als Tripel $m=(V, T, U)$ darstellen lassen. Dabei ist V der Datenwert, T ein Zeitwert oder Zeitstempel und U eine Zeitspanne oder Dauer. Der Gültigkeitszeitraum *L* von *m* ist das rechts offene Intervall $L(m)= [T, T+U)$. Weiterhin wird gefordert, daß innerhalb eines zeitkritischen Datenstroms *s* die zeitkritischen Daten nach aufsteigenden Zeitstempeln sortiert sind und die Gültigkeitszeiträume sich nicht überlappen.

Für die zeitkritischen Datenströme werden weitere Unterteilungen in kontinuierliche und nichtkontinuierliche zeitkritische Datenströme sowie periodische und aperiodische Ströme vorgenommen. Als einzige wichtige Kombination werden in [Herrtw90] [Herrtw91] die periodischen, kontinuierlichen, zeitkritischen Datenströme herausgestellt und weiter verfolgt. Eine nähere Behandlung der Zusammenhänge in diesem Datenmodell fehlt.

In einer Zeitkapsel wird ein Datenstrom mit einer Uhr verknüpft. Eine Uhr ist dabei relativ zu einer physikalischen („Echt–") Zeit am Ort der Uhr definiert als Quadrupel $c = (R, S, V_0, T_0)$, wobei R die Rate bestimmt, mit der die Uhr je Sekunde tickt, S den Anzeigeunterschied von einer Sekunde zur nächsten angibt, V_0 den Initialwert der Uhr beim ersten Tick und T_0 die Startzeit der Uhr in („Echt–") Zeit festlegen [Herrtw90] [Herrtw91].

Zeitkapseln können dazu benutzt werden, auf gespeicherte zeitkritische Datenströme zeitabhängig zuzugreifen, dabei wird immer nur das aktuelle Datum ausgegeben. Es wird darauf hingewiesen, daß bei Unterschieden zwischen den Raten bei der Aufzeichnung und beim Lesen Effekte wie Zeitlupe und Zeitraffer auftreten, diese Effekte werden aber nicht weiter untersucht.

Andere Einsatzmöglichkeiten der Zeitkapseln sind die der aufzeichnenden Zeitkapseln, der zeitgesteuerten Pipes und der speziellen Zeitkapseln für die Ein/Ausgabe. Ob und wie diese Zeitkapseln miteinander kooperieren können, wird in [Herrtw90] [Herrtw91] nicht beschrieben. Es wird aber gezeigt, daß eine Kooperation auf die Uhreneinstellung zurückzuführen ist, und wie durch geeignete Einstellung der Startzeiten der Uhren selbst in einem verteilten System eine Kooperation möglich wird.

Diese Arbeit zeigt, daß neben der Integration der Zeitattribute in die Datenbeschreibung auch ein Instrument zu deren Interpretation, eine Uhr, eingeführt werden muß. Ob diese Uhr allerdings, wie in [Herrtw90] [Herrtw91] vorgeschlagen, alle Funktionen wie auf den Speicher schreiben, vom Speicher lesen, Übertragen sowie die Ein– und die Ausgabe kontrollieren muß, wird nicht erörtert.

In [Gibbs94] werden logische Aspekte von zeitbasierten Medien für die Datenhaltung in Datenbanken in den Vordergrund gerückt und die Charakteristika hervorgehoben, die die Daten von anderen unterscheiden. Zu diesen Charakteristika gehören:

- *die Interpretation*
  Mediendaten werden nicht mehr als ein BLOB (binary large object) angesehen, sondern entsprechend eines definierten strukturierten Datentyps interpretiert.
- *die Qualitätsparameter*
  durch die Verarbeitung im Rechner tritt ein Qualitätsverlust ein. Dieser kann bei vielen Kodierungsverfahren eingestellt werden. Daher sollten diese Faktoren auf der Datenmodellierungsebene *nicht* sichtbar sein, sondern als Attribute der Einzelwerte angegeben werden [Gibbs94].
- *die Herleitbarkeit (derivation):*
  läßt sich ein Wert nach einer Funktion aus anderen Werten berechnen, so kann es bei den großen Datenmengen der kontinuierlichen Medien günstiger sein, die Parameter der Funktion zu speichern, als ihren Wert. Bei jedem Zugriff muß dann die Funktion erneut ausgeführt werden. Falls der Zugriff direkt mit einer Präsentation der Werte in Realzeit verbunden ist, muß die Funktion entsprechende Zeitbedingungen einhalten.

- *die Komposition*
  Da Multimediadaten sich durch die Verbindung verschiedener Medien auszeichnen, muß ein Datenbankmodell die Komposition mehrerer Einzelmedien durch Repräsentation der verschiedensten Beziehungen unterstützen.

Als grundlegende Datenabstraktion werden die zeitbehafteten Ströme (*timed streams)* verwendet. Diese sind eine zeitliche Abfolge von *Medienelementen*. Medienelemente werden durch einen *Medientyp* beschrieben, der Attribute und Kodierung der Elemente festlegt, sowie durch *Elementdescriptoren*, die Abweichungen einzelner Elemente beschreiben können. Solche Abweichungen können zum Beispiel andere Farbpaletten für einige Videobilder sein.

Weiterhin wird ein *diskretes Zeitsystem* D_f als Abbildung von den Natürlichen Zahlen in die Reellen Zahlen definiert. Die Elemente des Definitionsbereichs von D_f heißen *diskrete Zeitwerte*, die Elemente des Wertebereichs heißen *kontinuierliche Zeitwerte* und werden in Sekunden angegeben. Die Abbildung ist von der Form $D_f : i \rightarrow (1/f) \cdot i$, wobei f die *Frequenz* des Zeitsystems genannt wird. Beispiele für diskrete Zeitsysteme sind $D_{29,97}$ für North American Video, D_{25} für europäisches Fernsehen, D_{24} für Film und D_{44100} für CD–Audio.

Ein zeitbehafteter Strom besteht nach [Gibbs94] aus einer endlichen Folge von Tripeln $<e_i,s_i,d_i>$, $i=1, .. , n$. Er basiert auf einem Medientyp T und einem diskreten Zeitsystem D. Die e_i sind die Elemente von T, s_i und d_i sind diskrete Zeitwerte im Zeitsystem D. Der Wert s_i heißt Startzeit von e_i und d_i ist die Dauer. Startzeit und Dauer müssen folgenden Bedingungen genügen:

1. $s_{i+1} \geq s_i$
2. $d_i \geq 0$

Unter den zeitbehafteten Strömen werden weitere Einteilungen vorgenommen. Dazu gehören die kontinuierlichen Ströme mit $s_{i+1} = s_i + d_i$; die nicht kontinuierlichen Ströme mit $s_{i+1} \neq s_i + d_i$ wobei Lücken und Überschneidungen zwischen Elementen zugelassen werden; ereignisbasierte Ströme mit $d_i = 0$ für $i=1, .. , n$. Weiterhin gibt es Ströme mit konstanter Frequenz, das sind kontinuierliche Ströme, bei welchen alle Elemente die gleiche Dauer haben; und Ströme mit konstanter Datenrate, wobei das Verhältnis von der Speichergröße eines Werts zu seiner Dauer für alle Elemente konstant ist.

In [FrBöBr92] wird ein weiterer Ansatz zur Modellierung der Daten kontinuierlicher Medien vorgestellt. Darin wird die Folge der Einzeldaten nicht als flacher Strom betrachtet, sondern es werden weitere Strukturen untersucht. Das zentrale Abstraktionsmodell ist die Sequenz, die zwar weiterhin linear ist, ihre Elemente können aber wieder strukturiert sein. So wird eine Menge zusammengehöriger Audiodaten als Liste von Sequenzen modelliert, wobei jede Sequenz als Elemente Arrays von Samples enthält. Die Sequenz wird als abstrakter Datentyp über ihre Funktionen definiert. Dabei wird die Semantik allerdings nicht formal sondern nur informell beschrieben. Der Zeitbezug der Daten wird auf den verschiedenen Granularitätsstufen jeweils entsprechend [Herrtw90] mit Startzeit und Dauer eingeführt. Dieser Teil von [FrBöBr92] stellt eines Ausgangspunkt der in dieser Arbeit vorgestellten Datenmodellierung dar.

Zusammenfassend erkennt man, daß als Datenmodell eine Folge von Einzelwerten mit den Attributen Startzeit und Dauer geeignet ist. In den vorgestellten Ansätzen werden verschiedene zum Teil einander entsprechende Eigenschaften dieser Folgen aufgeführt. Es fehlt aber

eine Herleitung der Eigenschaften und ihrer Zusammenhänge, sowie eine Beschreibung von Operationen, die sich auf dieses Datenmodell anwenden lassen. Gerade diese Operationen aber sind für die funktionsorientierten Bausteine besonders wichtig.

2.2.3 Zeitmodelle

Wie schon in den semantischen Datenmodellen für kontinuierliche Medien [Herrtw90] [Herrtw91] und [Gibbs94] durch die Uhren und das diskrete Zeitsystem deutlich wird, dient zur Interpretation der Daten kontinuierlicher Medien ein Zeitmodell. Dabei sollte auf der Grundlage der allgemeinen verteilten Systeme eine möglichst allgemeingültige Form gefunden werden.

In der Untersuchung über die relativistische Struktur logischer Zeit in verteilten Systemen [Mattern91] wird zwischen der *dichten Realzeit* und der *diskreten logischen Zeit* unterschieden. Die Realzeit ist linear; Modelle für sie sind die rationalen oder die reellen Zahlen. Die logische Zeit dagegen ist in verteilten Systemen nicht linear. Einzelne Prozesse messen ihre lokale Zeit durch *Uhren*, die mittels eines Zählers realisiert werden, der bei jedem Ereignis des Prozesses um 1 erhöht wird. Die lokale Zeit ist also diskret und linear.

Der Zeit kommt in Verbindung mit Multimediaanwendungen eine doppelte Aufgabe zu. Zum einen bestimmt sie die Gültigkeitsdauer der Werte eines einzelnen kontinuierlichen Medii, zum anderen bestimmt sie eine gemeinsame Grundlage für die Synchronisation der verschiedenen kontinuierlichen Medien.

In allen Modellen für kontinuierliche Medien, die die Zeit einschließen, wird sowohl eine äußere als auch eine innere Zeit berücksichtigt. Allerdings unterscheiden sich die Ansätze im Grad der Integration der beiden Zeiten in das Modell. Wie bereits erwähnt, kennen die aktiven Objekte der Klasse MediaObject in [Gibbs91], eine Weltzeit und eine Objektzeit sowie die Umrechnung von der einen in die andere. Dabei bestimmt die Weltzeit die Zeit, in der die Anwendung abläuft, die Objektzeit dagegen die Gültigkeitszeiträume eines kontinuierlichen Medii. Dabei wird in [Gibbs91] keine Beziehung zu einer vom Benutzer meßbaren Realzeit definiert. In [Herrtw90] [Herrtw91] ist die äußere Zeit dagegen gerade nur durch die Beziehung der Uhr der Zeitkapsel zu einer Realzeit gegeben. Es wird hier allerdings nicht bestimmt, in welcher Art und Weise diese Zeiten ihr Voranschreiten betreiben. Dieses Verhältnis des Voranschreitens wird wiederum durch die diskreten Zeitsysteme in [Gibbs94] definiert, ohne daß dort der Bezug zu den Daten hergestellt wird. Auch eine Beziehung zwischen [Gibbs91] und [Gibbs94] wird nicht hergestellt.

Somit ergeben sich für die bisher genannten Zeitmodelle einzeln nur eingeschränkte Sichtweisen und erst eine Kombination der verschiedenen Ansätze scheint ein umfassendes Modell zu liefern. Allerdings beschreibt keiner der Ansätze Operationen auf dem Zeitmodell oder den Uhren, so daß kein Modell im Sinne der abstrakten Datentypen oder der objektorientierten Modellierung vorliegt.

Das an der Carnegie Mellon Universität entwickelte TACTUS [Dann92] kennt beide Aufgaben der Zeit in Multimediaanwendungen und enthält damit zum einen die Möglichkeit der Spezifikation von zeitlichen Beziehungen zwischen kontinuierlichen Medien und eine Methode zu deren Umsetzung in Zeitparameter, zum anderen ist in TACTUS eine Hierarchie von Uhrenmodellen definiert, die die einzelnen kontinuierlichen Medien gemäß der Zeitparameter steuern.

Die Uhrenmodelle von TACTUS kennen eine äußere Zeit, die von dem Objekt *RealTime* als Uhr gemessen wird. Diese RealTime–Uhr wird über die Zeitsteuerungen des Betriebssystems fortgeschaltet. Des weiteren gibt es für die interne Zeit *Clock*–Objekte, die die Uhren zu den kontinuierliche Medien repräsentierenden *Active*–Objekten bilden. Eine Unterklasse von Clock, die *Stream*–Objekte, bilden jeweils eine gemeinsame Uhr für eine zu synchronisierende Menge von Active–Objekten. In der Klassenhierarchie gilt, daß Stream–Objekt ein Clock–Objekt und Clock–Objekt ein Active–Objekt ist. Somit verlangt jede Uhr als Stream–Objekt oder als Clock–Objekt nach einer sie steuernden Uhr, da jedes Active–Objekt mit einer Uhr verbunden sein muß. Die einander steuernden Uhren bilden damit einen Baum von Uhrenobjekten, dessen Wurzel die RealTime–Uhr ist, dessen innere Knoten Stream–Objekte sind, und dessen Blätter von den Active–Objekten oder geeigneten Unterklassen von Active–Objekten gebildet werden.

Die Steuerung eines Active–Objekts durch seine Uhr erfolgt mittels der Methoden **`requestTick`** und **`Kick`**. Dabei wird die erste Methode vom Active–Objekt bei seiner Uhr, die zweite von der Uhr bei ihrem Active–Objekt aufgerufen. Voraussetzung für diesen Austausch von Methoden ist, daß jedem Active–Objekt mittels seiner Methode **`UseClock`** die zugeordnete Uhr bekanntgemacht wird.

In [Dann92] wird nicht erläutert, wie die Uhren eingestellt werden, oder welche Parameter sie haben. Das alles bleibt im TACTUS-Server verborgen, der die Synchronisation der Medien und die Uhrenwerte berechnet.

Ein dem TACTUS–Ansatz ähnliches Modell von einem Uhrenbaum, der zur Steuerung von zu synchronisierenden kontinuierlichen Medien dient, wurde an der Universität Stuttgart im Rahmen des *Cinema*–Ansatzes entwickelt [RotHelb94] [RotHelb96]. Auch diese Arbeiten fanden parallel zur vorliegenden Arbeit aber davon unabhängig statt. Im *Cinema*–Ansatz werden die Definitionen eines Datenstroms und einer damit verbundenen Uhr aus [Herrtw90] übernommen, wobei einige Umbenennungen vorkommen. In [Herrtw90] ist eine Uhr gegeben durch $c = (R, S, V_0, T_0)$. In *Cinema* ist eine Uhr definiert als $C ::= (R, M, T, S)$, wobei R und S aus [Herrtw90] übernommen sind und V_0 dem T, T_0 dem M entspricht. In *Cinema* wird explizit die Umrechnung von Realzeit t in Medienzeit m angegeben als: $m = M + S \cdot R\,(t - T)$.

Bezogen auf das Komponentenmodell von *Cinema* wird mit dem Uhrenmodell eine Steuerung der Komponenten erreicht. Dazu werden in [RotHelb94] sechs Operationen auf den Uhren definiert. *Start (M)* startet eine Uhr zum Medienzeitpunkt M; *Halt (M)* hält eine Uhr an, sobald sie den Wert M erreicht hat; *Prepare (M)* bereitet das Starten der Uhr zur Medienzeit M vor; *Clear ()* löscht die Inhalte aller internen Puffer, die mit den Strömen verbunden sind; *Scale (M, S)* ändert die Geschwindigkeit S der Uhr zum Medienzeitpunkt M; *Lock (O)* und *Unlock (O)* werden benötigt, um in einer Uhrenhierarchie die Weiterleitung von Operationen zu verhindern oder wieder zuzulassen. Durch die Angabe der Operation im Parameter O lassen sich einzelne Operationen selektiv sperren. Aus einer anderen Sicht bewirkt das Sperren und Entsperren, daß einige Operationen intern in Uhren bleiben. Dieses Konzept der Unterscheidung von internen und externen Operationen findet sich auch in der vorliegenden Arbeit.

Die Uhren des *Cinema*–Modells werden entweder mit Quellen– oder mit Senkenkomponenten, nie aber mit Zwischenkomponenten verbunden. Senkenkomponenten müssen mit einer Uhr verbunden sein, bei Quellen sind die Uhren optional. In [RotHelb96] werden Beziehungen zwischen den Komponenten untersucht. Es wird gezeigt, wie sich Steuerung und

Synchronisation über einen Baum von Uhren, der Uhrenhierarchie, verteilen lassen. Zwar läuft im Gegensatz zum TACTUS–Modell jede Uhr einzeln nach ihrer örtlichen Realzeit und der obigen Gleichung, dennoch stehen in einer Uhrenhierarchie Uhren zueinander entweder in einer *control*–Beziehung oder in einer *synchronization*–Beziehung. Während in einer *control*–Beziehung die Operationen an einer Uhr zu den mit ihr abhängig verbundenen Uhren weitergeleitet werden, werden in einer *synchronization*–Beziehung auch die Werte der abhängigen Uhren beeinflußt, so daß ihre Uhren in synchronisierter Weise fortgeschaltet werden. Diese Unterscheidung entspricht den in der vorliegenden Arbeit eingeführten Begriffen *enge* und *lose Kopplung*.

Die Uhrenhierarchie ist ein Baum mit gerichteten Kanten, wobei die Knoten im Baum Uhren und die Kanten *control*– oder *synchronization*–Beziehungen sind. Jeder Kante wird zusätzlich zu ihrem Beziehungstyp mit den Attributen Referenzpunkt und Verzögerung versehen. Ein Referenzpunkt bezeichnet je einen Zeitpunkt in den Medienzeiten der beiden miteinander verknüpften Uhren, die der gleichen Realzeit zugeordnet werden sollen. Das Verzögerungsattribut bestimmt, mit welcher Verzögerung Operationen auf den Uhren weitergeleitet werden sollen.

Mit Hilfe der Zeittransformationsfunktion $Trans(C_1, C_2, m)$ zur Umsetzung der Uhrenbeziehungen wird unter Verwendung der Uhrenparameter beider Uhren einen Uhrenwert m der Uhr C_1 in einen Wert $m_2=Trans(C_1, C_2, m)$ der Uhr C_2, umrechnet. In [RotHelb94] wird ausdrücklich hervorgehoben, daß die Uhren unabhängig voneinander nur über ihren Realzeitbezug laufen.

Mit Hilfe dieser Transformationsfunktion läßt sich nachweisen, daß sich die Uhrenhierarchie auf zwei Stufen reduzieren läßt. Eine Stufe bildet die Wurzel der Hierarchie, die zweite Stufe bilden die Blätter. Alle dazwischen liegenden Stufen lassen sich mittels der Transformationsfunktion entfernen. Eine entsprechende Hierarchie stellt das in dieser Arbeit entworfene Ticker-Schrittgeber-Modell dar.

2.2.4 Synchronisation

Durch die postulierte Unabhängigkeit der einzelnen Medien eines Multimediasystems voneinander ergeben sich sowohl dessen Flexibilität als auch der Zwang zur Synchronisation der unabhängigen Medien. Die Synchronisation realisiert die zeitlichen Beziehungen oder Zeitrelationen zwischen den einzelnen Medien. Alle Zeitrelationen gehen davon aus, daß man von zwei Ereignissen feststellen kann, ob sie gleichzeitig geschehen. Ein Nacheinander von Ereignissen wird durch eine Translation gleichzeitiger Ereignisse auf der Zeitachse erreicht. Damit stellen sich für die Synchronisation drei Hauptfragen:

1. Unter welchen Bedingungen sind Ereignisse gleichzeitig?
2. Wie kann man Zeitrelationen definieren?
3. Wie kann man Zeitrelationen realisieren?

Zu allen drei Fragen gibt es eine Menge von Untersuchungen, die hier nicht annähernd alle behandelt werden können. Es sollen aber einige Lösungsansätze vorgestellt werden.

Die erste Frage läßt sich zunächst einfach damit beantworten, daß zwei Ereignisse objektiv gleichzeitig zueinander sind, wenn sie mit der selben Uhr gemessen werden und dabei den gleichen Uhrenwert erhalten. Das ist lokal an einem Ort einfach, in verteilten Systemen mit unterschiedlichen Uhren und unterschiedlichen Signallaufzeiten aber unmöglich. Hier ist die

Umkehrung der Betrachtung hilfreich, die in [Mattern91] vorgenommen wird. Es werden zunächst alle die Ereignisse zusammengestellt, die in einer Ursache–Wirkungs–Beziehung zueinander stehen. Wenn für zwei Ereignisse *e* und *e'* weder *e* eine Ursache für *e'* noch *e'* eine Ursache für *e* ist, dann sind *e* und *e'* parallel zueinander und können in einem globalen Sinn gleichzeitig sein. Diese Gleichzeitigkeit ist aber nicht transitiv!

Als weitere Antworten auf die gestellten Fragen findet man somit folgendes:

1. gleichzeitig können alle Ereignisse sein, die in keiner Ursache-Wirkungs-Relation zueinander stehen.
2. eine Zeitrelation ist eine Beschreibung der Ursache-Wirkungs-Relation.

Betrachtet man nun alle Ereignisse in einem verteilten System, so gilt zunächst, daß Ereignisse nach den einzelnen Orten der Verteilung zusammengefaßt werden können. Dabei gilt, daß Ereignisse an einem Ort in Ursache-Wirkungs-Relation stehen, mit der Ausnahme spezieller asynchroner Ereignisse; Ereignisse an verschiedenen Orten stehen nicht in Ursache-Wirkungs-Relation, mit der Ausnahme spezieller Kooperationsereignisse. Aufgabe der Synchronisation ist es, lokal an jedem Ort dafür zu sorgen, daß alle Ursache-Wirkungs-Relationen eingehalten werden.

Diese Überlegungen helfen bei der Beantwortung der Frage, wie weitgehend die Gleichzeitigkeit von Ereignissen in einem verteilten Multimediasystem garantiert werden muß, oder anders formuliert, ob jeder gedachte globale Beobachter eines verteilten Multimediasystems zwei zueinander parallele Ereignisse an verschiedenen Orten auch als im selben Augenblick geschehen erkennen muß. Da diese Frage nicht umfassend zu beantworten ist, und weit in den Bereich der kooperativen Zusammenarbeit reicht, werden hier die minimalen Anforderungen an die Synchronisation beschrieben. Es bleiben zunächst folgende Forderungen an verteilte Multimediaanwendungen.

- Gleichzeitigkeit wird zunächst auf einen Ort definiert, so daß nur Ereignisse, die am selben Ort stattfinden als gleichzeitig gefordert werden können
- Jeder Benutzer eines verteilten Multimediasystems muß Ereignisse in der Reihenfolge erhalten, daß nie lokal bei ihm ein Ereignis *e* vor einem Ereignis *e'* auftritt und *e* die Wirkung der Ursache *e'* ist.
- Ist für zwei Ereignisse *e* und *e'* gefordert, daß *e* und *e'* gleichzeitig stattfinden sollen, so muß jeder Benutzer lokal bei sich *e* und *e'* als im selben Augenblick geschehen erkennen.

Diese Form der lokalen Gleichzeitigkeit ist transitiv. Allerdings sagt sie nichts darüber aus, ob zwei Benutzer, die das gleiche Ereignis an unterschiedlichen Orten beobachten, dies objektiv gleichzeitig also im selben Augenblick tun. Es soll vielmehr herausgestellt werden, daß das keine notwendige Anforderung ist.

Die Ursache-Wirkungs-Beziehung kann in hochgradig verteilten, kooperativen Anwendungen gleichberechtigter Partner, wie zum Beispiel in einem Telekonferenzsystem, sehr komplex werden. Daher können zur Vereinfachung verschiedene weitere Maßnahmen ergriffen werden. Beispiele für solche Maßnahmen sind Token-Mechanismen (Rednerstab) und Verzögerungen bei der Verteilung von Ereignissen, damit Reaktionen auf diese Ereignisse, also Wirkungen, erst dann eintreten können, wenn alle Teilnehmer alle möglichen Ursachen empfangen haben.

Eine wichtige Bedingung ist die, daß der Benutzer die Ereignisse als im selben Augenblick geschehen erkennen soll. Der menschliche Benutzer hat für diese Wahrnehmung keine festen Toleranzwerte, sondern bezieht unterschiedliche Situationen und Darstellungen mit ein. Eine Untersuchung dazu wird in [Stein94] dargestellt. Die Tatsache, daß solche Toleranzen existieren, ist generell eine Voraussetzung für die technischen Medien Audio und Film, besonders aber für die digitale Verarbeitung und multimediale Anwendungen im Rechner.

Damit ist festgehalten, wie man die allgemeine Gleichzeitigkeit durch objektive Messung nachweisen kann und wie man sie für verteilte Systeme auf eine lokale Gleichzeitigkeit reduzieren kann. Weiterhin zu untersuchen sind die Definition und Realisierung von Zeitrelationen.

Die Definition der Zeitrelation zweier Ereignisse läßt sich leicht aus den Referenzpunkten von [RotHelb94] herleiten. Zwei Ereignisse sind genau dann gleichzeitig, wenn die Uhrenwerte der zu den beiden Ereignissen lokalen Uhren zum Zeitpunkt der Ereignisse auf den selben Zeitpunkt einer gemeinsamen Uhr bezogen werden können. Ein Referenzpunkt definiert also genau die Zeit, zu der beide Ereignisse stattfinden, und zwar in beiden Zeitbezugssystemen.

Dehnt man die Zeitrelationen von Ereignissen auf Aktionen mit einer Dauer aus, so kann man jeder Aktion ein Anfangs– und ein Endeereignis zuordnen. Aus atomaren Aktionen, die nur diese beiden Ereignisse haben, lassen sich längere Aktionen zusammensetzen, die als Synchronisationspunkte die Anfangs– und Endeereignisse der atomaren Aktionen besitzen [Hoep91a] [Hoep91b]. Die atomaren und die zusammengesetzten Aktionen beschreiben die Daten eines kontinuierlichen Medii. Die Synchronisationspunkte der Daten bilden die Menge der Zeitpunkte, die in Referenzpunkten verwendet werden kann. Aus der einen Zeitrelation *gleichzeitig* werden verschiedene vom Benutzer zu definierende Zeitrelationen.

Für die vom Benutzer zu definierenden Zeitrelationen sind verschiedene Ansätze vorgestellt worden. Alle beruhen auf ex ante bekannten Zeitintervallen, deren zeitliche Lage zueinander untersucht wird. Die Frage, wie man Zeitrelationen zwischen Aktionen definieren kann, wurde schon in [Allen83] untersucht und in dreizehn Relationen angegeben. Ausgehend davon wurden in [Hoep91a] [Hoep91b] Pfadausdrücke zur Beschreibung der Relationen entwickelt. Dabei werden die dreizehn Relationen, die vollständig alle möglichen Beziehungen beschreiben, um die inversen Relationen auf *before, meets, during, overlaps, starts, finishes,*und *equal* reduziert. Die inversen Relationen erhält man durch Vertauschen der beteiligten Aktionen. In [Hoep91a] [Hoep91b] wird zu je zwei Aktionen, die in einer Zeitbeziehung zueinander stehen, eine Zeitgeberaktion so definiert, daß die Kombination einer Aktion mit der Zeitgeberaktion eine zusammengesetzte Aktion bildet, die entweder das Anfangsereignis gleichzeitig mit dem der zweiten Aktion hat, oder das Endeereignis der zweiten Aktion ist gleichzeitig mit dem Anfangsereignis der zusammengesetzten Aktion. Die Zeitgeberfunktion füllt die Lücke zwischen den beiden anderen Aktionen.

Das Spezifikationsmodell für Zeitrelationen in [Little90] benutzt erweiterte zeitabhängige Petri–Netze. Die Erweiterungen bestehen in Ressourceninformationen, die den Prozesse repräsentierenden Stellen zugeordnet werden. Das so entstandene Petri–Netz wird *Object Composition Petri Net* (OCPN) genannt. Die Verwendung von OCPN's ist ein verbreitetes Hilfsmittel und wird in [PraRag94] gut veranschaulicht. OCPN's und ähnliche Methoden werden in [WahlRo94] und [WahlRo95] nebeneinander betrachtet.

Ein anderer Ansatz besteht in der Synchronisation an einer gemeinsamen Zeitachse [BlHüLa91] [Blako93]. Diese Zeitachse gilt entweder generell für alle Zeitintervalle, wie zum Beispiel im Projekt Athena [Hodges89] oder in Quicktime [Ortiz91] [Littm91], oder jedes Intervall besitzt eine eigene Zeitachse und diese wird zu einer allgemeinen Weltzeit in Beziehung gesetzt. Ein Beispiel hierfür ist die Spezifikation im *active media* Modell [Tsich91]; ähnliche Ansätze beinhaltet HyTime (*Hypermedia and time–based Structuring Language*) [HyTime91] [Gold91] [Gold91a]. Die gemeinsame Zeitachse mit lokalen Zeitachsen für die einzelnen Medienstücke entspricht auch dem in dieser Arbeit vorgestellten Ticker–Schrittgeber–Modell mit den Startzeitpunkten der Sequenzen.

In [Unge92] wird gezeigt, wie die Beschreibung der Zeitrelationen in den Modellen von [Little90] und [Hoep91a] auf eine einfache Relation zu einer gemeinsamen Zeitachse umgerechnet werden können. Mittels dieser Relationen lassen sich einerseits auf einfache Weise Widersprüche finden, die in den Definitionen auftreten können, andererseits können daraus die Startzeitpunkte der einzelnen Medien für das in dieser Arbeit propagierte Ticker–Schrittgeber–Modell berechnet werden. Die Relationen eignen sich damit sowohl zur internen Repräsentation verschiedener vom Benutzer vorgenommener Beschreibungen, als auch zur Steuerung des Ablaufs der Präsentation in einfacher Weise. Zu ähnlichen Ergebnissen, die [Unge92] bestätigen, kommt [FunPon94].

Ein dritter Ansatz der Intersynchronisationsbeschreibungen durch den Benutzer basiert auf der Synchronisation an Referenzpunkten [Stein90a]. Hierbei werden kontinuierliche Medien als Sequenzen zeitunabhängiger Teileinheiten betrachtet, die in konstanten Zeitabständen dargestellt werden. Die Synchronisation wird in diesem Fall auf die Teileinheiten bezogen, die zur gleichen Zeit dargestellt werden sollen [Blako93].

Die Realisierung der Synchronisation kann in lokale Maßnahmen zur Realisierung der lokalen Gleichzeitigkeit und verteilte Maßnahmen zur Bestimmung der Ursache-Wirkungs-Relation eingeteilt werden. Die lokalen Maßnahmen beziehen sich dabei auf den Ort der Präsentation und stellen einen Bezug zu lokalen Uhren her. Verteilte Maßnahmen zur Synchronisation müssen immer dann eingesetzt werden, wenn Daten an der Senke zu synchronisieren sind aber mindestens eine Quelle an einem anderen Ort als die Senke liegt. In [LeydTeu91] werden folgende drei Orte der Synchronisation untersucht: nur an der Senke, im Netzwerk und an der Quelle.

Ist nur die Datensenke für die Synchronisation zuständig, kann es sein, daß sehr große Puffer benötigt werden, um zu schnell oder in wechselnden Schüben ankommende Daten zwischenzuspeichern. Es ist daher sinnvoll schon vorher Maßnahmen zur Synchronisation zu ergreifen. In [Ferrari91] wird ein Verfahren zur Verteilung des benötigten Pufferplatzes auf das Netzwerk vorgestellt, das zur Minimierung des gesamten Pufferplatzes beiträgt. Ein weiterer Vorteil der Verteilung des Pufferplatzes auf das Netzwerk ist Reduktion der Ressourcenanforderungen an die Senke.

Die Synchronisation der Information an der Datenquelle kann nur dann erfolgen, wenn alle miteinander zu synchronisierenden Daten an der selben Quelle entstehen. Entsprechend dem Fernsehsignal, bei dem Bild und Ton gemeinsam ineinander verwoben übertragen werden, kann die Quelle die Daten so miteinander verbinden, daß die zu synchronisierenden Daten auch miteinander an der Senke ankommen. Die Synchronisation an der Quelle, auch an mehreren Quellen miteinander, spielt in einem speziellen Bereich von Multimediaanwendungen eine Rolle, in dem vorgefertigte Multimediadaten an einen oder mehrere Benutzer verteilt

werden. Anwendungen dieser Art sind unter dem Stichwort *Video–on–Demand* bekannt geworden und stellen eine Art Videothek im Direktzugriff dar. Ein Ansatz zur Synchronisation mehrerer Quellen wird in [LittlKao92] vorgestellt. Die Synchronisation mehrerer Quellen erfolgt dadurch, daß die in Video–on–Demand a priori bekannten Synchronisationsanforderungen in den verschiedenen Quellen als Referenzpunkte definiert werden, die beim Zusammentreffen aller Datenströme ausgewertet werden. Die Auswertung kann dabei von einer speziellen Instanz im Datenpfad vor der Senke oder von der Senke direkt ausgeführt werden.

2.2.5 Datenübertragung

Die kontinuierlichen Medien stellen gegenüber den Medien Text, Graphik und Bild neue Anforderungen an die Datenübertragung. Im wesentlichen beruhen diese Anforderungen darauf, daß eine definierte Menge an Daten pro Zeit übertragen werden muß. Zur Lösung der mit den neuen Anforderungen verbundenen Probleme wurde eine große Vielzahl von Vorschlägen und Entwicklungen vorgestellt. Zwei Beispiele sind das *Session–Reservation–Protokoll (SRP)* [AndHerr91] und das *Internet Stream Protokoll – Version 2 (St–II)* [Topo90].

Die gemeinsame Grundlage aller Ansätze zur Übertragung der Daten kontinuierlicher Medien ist eine Beschreibung der Anforderungen dieser Daten durch Dienstgüteparameter (Quality of Service ’QoS’ Parameter). In [LeydTeu91] wird als notwendige Teilmenge der QoS Parameter für Daten kontinuierlicher Medien folgender Satz von Parametern gefordert:

- Durchsatz oder Bitrate
- maximal zulässige Übertragungsverzögerung
- maximale Varianz der Übertragungsverzögerung (jitter)
- Genauigkeit der Synchronisation
- Bittfehlerrate
- Paketfehlerrate

Dieser Satz von Parametern ist noch sehr an der Bitübertragung orientiert und kann daher für den anwendungsorientierten Übertragungsbedarf noch gestrafft werden. Diese Parameter können beim Verbindungsaufbau verhandelt werden [WoQaGh94].

Es gibt allerdings keinen allgemeingültigen Satz von Werten für die QoS Parametern, der für alle Daten kontinuierlicher Medien einheitlich ihre Anforderungen beschreibt. Dafür sind die Kodierungen der Daten, die übertragen werden sollen, zu unterschiedlich. Außerdem spielen auch die Anforderungen der Benutzer eine wichtige Rolle, da sie bestimmen, wie gut die Übertragung mindestens sein muß. Oft müssen die QoS Parameter erst aus den an die Präsentation gestellten Anforderungen berechnet werden. Verschiedenen Ebenen der QoS Parameter und ihre Realisierung werden in [CaCoHu94] und in [JaSrNe95] betrachtet.

In der folgenden Tabelle sind einige Anhaltspunkte für die Bestimmung der QoS Parameter angegeben. Sie sind zum Teil aus [HehSaSt90] abgeleitet.

Datenart	Datenrate	Verzögerung	Jitter	Fehlerrate
Audio				
telephonqualität	64 kbit/s	0.25 s	10 ms	10^{-1}
mono HIFI	360 kbit/s	0.25 s		
stereo HIFI	720 kbit/s	0.25 s		
Video				
unkomprimiert	240 Mbit/s	0.25 s	10 ms	10^{-3}
MPEG I	1 Mbit/s	0.25 s		10^{-9}
MPEG II	5-10 Mbit/s	0.25 s		10^{-9}

Tab. 2 *Multimediadaten und QoS Parameter*

Eine eingehendere Behandlung der Datenübertragung geht über den Rahmen dieser Arbeit hinaus und ist für die Ergebnisse dieser Arbeit auch nicht weiter relevant.

2.2.6 Anforderungskatalog für semantische Datenmodelle

Aus den untersuchten Ansätzen zur semantischen Modellierung der Daten kontinuierlicher Medien läßt sich folgender Anforderungskatalog mit den Hauptpunkten zeitbasierte Daten und Uhrenmodell herleiten.

- zeitparametrisiertes Datenmodell
  - > Wertetupel: (Wert, <Zeitattribute>)
  - > Operationen auf den Daten
    - Operationen auf Werten
    - Operationen auf Zeitattributen
    - Wechselwirkung zwischen den Operationen
  - > Strukturierung der Daten nach Granularitätsstufen
- Uhrenmodell
  - > Trennung von Realzeit und Medienzeit
  - > Uhrendefinition und Bestimmung der Uhrenwerte
  - > Operationen auf den Uhren

Die behandelten Datenmodelle stellen bisher jeweils nur einen für eine bestimmte Anwendung wichtigen Aspekt heraus. Die Datenformate von DVI, JPEG und MPEG befassen sich vorwiegend mit der effizienten Speicherung und der schnellen Präsentation von gespeicherten Videodaten [Blako93].

Die semantische Modellierung der Daten kontinuierlicher Medien ist für die Modellierung von funktionsorientierten Bausteinen und deren Operationen grundlegend, da hier die semantische Beschreibung der Speicher– und der Verarbeitungsoperationen abgeleitet werden kann.

Bisherige Modelle betrachten aber nur einzelne Aspekte kontinuierlicher Medien. Ausgehend vom Modell der Zeitkapsel [Herrtw90] wird in der vorliegenden Arbeit unter Einführung der in [FrBöBr92] beschriebenen Granularitätsstufen ein allgemeines Datenmodell entwickelt.

3 Ein Funktionenmodell für Anwendungen kontinuierlicher Medien

In diesem Kapitel werden zunächst typische, immer wiederkehrende Beispiele für Anwendungen oder Anwendungsteile vorgestellt. Daraus läßt sich ein Satz von Grundfunktionen ableiten, der zu einem vollständigen Funktionenmodell erweitert wird. Im Gegensatz zu bisher bekannten Einteilungen wird hier nicht von Anwendungsbereichen, Geräten oder Daten ausgegangen. Es wird vielmehr gezeigt, aus welchen „Bausteinen" sich multimediale Anwendungen aufbauen lassen. Für eine grundlegende Lösung werden die einfachsten Funktionen auf einem allgemeinen kontinuierlichen Medium untersucht. Neben einem einfachen Aufbau der Anwendungen ergeben sich Vorteile durch die Wiederverwendbarkeit der Anwendungsteile.

3.1 Multimediale Anwendungen

Die Integration der Monomedien im Rechner bietet die neue Möglichkeit, alle Medien von einem Gerät aus gemeinsam zu steuern, und die Auswertung der Beziehungen zwischen den Monomedien vom Rechner durchführen zu lassen. Dadurch können dem Benutzer eine Vielzahl von Aufgaben abgenommen werden. Weniger Aufgaben für den Benutzer bedeuten meist auch einfachere Handhabbarkeit und größere Benutzerfreundlichkeit der Anwendung sowie weniger Bedienfehler.

Auf diese multimedialen Systeme lassen sich nun viele bekannte Medien abbilden oder einfach erweitern [Brand90]. Ein Beispiel bietet sich in der Musik. Bisher gibt es für Solisten die Möglichkeit, zum Zusammenspiel statt eines realen Orchesters eine sogenannte Minus-One-Aufzeichnung zu verwenden. Das sind spezielle Aufzeichnungen, die nur den Orchesterpart eines Konzerts ohne das Soloinstrument enthalten. Sie haben den Nachteil, daß man an eine vorgegebene Geschwindigkeit gebunden ist, und, daß man das vom Orchester verwendete Notenmaterial benötigt. In einem multimedialen System mit digital gespeicherter Musik ist es dagegen zum Beispiel möglich, die Geschwindigkeit zu verändern, ohne die Klanghöhe zu beeinflussen. Die Noten können als spezielles Textdokument gespeichert und passend zum augenblicklichen Spiel angezeigt werden. Dazu kann sogar noch der Takt in den Notentext eingeblendet werden. Die selbe Anwendung ist für Sänger unter dem Begriff „Karaoke" bekannt. Multimediale Systeme bieten auch neue Möglichkeiten zur Interaktion und Kooperation. Als Beispiele sein hier interaktives Entwerfen, Videokonferenzen und interaktive Spielfilme genannt. Weitere Ausführungen zu diesem Thema sind in [Brand90] und [WiBla94] zu finden. In diesem sich rasch entwickelndem Bereich lassen sich ständig weitere, ganz neue Anwendungen ergänzen.

3.1.1 Eine Beispielanwendung

Anhand des Karaoke–Beispiels, das in dieser Arbeit immer wieder aufgenommen wird, soll gezeigt werden, welche verschiedenen, zum Teil aufeinander aufbauenden Systeme grundlegende Multimediaanwendungen sind. Zunächst benötigt man einfache ***Aufzeichnungs–*** und ***Wiedergabe–Systeme***, um die einzelnen Medien, hier Musik und Stimme, als Mediendaten zu erfassen und präsentieren zu können. Das einfache Wiedergabesystem muß

die gleichzeitige Präsentation mehrerer unabhängiger Medien durchführen können, das heißt deren Synchronisation beinhalten. Diese einfachen Systeme bilden die Grundlage für die im folgenden beschriebenen komplexeren Systeme.

Ein ***Autorensystem*** dient der Definition der Beziehungen zwischen den unabhängigen Medien, hier den einzelnen Instrumentalstimmen, und der Bearbeitung der aufgenommenen Daten. Ein Autorensystem beinhaltet Schnittstellen zum Aufzeichnungssystem, zum Wiedergabesystem und zu weiteren Systemen, wie Textverarbeitung, Graphikanwendungen und Bildverarbeitung. Mit einem Autorensystem wird ein multimediales Dokument erstellt. Ein ***Lesersystem*** ist ein interaktives Wiedergabesystem, das es dem Leser erlaubt, ein Dokument schrittweise, in unterschiedlichen Geschwindigkeiten und ab von ihm bestimmten Punkten zu präsentieren. Dazu benötigt es ähnliche Funktionen, wie das Autorensystem, mit dem Unterschied, daß die Funktionen des Lesersystems nicht persistent sind. Ein Lesersystem verwendet oder integriert ein Wiedergabesystem. Es kann bis zu Hypermediasystemen mit integrierter Präsentation von Text, Graphik und Video erweitert werden.

In der Beispielanwendung für Karaoke sollen nun zunächst nur die kontinuierlichen Medien betrachtet werden. Die verschiedenen Instrumentalstimmen sein bereits aufgezeichnet. Mittels des Karaoke–Autorensystems werden sie nun zu einem einzigen Instrumentalkanal gemischt. Dazu müssen die Instrumentalstimmen in der Geschwindigkeit und zu jedem Zeitpunkt in der Lautstärke aufeinander abgestimmt werden. Zu dem Instrumentalkanal wird als weiterer eigenständiger Kanal der Gesang angepaßt. Als Ergebnis wird ein Dokument bestehend aus zwei Medien, dem Instrumentalkanal und dem Gesangskanal, gespeichert. Dieses Dokument kann von einem Karaoke–Lesersystem wiedergegeben werden. Dabei kann der Leser zunächst beide Kanäle so anhören, wie sie vom Autorensystem zusammengemischt wurden. Zum Üben soll die Möglichkeit bestehen, die Lautstärke des Gesangskanals zu verändern, ihn ganz stumm zu schalten, aber zu jedem beliebigen Zeitpunkt wieder hören zu lassen. Darüber hinaus soll der Leser die Möglichkeit haben, die Abspielgeschwindigkeit zu verändern, so daß die Tonhöhe erhalten bleibt. Auf diese Weise kann der Leser langsam üben oder Zeile für Zeile durchgehen, was später schnell vorgetragen wird. Für die Präsentation der Karaoke soll der live gesungene Part parallel verarbeitet und gemeinsam mit der aufgezeichneten Version vorgetragen werden. Dazu ist ein **Live–Präsentations–System** bereitzustellen. Die Einflußmöglichkeiten auf die Live–Daten müssen stark beschränkt werden, damit ihre Originalität erhalten bleibt, Vergleiche möglich sind und Manipulationen verhindert werden.

3.1.2 Erstellung von Multimediaanwendungen

Bei der Erstellung der Anwendung spielen die Unterschiede zwischen Autorensystemen und Anwendungs– oder Leser–Systemen zunächst keine Rolle. Es kann vielmehr, wie im folgenden beschrieben, ähnlich dem Aufbau von analogen Audio/Video–Anwendungen vorgegangen werden.

Unabhängig von der Intention der Anwendung, wird zunächst ihre Funktion beschrieben und in Teilfunktionen zerlegt. Es werden Gruppen von Funktionen zusammengestellt, die die jeweiligen Teilfunktionen realisieren. Die Teilfunktionen werden so miteinander verbunden, daß die geforderte Gesamtfunktionalität erbracht wird. Die Geräteauswahl für den Ein-

satz in der Anwendung erfolgt sowohl nach der von ihnen gebotenen Funktionalität als auch nach der Qualität der Funktionen und deren Kosten. Sind die Geräte festgelegt, werden sie untereinander gemäß der Funktionsverbindungen verkabelt.

Welche Verbindungen vorgenommen werden, ist zum einen von der Anwendung, zum anderen aber von der Kompatibilität der Geräte untereinander abhängig. Aus Kompatibilitätsgründen kann es sein, daß die Anwendung um besondere Geräte zur gegenseitigen Anpassung der bereits ausgewählten Geräte erweitert wird. Diese Geräte können sowohl einfache Konvertierungen der Daten ausführen als auch die Qualität oder Dienstgüte erhöhen.

Der Aufbau von Multimediaanwendungen mittels Geräten läßt sich auf die Modellierung von multimedialen Anwendungen mittels Funktionen auf dem Rechner übertragen. Im ersten Schritt der Modellierung werden also die Funktionen der Geräte als Grundlage zur Entwicklung der wiederverwendbaren Bausteine herangezogen. Die Geräte liefern den Ansatzpunkt für die Beschreibung der Funktion der Bausteine; dabei sind zunächst Grundfunktionen zu isolieren. Zu diesen Grundfunktionen sind Erweiterungen so zu definieren, daß jede beliebige Anwendung entwickelt werden kann. So steht bei diesem Aufbau die Modularität und funktionale Abgeschlossenheit der Anwendungsteile im Vordergrund.

Dieses Verfahren zur Erstellung von Multimediaanwendungen kann sowohl für einen statischen als auch für einen dynamischen Aufbau verwendet werden. Die in diesem Kapitel verwendete graphische Beschreibung macht darüber keine Einschränkungen. Erst die noch zu wählende Beschreibungsmethode für verteilte Anwendungen bestimmt, ob nur ein statischer oder auch ein dynamischer Aufbau möglich ist.

3.1.3 Verteilungsaspekte

In diesem Abschnitt wird eine kurze Übersicht über verschiedene Aspekte der Verteilung gegeben. Es wird gezeigt, in welchen Bereichen die Verteilung der Anwendungen Auswirkungen hat, und unter welchen Bedingungen die Orte der Verteilung eine Relevanz erhalten oder aber Ortstransparenz herrscht.

Betrachtet man verteilte Anwendungen, so läßt sich die Verteilung in drei Dimensionen beschreiben. Verteilbar sind Hardware, Steuerung und Daten [SloKr89]. Wenn man zur Hardware die Geräte und ihre Treiber rechnet, so bildet die Software die Steuerung. Wie bereits beschrieben, kann die Hardware multimedialer Anwendungen grundsätzlich als verteiltes System angesehen werden. Die Verteilung der Hardware gibt somit Randbedingungen für die Verteilung der Software vor.

Die Verteilung der Software in Form von einzelnen Softwarekomponenten wird durch die geforderte Rechen– und Kommunikationsleistung der Komponenten beeinflußt. Je größer das Kommunikationsaufkommen zwischen zwei Komponenten, desto kürzer und billiger sollen die Kommunikationswege sein. Für weite Kommunikationswege soll die Leistung gesenkt werden, indem beispielsweise die Rechenleistung durch zusätzliche Komprimierung und Dekomprimierung erhöht wird. Bei Anwendungen, an denen mehrere Benutzer beteiligt sind, ist durch diese ein Teil der Verteilung der Steuerung vorgegeben. Die Verteilung der Daten ergibt sich mit dem rasch wachsenden Volumen der Multimediadaten als ökonomische Anforderung.

Im Gegensatz zu vielen anderen verteilten Anwendungen kann die Verteilung von Multimediaanwendungen nie gänzlich transparent sein, da durch die Schnittstellen zur Umwelt und zum Benutzer bestimmte Orte vorgegeben sind. Es ist zum Beispiel nicht egal, ob

eine Kamera in Halle A oder eine in Halle B verwendet wird, um die Produktion in Halle A zu überwachen. Ebenso ist es wichtig, an welchem Bildschirm der Gesprächspartner der Videokonferenz dargestellt wird, und ob der Ton im gleichen Raum zu hören ist. Diese Ortsabhängigkeit spielt aber erst bei der Benutzung einer Anwendung eine Rolle und nicht notwendigerweise bei der Modellierung. Die Verteilung der Komponenten muß also durch den Benutzer gesteuert werden. Dies sind Aufgaben, die insgesamt dem Management verteilter Systeme zugeordnet werden und nicht multimediaspezifisch sind. Daher wird im folgenden, sofern die Ortsabhängigkeit nicht explizit erwähnt wird, die Verteilung als transparent angesehen. Es werden somit zunächst nur die Funktionen der Steuerung und die Kommunikation betrachtet, die konkrete Verteilung kommt erst bei der Aktivierung einer bestimmten Anwendung hinzu.

3.2 Grundfunktionen kontinuierlicher Medien

3.2.1 Aufspaltung der Monomedien in Grundfunktionen

Die in Kapitel 1 vorgestellten Monomedien lassen sich nach ihren Grundfunktionen weiter unterteilen in *Perzeptions–*, *Übertragungs–*, *Speicher–* und *Präsentations–Medien*. Werden die Mediendaten mit einbezogen, so kann man nach [MHEG90] zusätzlich von *Repräsentationsmedien* sprechen, die den Datentyp beschreiben, und von *Transportmedien*, die die physikalischen Datenträger bezeichnen, mittels derer man Daten zwischen verschiedenen Speichern transportieren kann.

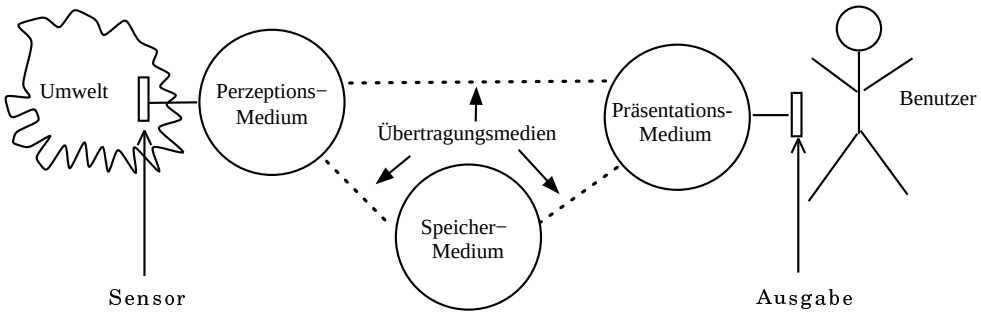


Abb. 2 Medienaufteilung nach Grundfunktionen

Perzeptionsmedien sind von den menschlichen Sinnen abgeleitet. Sie nehmen über eine spezielle Schnittstelle, einen Sensor, Ausschnitte aus der Umwelt auf und kodieren diese zu Mediendaten. Die Übertragungsmedien dienen der lokalen Translation der Mediendaten; sie überwinden räumliche Distanzen. Die Speichermedien dienen der temporalen Translation; sie überwinden zeitliche Distanzen. Außerdem kann an den gespeicherten Daten mit Hilfe von Werkzeuge eine Bearbeitung, Veränderung oder Ergänzung vorgenommen werden. Die Präsentationsmedien interpretieren die Mediendaten zu einer Darstellung, die als Ausgabe an den Benutzer gegeben wird. Betrachtet man die Perzeption, Speicherung, Übertragung und Präsentation und ihre Beziehungen untereinander, so erkennt man, daß die Übertragung eine

Sonderrolle einnimmt. Sie wird nicht direkt vom Benutzer, sondern über die Perzeption, Speicherung oder Präsentation gesteuert. Somit werden die Grundfunktionen für die Anwendung nur durch die Perzeption, Präsentation und Speicherung gebildet.

Den Grundfunktionen ist gemeinsam, daß sie einerseits eine Schnittstelle für die Mediendaten haben, die kontinuierlich Daten aufnimmt oder abgibt. Andererseits benötigen sie zusätzlich zu dieser Schnittstelle eine weitere, über die ein Benutzer die Funktion steuert (zum Beispiel starten oder anhalten).

Die vorgestellten Perzeptions–, Übertragungs–, Speicher– und Präsentations–Medien kennzeichnen eine eher technisch funktionale Sichtweise und entsprechen nicht direkt den Medien, die in der täglichen Anwendung auftreten. Zur besseren Unterscheidung werden die Medien aus der täglichen Anwendung im folgenden als *Anwendungsmedien*, und die hier vorgestellten Medien, die aus der Aufteilung nach Grundfunktionen hervorgehen, nur noch durch ihre Funktion bezeichnet.

Die Grundfunktionen Perzeption, Speicherung und Präsentation werden von der Anwendung direkt gesteuert. Jede einzelne ist dabei in eine Betriebsumgebung eingebettet. Diese Umgebungen liefern die Betriebsvoraussetzungen für die Grundfunktionen. Für den Speicher übernimmt die Umgebung vor allem die Integration von Dateisystem, Fileservern und Datenbanken. Die Umgebungen für die Perzeption und Präsentation realisieren vor allem den Zeitbezug. Ausführlicher werden die Umgebungen am Ende dieses Kapitels behandelt.

3.2.2 Basisanwendungen

Manche Anwendungsmedien lassen sich direkt den Grundfunktionen zuordnen. Rundfunk ist ein Übertragungsmedium für akustische Sinneseindrücke. Druckmedien speichern und präsentieren Schrift. Andere Anwendungsmedien kombinieren die vorgestellten Medien, wie z. B. Video ein Speichermedium in einem zugehörigen Abspielgerät und einen Monitor als Präsentationsmedium benötigt. Typisch hieran ist, daß die Perzeptions–, Speicher– und Präsentations–Medien in Form von **Geräten** in der Anwendung auftreten, die Übertragungsmedien dagegen meist Kabel– oder Funk–**Verbindungen** sind.

Betrachtet man die verschiedenen Anwendungsmedien einerseits und die möglichen Kombinationen von Perzeptions–, Übertragungs–, Speicher– und Präsentations–Medien andererseits, so lassen sich drei Kategorien von einfachen Anwendungsmedien unterscheiden. Jede Kategorie ist ein Aggregat aus Perzeptions–, Übertragungs–, Speicher– oder Präsentations–Medien, wobei das Übertragungsmedium ein optionaler Bestandteil ist, da zum Beispiel eine Videokamera mit integriertem Beobachtungsmonitor ein Übertragungsmedium nicht zwingend benötigt.

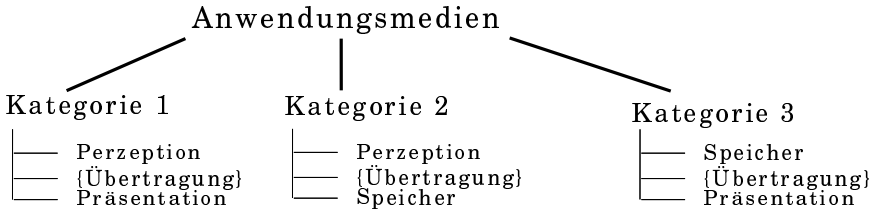


Abb. 3 Kategorien von Anwendungsmedien

1. Kategorie der Anwendungsmedien
   Sie nehmen Mediendaten aus der Umwelt auf und stellen sie sofort dar;
   z.B. Meßfühler mit Anzeige, Fernseh–Live–Sendung.
   Im Karaoke–Beispiel ist die Präsentation des live gesungenen Parts eine Anwendung der Kategorie 1.
2. Kategorie der Anwendungsmedien
   Sie nehmen Mediendaten aus der Umwelt auf und speichern diese;
   z.B. verschiedene Photo–, Film– oder Videokameras; Tonbandgeräte.
   Im Karaoke–Beispiel ist das einfache Aufzeichnungssystem ein Anwendungsmedium der Kategorie 2.
3. Kategorie der Anwendungsmedien
   Sie stellen gespeicherte Mediendaten dar;
   z.B. Stereoanlagen, Walk-Man, Videogeräte.
   Im Karaoke–Beispiel ist das einfache Wiedergabesystem ein Anwendungsmedium der Kategorie 3.

Diese drei Kategorien stellen die grundlegenden Anwendungen kontinuierlicher Medien dar. Sie können untereinander kombiniert oder in andere Anwendungen integriert werden und lassen sich so zum Beispiel im Bereich von Gruppenarbeit zur Videokonferenz oder der Lehrsysteme zur Situationssimulation verwenden. Die beiden Kategorien 2 und 3 gehören zusammen und bilden ein Gegensatzpaar. Sie werden oft in einem Gerät zusammengefaßt, wie z. B. in Tonbandgeräten.

Die Anwendungsmedien zeichnen sich dadurch aus, daß zwar die drei vorgestellten Kategorien beschrieben werden können, es allerdings nicht möglich ist, diese in Form einer Klassenhierarchie bereitzustellen oder eine solche für weitere Anwendungen auszubauen. Das ergibt sich daraus, daß eine Anwendung immer spezielle Geräte mit von der Anwendung geforderten Eigenschaften benutzt. Betrachtet man das Problem der Zusammensetzung der Anwendungen genauer, erkennt man, daß es hier allgemein um die Definition verteilter Anwendungen geht, die hier nicht erschöpfend behandelt werden kann.

3.2.3 Datenflußstruktur zwischen Grundfunktionen

Wie die verschiedenen, datenflußorientierten Modellierungsansätze zeigen, ist es in der Modellierung sinnvoll, zwischen Quellen und Senken von Multimediadaten unterscheiden zu können. Daher werden im folgenden die Speichermedien nach ihrem Datenflußverhalten in zwei Grundfunktionen, schreibender und lesender Speicher, unterteilt. Es tritt eine neue Übertragung von Multimediadaten zwischen Speichern auf. Diese Übertragung unterschei-

det sich von den bisher betrachteten, da sie von der Umwelt beziehungsweise vom Benutzer entkoppelt ist. Hier kann eine Datenübertragung mit konventionellen Verfahren stattfinden (z. B. FTAM, ftp). Implizit existiert für die gespeicherten Daten auch die Gegenrichtung, da sie sich wahlweise lesenden oder (über-) schreibenden Speichern zuordnen lassen. Für die Datenübertragung vom lesenden zum schreibenden Speicher wird eine weitere Anwendungskategorie eingeführt. Aufgrund ihrer Zeitunabhängigkeit ist sie auch als Hintergrundanwendung ausführbar. Sie erhält die Kategorie 4.

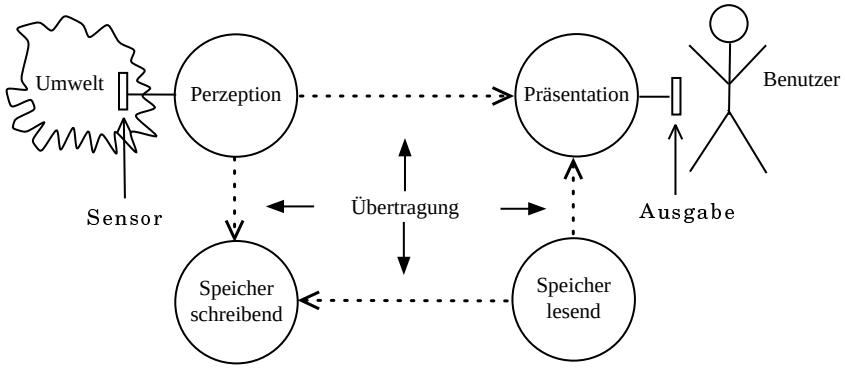


Abb. 4 Spaltung der Grundfunktionen in Quellen und Senken

Im Grundfunktionenmodell mit Quellen und Senken zur Darstellung der Datenflußstruktur liegen sich die beiden Quellen und Senken jeweils diagonal gegenüber. Das gesamte Grundfunktionenmodell kann zur Repräsentation unterschiedlicher Anwendungen horizontal oder vertikal geteilt werden, so daß jeweils eine Quelle mit einer Senke verbunden wird.

Durch horizontale Teilung erhält man zwei Ebenen. Auf der Ebene von Perzeption und Präsentation werden kontinuierliche Medien kontinuierlich zeitabhängig interpretiert; das erfordert bei der Verarbeitung Realzeitfähigkeit der Rechner. Diese Ebene entspricht der Multimediaanwendungskategorie 1. Auf der Speicherebene liegen dagegen zwar die Daten der kontinuierlichen Medien vor, werden aber zeitunabhängig und nicht notwendigerweise kontinuierlich verarbeitet. Das entspricht allgemeiner Datenverarbeitung, und ist in diesem Sinn nicht multimediaspezifisch, da zeitunabhängig. Für die Datenverarbeitungsfunktionen gilt aber, daß sie die Multimediadaten typgerecht verarbeiten müssen.

Teilt man die Grundfunktionen vertikal in Perzeption und schreibenden Speicher gegenüber Präsentation und lesenden Speicher, erhält man die Multimediaanwendungskategorien 2 und 3. Durch den Bezug zur Umwelt beziehungsweise zum Benutzer werden Realzeitbedingungen an die gesamte Anwendung gestellt. Hier fordern die Perzeption und die Präsentation von den Speichern die Einhaltung von Zeitbedingungen.

3.2.4 Anwendungsbeschreibung im Grundfunktionenmodell

Mit dem Grundfunktionenmodell lassen sich verschiedene Anwendungen beschreiben. Dazu werden zunächst die verschiedenen Kategorien der Anwendungsmedien herangezogen, gleichzeitig damit ergeben sich einfache Beispiele für die verschiedenen Anwendungssysteme des Karaoke–Beispiels. Weitere Anwendungsbeispiele werden aus den Funktionseinhei-

ten nach [SteiMey92] abgeleitet. Diese Funktionseinheiten dienen dabei auch als Nachweis der Vollständigkeit des Grundfunktionenmodells, da sich mit ihm alle Funktionseinheiten nachbilden lasen.

Für die Bildung der Anwendungsmedien der Kategorien 1 bis 4 wird jede Grundfunktion mit genau einer weiteren verbunden. Anders gesagt werden zwei Grundfunktionen und die Übertragung zwischen ihnen aktiviert.

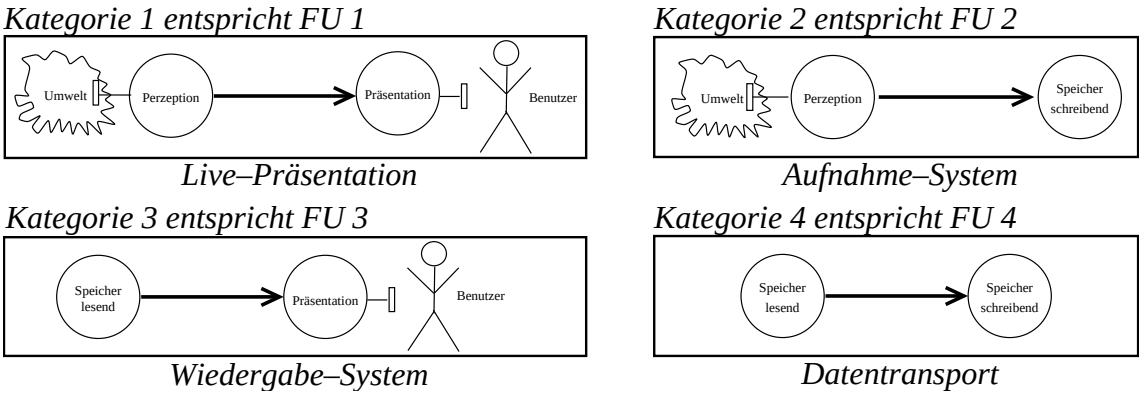
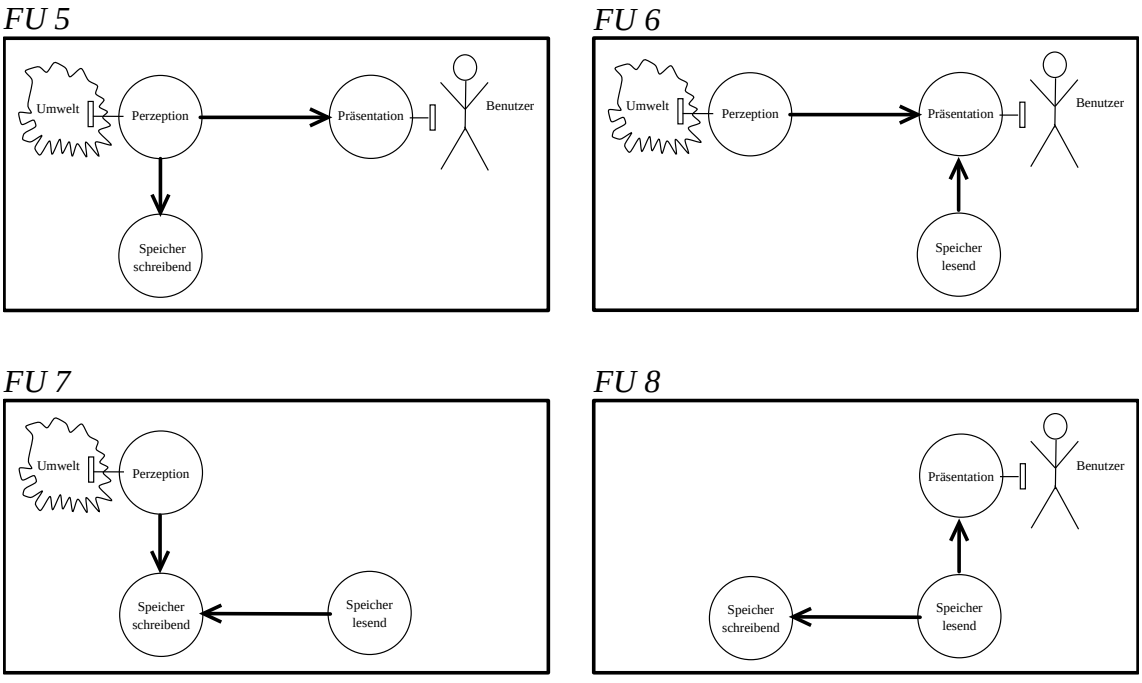


Abb. 5 Anwendungsmedien der Kategorien 1 bis 4 im Grundfunktionenmodell

Die restlichen Funktionseinheiten FU 5 bis FU 9 erhält man, wenn man zusätzlich zuläßt, daß eine Grundfunktion gleichzeitig mehrere Verbindungen haben kann. Im Grundfunktionenmodell ist darüber ebenso, wie in [SteiMey92] keine Einschränkung gemacht. Es ergeben sich somit die in Abbildung 6 angegebenen Kombinationen.



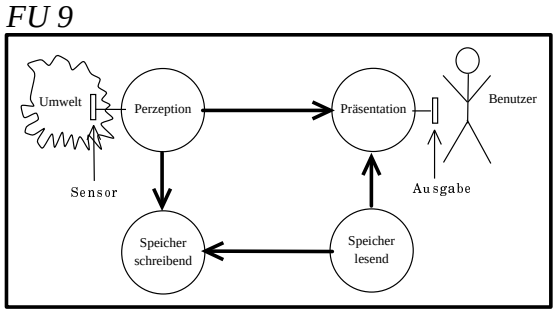


Abb. 6 Funktionseinheiten 5 bis 9 im Grundfunktionenmodell

Stellt man so die Grundfunktionen den Funktionseinheiten gegenüber, erkennt man ihren Vorteil. Mit den wenigen Grundfunktionen, die noch keine Anwendung festlegen, lassen sich sehr flexibel alle Anwendungen beschreiben. Die Grundfunktionen selbst sind dabei stets wiederverwendbar und verfeinerbar, bieten also eine Grundlage der Bausteinspezifikation für kontinuierliche Medien.

Bei der Realisierung der Funktionseinheiten 5 bis 9 mittels des Grundfunktionenmodells fällt auf, daß aus den in Abb. 4 vorgesehenen möglichen Verbindungen zwischen Grundfunktionen mehrere ausgewählt werden können. Modellierungstechnisch ist dies ein Nachteil, der vielen ähnlichen Ansätzen ebenfalls anhaftet: Eine Grundfunktion kann theoretisch beliebig viele Verbindungen, d. h. Schnittstellen, zu anderen Grundfunktionen haben. Im folgenden Abschnitt wird gezeigt, wie dieser Nachteil durch Entkopplung der Verteilung und der Mischung von den Grundfunktionen beseitigt werden kann.

3.3 Erweiterung des Modells

Im bisher vorgestellten Grundfunktionenmodell können lediglich kontinuierliche Daten aufgenommen, gespeichert, gelesen und dargestellt werden, es ist aber nicht möglich die Daten dabei zu verändern. Daher wird das Grundfunktionenmodell um datenverarbeitende Funktionen erweitert. Dies sein Funktionen, die ausschließlich Daten verändern (Symbol: ◇). Im soweit entwickelten Modell ist es noch nicht möglich, eine Verteilung oder Mischung von kontinuierlichen Daten direkt zu beschreiben. Es wird daher eine weitere Funktion (Symbol: ⬡) eingeführt, die ausschließlich die Verteilung und/oder die Mischung durchführen kann. Im Unterschied zu den Grundfunktionen, die eine gewisse Hardware voraussetzen, können diese Verarbeitungsfunktionen ganz in Software realisiert werden. Aus dem Grundfunktionenmodell wird durch die Integration der Verarbeitungsfunktionen das allgemeine Funktionenmodell für Anwendungen kontinuierlicher Medien.

3.3.1 Einordnung von Verarbeitungsfunktionen in das Funktionenmodell

In jede der vier Übertragungen kontinuierlicher Daten können Verarbeitungsfunktionen zur Datenveränderung eingefügt werden, die genau zwei Funktionen miteinander verbinden. Daher werden zunächst nach ihrer Position im Modell die Funktionstypen f_1, f_2, f_3, f_4 unterschieden. Worin die Unterschiede und Gemeinsamkeiten der Funktionstypen bestehen, ist hier nicht relevant.

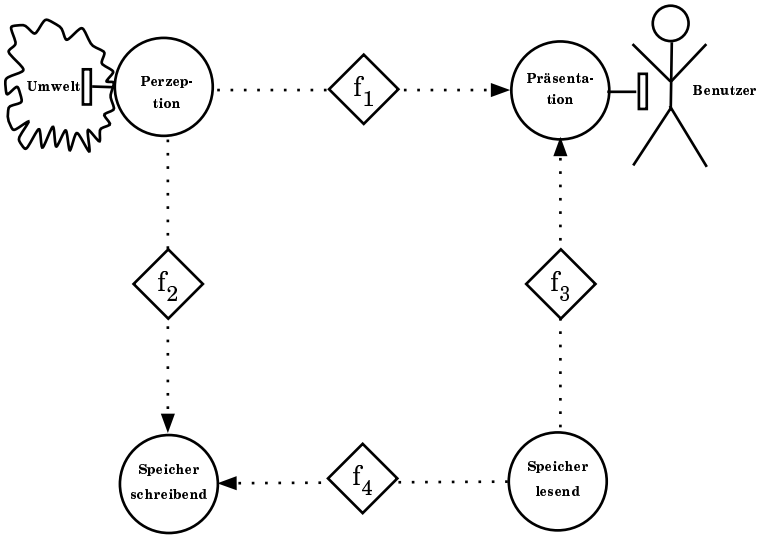


Abb. 7 das Funktionenmodell mit Verarbeitungsfunktionen

In einem separaten Schritt wird der Funktionstyp f_5 zum Verteilen und Mischen in das Modell eingefügt, er kann mit jeder Grundfunktion verbunden werden. Seine Eigenart besteht darin, daß nicht mehr genau zwei Grundfunktionen durch eine Verarbeitungsfunktion verbunden werden, sondern in einer Anwendung mindestens drei Funktionen mit f_5 verbunden sind. Da alle Grundfunktionen mit f_5 verbunden werden können, wird f_5 in der Mitte des Modells dargestellt.

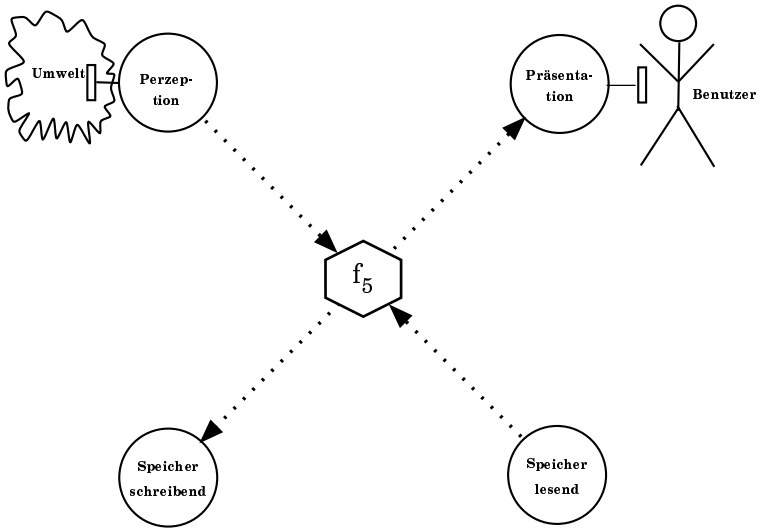


Abb. 8 Grundfunktionen mit Mischen und Verteilen

Wie im Grundfunktionenmodell können auch im Modell der Grundfunktionen mit Mischen und Verteilen Verarbeitungsfunktionen eingesetzt werden. Dabei sind auf jeder Verbindung zwei der Typen f_1, f_2, f_3, f_4 erlaubt. In der Abbildung 9 werden daher beide Indizes der zulässigen Typen aufgeführt.

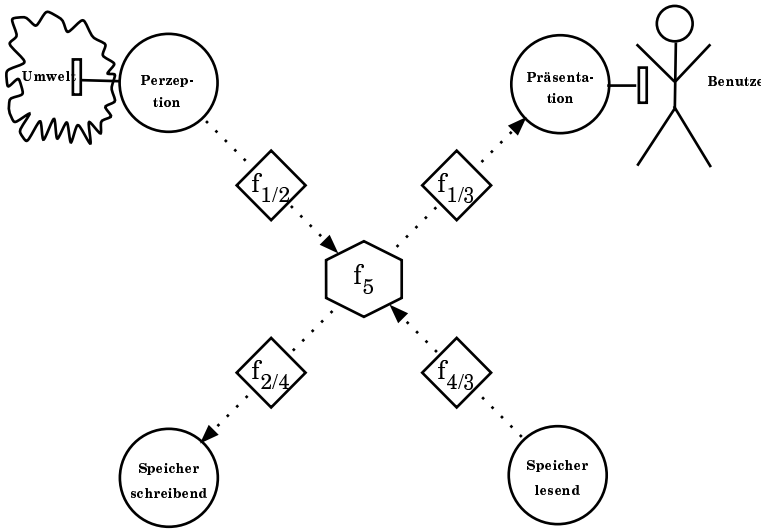


Abb. 9 Mischen und Verteilen mit Verarbeitungsfunktionen

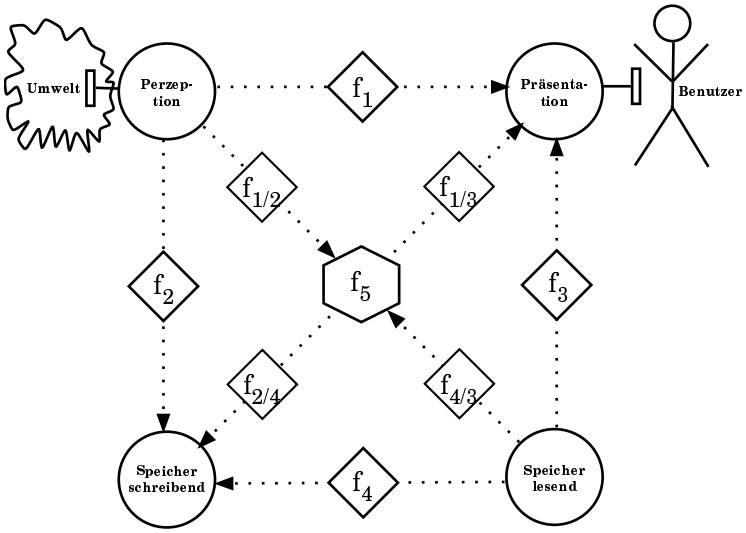


Abb. 10 Das Funktionenmodell

Das vollständige erweiterte Gundfunktionenmodell, das in Abbildung 10 beschrieben ist, wir nur noch kurz als das Funktionenmodell bezeichnet. Im Funktionenmodell sind alle Verarbeitungsfunktionentypen einschließlich Mischen und Verteilen enthalten. Auf jeder Verbindung können Daten verändert werden. Die Einführung der Verarbeitungsfunktionen in die Datenübertragung kann auch als Verfeinerung des Übertragungswegs verwendet werden, ergibt aber für die Anwendung eine grundlegend neue Funktionalität.

Das Funktionenmodell erlaubt nun zusätzlich zum Grundfunktionenmodell die explizite Spezifikation der Veränderung oder Vervielfältigung der Daten. Es ist bezüglich der Funktionstypen vollständig, da die Grundfunktionen vollständig vermascht sind [Fritzsche96].

3.3.2 Beschreibung der Verarbeitungsfunktionen

Die Funktionstypen f_1, f_2, f_3, f_4 lassen sich einheitlich als einfache Datentransformation zeitabhängiger Daten einer Menge *D* in zeitabhängige Daten einer Menge *D'* beschreiben durch: $f_{1...4}: D \rightarrow D'$.
Die Funktionstypen unterscheiden sich in ihren Randbedingungen, die anschließend untersucht werden. Welcher Art die Datentransformationen sein können wird mit der Datenmodellierung im folgenden Kapitel beschrieben.

Der Funktionstyp f_5 kann in verschiedene Varianten zerlegt werden, die sich in ihrer Semantik unterscheiden, in ihrer Darstellung im Modell aber gleichen. Es sollen hier zunächst die Varianten ausgewählt werden, die die einfachsten, grundlegenden Funktionen bieten, die nicht weiter zerlegbar sind. Im folgenden werden diese Varianten geeignet erweitert, damit sie einfache, universal verwendbare Bausteine liefern.

Allgemein ist f_5 *:* $f_5 : (D_1, \ldots, D_n) \rightarrow (D_1', \ldots, D_m')$

Falls n < m ist, werden einige Daten mehrfach benutzt, das entspricht einer Verteilung der Daten. Falls n > m ist, müssen einige Daten gemischt werden. Falls n = m ist, können sowohl Verteilung als auch Mischen von Daten auftreten oder keines von beiden. Daher wird f_5 in die Varianten f_5° *Verteilung* oder *Vervielfältigung* und f_5* *Mischen* unterteilt. Nicht weiter zerlegbare Funktionen erhält man für die Fälle:

1. f_5° mit n=1 und m>1;
2. f_5* mit n>1 und m=1.

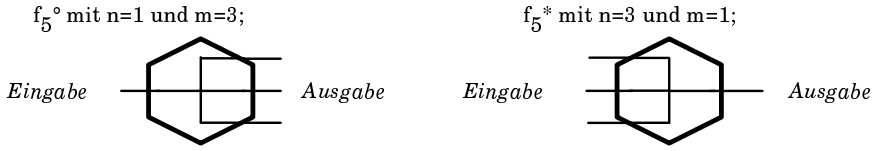


Abb. 11 Beispiele für einfache Verteilung f_5° *und einfaches Mischen* f_5*

Für die einfachen Funktionen können je zwei Fälle unterschieden werden, die echte Funktion und der Spezialfall der Selektion.

1. f_5°*, n=1, m>1;*

a. Verteilung: f_5°*:* $D \rightarrow (D_1', \ldots, D'_m)$ *mit* $D_j' = D$ *für* $1 \leq j \leq m$.

b. f_5° *heißt Ausgabeselektion, falls genau ein* $j' \in \{1,\ldots,m\}$ *existiert mit:* $D'_{j'}=D$ *und* $D'_j = \emptyset$ *für* $j \in \{1,\ldots,m\}$ *und* $j \neq j'$*. Dann heißt* $D'_{j'}$*, die aktive Ausgabe, alle anderen Ausgaben sind inaktiv.*

2. f_5*, n>1, m=1;*

a. Mischen: f_5*:* $(D_1, \ldots, D_n) \rightarrow D'$ *mit* $D' = mix(D_1, \ldots, D_n)$*,*
Die Ausführung von $mix(D_1,\ldots, D_n)$ *ist durch die Anwendung spezifiziert.*

b. f_5* *heißt Eingabeselektion, falls genau ein* $i' \in \{1,\ldots,n\}$ *existiert mit:* $D_{i'}=D'$*.*
Dann heißt $D_{i'}$*, die aktive Eingabe, alle anderen Eingaben sind inaktiv.*

Zur Verallgemeinerung sollen die Verteilung oder Vervielfältigung $f_5°$ und das Mischen f_5* n > 1 Eingaben und m > 1 Ausgaben haben. Zu den Mengen der Indizes In = {1, ... , n} und Out = {1, ... , m} werden die Teilmengen der aktiven Eingaben $In^a \subseteq$ In und der aktiven Ausgaben $Out^a \subseteq$ Out definiert. Damit wird beschrieben, welche Eingaben und Ausgaben selektiert sind. Es ist also möglich anzugeben, wieviel Ein– und Ausgabe–Stellen eine Funktion f_5 besitzt, und welche davon gerade benutzt werden.

Betrachtet man nun die verallgemeinerten Funktionen $f_5°$ und f_5* , so ist $f_5°$ reine Verteilung; zu jeder aktiven Eingabe gibt es mindestens eine aktive Ausgabe, aber jede aktive Ausgabe kommt von genau einer Eingabe. Die Funktion f_5* ist dagegen eine Mischung, die auch Verteilung enthalten könnte, diese ist aber für f_5* ausgeschlossen; jede aktive Eingabe wird zu nur genau einer Ausgabe gemischt, für jede aktive Ausgabe gibt es mindestens eine aktive Eingabe.

Abb. 12 verallgemeinerte Verteilung und verallgemeinertes Mischen

1. *allgemeine Verteilung:* $f_5°: (D_1, \dots, D_n) \rightarrow (D'_1, \dots, D'_m)$,
   $In^a \subseteq In=\{1, \dots, n\}$, $Out^a \subseteq Out=\{1, \dots, m\}$:
   für $j \in Out$ *und* $j \notin Out^a$ *gilt:* $D'_j = \varnothing$,
   für $i \in In^a$ *gibt es ein* $j \in Out^a$ *mit :* $D'_j = D_i$,
   für $j \in Out^a$ *gibt es genau ein* $i \in In^a$ *mit :* $D'_j = D_i$.

2. *allgemeines Mischen:* $f_5*: (D_1, \dots, D_n) \rightarrow (D'_1, \dots, D'_m)$,
   $In^a \subseteq In=\{1, \dots, n\}$, $Out^a \subseteq Out=\{1, \dots, m\}$:
   für $j \in Out$ *und* $j \notin Out^a$ *gilt:* $D'_j = \varnothing$,
   für $i \in In^a$ *gibt es genau ein* $j \in Out^a$ *mit* $D'_j = mix_j\left(D_{i'_1}, \dots, D_{i'_k}\right)$
   und $i \in \{i'_1, \dots, i'_k\} \subseteq In^a$,
   für $j \in Out^a$ *gibt es ein* $i \in In^a$ *mit :* $D'_j = mix_j\left(D_{i'_1}, \dots, D_{i'_k}\right)$
   und $i \in \{i'_1, \dots, i'_k\} \subseteq In^a$.

   da f_5* *keine Verteilung enthalten soll, muß für alle* $a, b \in Out$ *gelten:*

$$
\begin{aligned}
&(a \neq b \\
&\wedge\ D'_a = mix_a(Di_1, \dots, Di_k) \quad \text{und } \{i'_1, \dots, i'_k\} \subseteq In^a \\
&\wedge\ D'_b = mix_b(Dj_1, \dots, Dj_l) \quad \text{und } \{j'_1, \dots, j'_l\} \subseteq In^a \\
&) \\
&\Rightarrow \{i_1, \dots, i_k\} \cap \{j_1, \dots, j_l\} = \varnothing
\end{aligned}
$$

Betrachtet man das Verhältnis zwischen f_5, $f_5°$ und f_5*, so erkennt man, daß man f_5 als die Nacheinanderausführung von f_5* nach $f_5°$ angeben kann. Für f_5 ist es gleich, ob zuerst gemischt und dann verteilt wird, oder umgekehrt. Die Unterschiede liegen in der Anzahl der Eingaben und der Ausgaben, also den *n* und den *m* von $f_5°$ und f_5*. Eine Vertauschung der Funktionen kann zur Aufwandsverminderung in f_5 dienen. Ohne Beschränkung der Allgemeinheit sei im folgenden f_5* nach $f_5°$ auszuführen. Das heißt, zuerst werden die Eingaben vervielfältigt und dann werden die zusammengehörenden zueinander gemischt. Es muß dabei gelten, daß $Out(f_5°)=Out^a(f_5°)=In(f_5*)$. Wie diese Forderung eingeschränkt werden kann, wird im Abschnitt über zusammengesetzte Verarbeitungsfunktionen erläutert. Damit läßt sich eine allgemeine Funktion f_5* auf mehrere einfache Mischfunktionen f_5* mit m = 1 reduzieren.

Man kann die Vervielfältigung $f_5°$ durch einen Vektor v der Dimension m beschreiben, die Mischung f_5* durch eine (n × m) Matrix A. Die Funktion f_5 ist dann die Vektormultiplikation v·A.

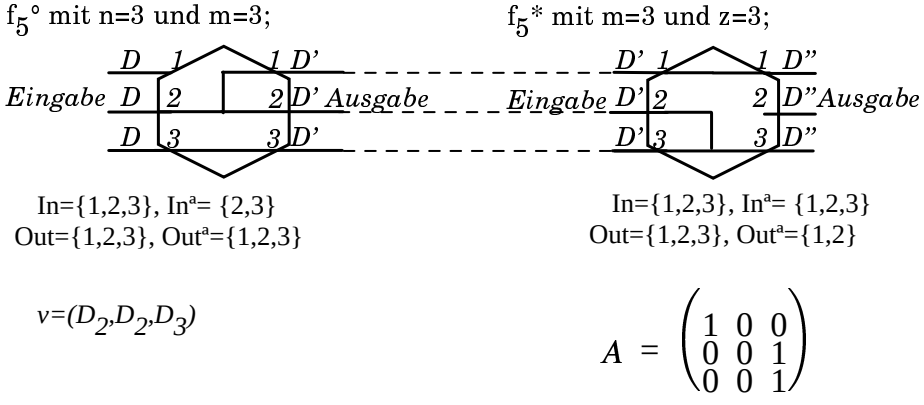
Sei $f_5 := (f_5* \; o \; f_5°) = f_5* \; (f_5° \; (D'_1, \ldots, D'_m))$ die Nacheinanderausführung von $f_5*:=(D'_1, \ldots, D'_m) \rightarrow (D''_1, \ldots, D''_z)$ nach $f_5°:=(D_1, \ldots, D_n) \rightarrow (D'_1, \ldots, D'_m)$, dann ist:

Der Vektor $v := (v_1, \ldots, v_k, \ldots, v_m)$ mit:
$v_k = D_{i_k}$ falls $f_5°: (D_1, \ldots, D_n) \rightarrow (D'_1, \ldots, D'_m)$, und $D'_k = D_{i_k}$ mit $i_k \in In^a$,
$v_k = D'_k = 0$ sonst .

Die Matrix $A_{m,z} := (a_{i,j})$ mit

$a_{i,j} = 1$	*falls $f_5*: (D'_1, \ldots, D'_m) \rightarrow (D''_1, \ldots, D''_z)$, und D'_i in $mix_j (D'_1, \ldots, D'_k) = D''_j$ enthalten ist, also $mix_j (D'_1, \ldots, D'_x, \ldots, D'_k)$ und $D'_x = D'_i$. und $1 \le x \le k$*
$a_{i,j} = 0$	*sonst.*

Beispiel:



$v=(D_2,D_2,D_3)$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$f_5 = (f_5^* \circ f_5^\circ) \text{ beschrieben durch } v \cdot A = (D_2, D_2, D_3) \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} = (D_2,\ 0,\ D_2 + D_3)$$

Ebenso wie die Mischfunktion f_5^* kann auch die Verteilfunktion f_5° als Matrix beschrieben werden. Bei f_5^* ergibt sich aus der Forderung der reinen Mischfunktion, daß in jeder Zeile nur maximal eine von Null verschiedene Ziffer stehen darf. Entsprechend gilt für die reine Verteilfunktion f_5°, daß sie beschrieben als Matrix B in jeder Spalte nur maximal eine von Null verschiedene Ziffer stehe haben darf.

Bezogen auf das obige Beispiel gilt dann:

$$f_5(D_1, D_2, D_3) = (f_5^* \circ f_5^\circ)(D_1, D_2, D_3) = (D_1, D_2, D_3) \cdot B \cdot A =$$

$$(D_1, D_2, D_3) \cdot \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Die Assoziativität der Matritzenmultiplikation erlaubt es, f_5 durch eine einzige Matrix $M=B \cdot A$ zu spezifizieren. Durch die Matrizenschreibweise von f_5^* und f_5° lassen sich diese so zu beliebigen Folgen von Mischen und Verteilen kombinieren. Ersetzt man die Besetzung der Mischmatrizen mit Einsen durch andere natürliche Zahlen, kann eine Gewichtung beim Mischen in Anteilen beschreiben werden. Beispiele hierzu und weitere Beziehungen zwischen den Funktionen werden in [Alireza94] vorgestellt.

3.3.3 Eigenschaften und Nebenbedingungen der Verarbeitungsfunktionen

Die Funktionstypen $f_{1...5}$ sind bisher nur als Veränderung oder Vervielfältigung von Daten beschrieben worden. Betrachtet man die Anwendungen der Verarbeitungsfunktionen im Modell, so erkennt man, daß sie in die Übertragung der Mediendaten integriert sind, und somit Folgen von Einzeldaten bearbeiten. Die Funktionen der Typen $f_{1...5}$ werden, solange Daten übertragen werden, fortlaufend auf die einzelnen Elemente angewandt.

An dieser Stelle sind nur zwei allgemeine Ausprägungen der Funktionstypen $f_{1...4}$ zu unterscheiden. Die eine Ausprägung ist Filtern des Datenstroms, die andere Verstärken. Während beim Filtern nur ein definierter Teil des Datenstroms ausgegeben wird, wird beim Verstärken eine definierte Veränderung des gesamten Datenstroms vorgenommen. Beide Ausprägungen können in jedem Funktionstyp auftreten. Zusammen mit dem Funktionstyp f_5 , Verteilen und Mischen, können damit prinzipiell alle Funktionen auf Multimediadaten beschrieben werden.

Betrachtet man die Beziehungen zwischen den Funktionstypen und den Anwendungskategorien, erkennt man, daß sich die Funktionstypen $f_{1...4}$ direkt den Anwendungskategorien 1 bis 4 zuordnen lassen. Der Funktionstyp f_5 bietet die Möglichkeit zwei Anwendungskategorien miteinander zu verbinden. Aus den Anwendungskategorien lassen sich Nebenbedingungen an die Verarbeitungsfunktionen ableiten. Die markantesten Nebenbedingungen liegen einerseits in den Zeitbeschränkungen, welchen Multimediaanwendungen unterliegen; ande-

rerseits in der Sequentialität mit der die Daten verarbeitet werden. Beide Nebenbedingungen haben somit Einfluß auf die mögliche Komplexität der Verarbeitungsfunktionen. Sowohl die Menge der Daten als auch der Verarbeitungsaufwand werden beschränkt. Es soll hier allerdings kein Maß für die Komplexität berechnet werden, sondern statt quantitativer Analysen werden nur qualitative Aussagen gemacht. Daher wird im folgenden nicht mehr von der Komplexität der Verarbeitungsfunktionen sondern von ihrer Mächtigkeit gesprochen.

Den stärksten Einschränkungen unterliegt der Funktionstyp f_1, der den Live–Anwendungen der Kategorie 1 zugeordnet ist. Hier müssen die Daten strikt sequentiell verarbeitet werden, ein Bezug ist nur auf frühere Daten, nicht auf folgende möglich. Gegenüber der Datenquelle (Perzeption) muß eine ausreichende Aufnahmegeschwindigkeit der Eingabe der Verarbeitungsfunktion gewährleistet sein, gegenüber der Datensenke (Präsentation) muß eine ausreichende Ausgabegeschwindigkeit der Verarbeitungsfunktion erbracht werden.

Der Funktionstyp f_2 unterliegt ebenfalls der strikten Serialität der Datenverarbeitung und muß eine ausreichende Aufnahmegeschwindigkeit an der Eingabe gewährleisten. An der Ausgabe bestehen allerdings keine festen Anforderungen. Das Spiegelbild des Funktionstyps f_2 ist der Funktionstyp f_3. Für ihn gilt die strikte Sequentialität und die ausreichende Geschwindigkeit an der Ausgabe, während an der Eingabe keine festen Anforderungen bestehen.

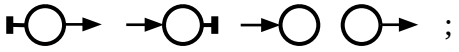
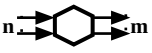
Die geringsten Einschränkungen bestehen beim Funktionstyp f_4, der weder an der Eingabe noch an der Ausgabe festen Anforderungen unterliegt. Daher können bei diesem Typ beliebig mächtige Funktionen durchgeführt werden.

Aus diesen Nebenbedingungen läßt sich ableiten, daß die mögliche Mächtigkeit der Verarbeitungsfunktionen von f_1 zu f_4 zunimmt. Genaueres darüber kann erst gezeigt werden, wenn ein Datenmodell mit integriertem Zeitbezug vorliegt. Die weiteren Untersuchungen liegen aber außerhalb des Rahmens dieser Arbeit.

Die Nebenbedingungen an f_5 können nicht so eindeutig festgelegt werden, da f_5 mit allen Grundfunktionen verbunden werden kann. Je nach Verbindung gelten dann die entsprechenden Anforderungen wie an die Funktionstypen $f_{1...4}$. Aus dem Aufbau von f_5 kann man aber schließen, daß für die Verteilung keine Einschränkungen existieren. An die Mischfunktion f_5* werden die Anforderungen entsprechend den Pfaden der Daten von der Quelle zur Senke wie bei den Funktionstypen $f_{1...4}$ gestellt.

3.3.4 Anwendungsbeschreibung

In diesem Abschnitt wird gezeigt, wie man mit Hilfe des Funktionenmodells verschiedene Anwendungen modellieren kann. Die Beschreibung der Anwendung erfolgt durch graphische Symbole. Ausgehend von den vier Anwendungskategorien werden die Anwendungen unter Berücksichtigung weniger Regeln aufgebaut. Diese Regeln sind:

1. Alle Funktionen haben eine feste Anzahl von (Daten–) Schnittstellen:
   a. Grundfunktionen genau eine  ;
   b. Funktionen der Typen $f_{1...4}$ genau zwei  ;
   c. Funktionen vom Typ f_5 in der Anwendung definiert. 

2. In einer Anwendung kann jede Grundfunktion durch eine ihrer Verbindung entsprechenden Verarbeitungsfunktion vom Typ ($f_{1...4}$) ersetzt werden, mit der wieder dieselbe Grundfunktion verbunden ist.
3. Um zwei Anwendungen, die eine gemeinsame Grundfunktion verwenden sollen, miteinander zu verbinden, kann jede Grundfunktion durch eine Verarbeitungsfunktion vom Typ f_5 ersetzt werden, die wieder mit derselben Grundfunktion verbunden ist.

Als Anwendungsbeispiele werden die Teilsysteme des Karaoke–Beispiels herangezogen. Dafür werden in Abbildung 13 zunächst einfache Aufzeichnungs– und Wiedergabesysteme definiert. Das Aufzeichnungssystem entspricht einer Anwendung der Kategorie 2. Soll es die Möglichkeit der Aussteuerregelung haben, so muß eine Verarbeitungsfunktion vom Typ f_2 verwendet werden. Die Ausprägung der Verarbeitungsfunktion ist eine Verstärkung. Das Wiedergabesystem entspricht einer Anwendung der Kategorie 3. Es sollte mit einer Lautstärkeregelung, also einer Verstärkung, ausgestattet sein. Der Typ der einzusetzenden Verarbeitungsfunktion ist f_3 . Das Wiedergabesystem ist in Abbildung 14 dargestellt.

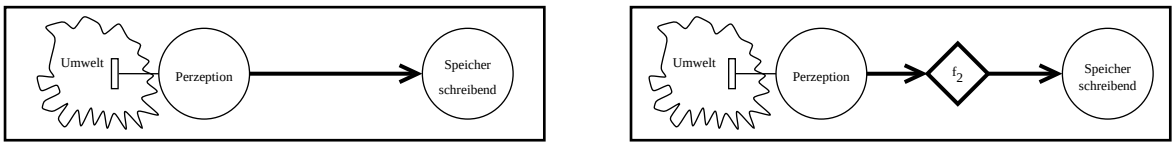


Abb. 13 Aufzeichnungssystem ohne und mit Aussteuerregelung (Verstärkung)

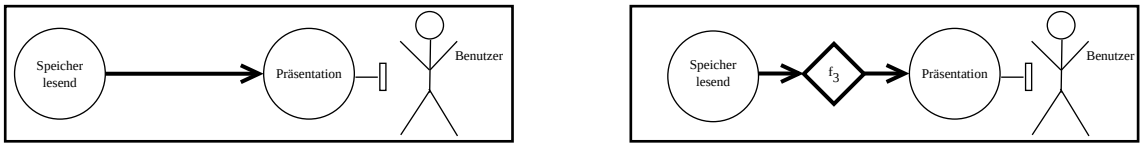


Abb. 14 Wiedergabesystem ohne und mit Verstärkung

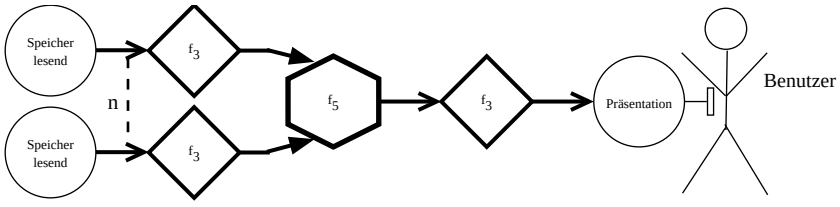


Abb. 15 Lesersystem ohne Live–Präsentation

Etwas umfangreicher wird das Modell des Lesersystems, für das zwei Varianten angeboten werden. Die eine ohne und die andere mit der Möglichkeit der Live–Präsentation. Im Lesersystem tritt zum erstenmal eine Verarbeitungsfunktion vom Typ f_5 auf (Abb. 15). Bei der Integration der Live–Präsentation kann dafür eine weitere Verarbeitungsfunktion vom Typ f_5 eingeführt werden oder, wie in Abbildung 16 dargestellt, die Verarbeitungsfunktion erweitert werden. Das umfangreichste System aus dem Karaoke–Beispiel ist das Autorensy-

stem. Es integriert Aufzeichnungs– und Wiedergabesysteme sowie die Bearbeitung von Mediendaten. Zur Verbindung der Teilsysteme sind drei Verarbeitungsfunktionen von Typ f_5 nötig.

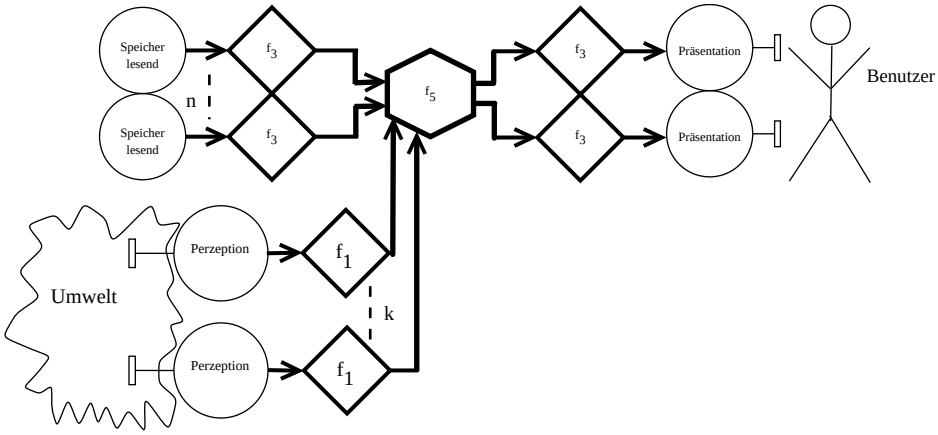


Abb. 16 Lesersystem mit Live–Präsentation

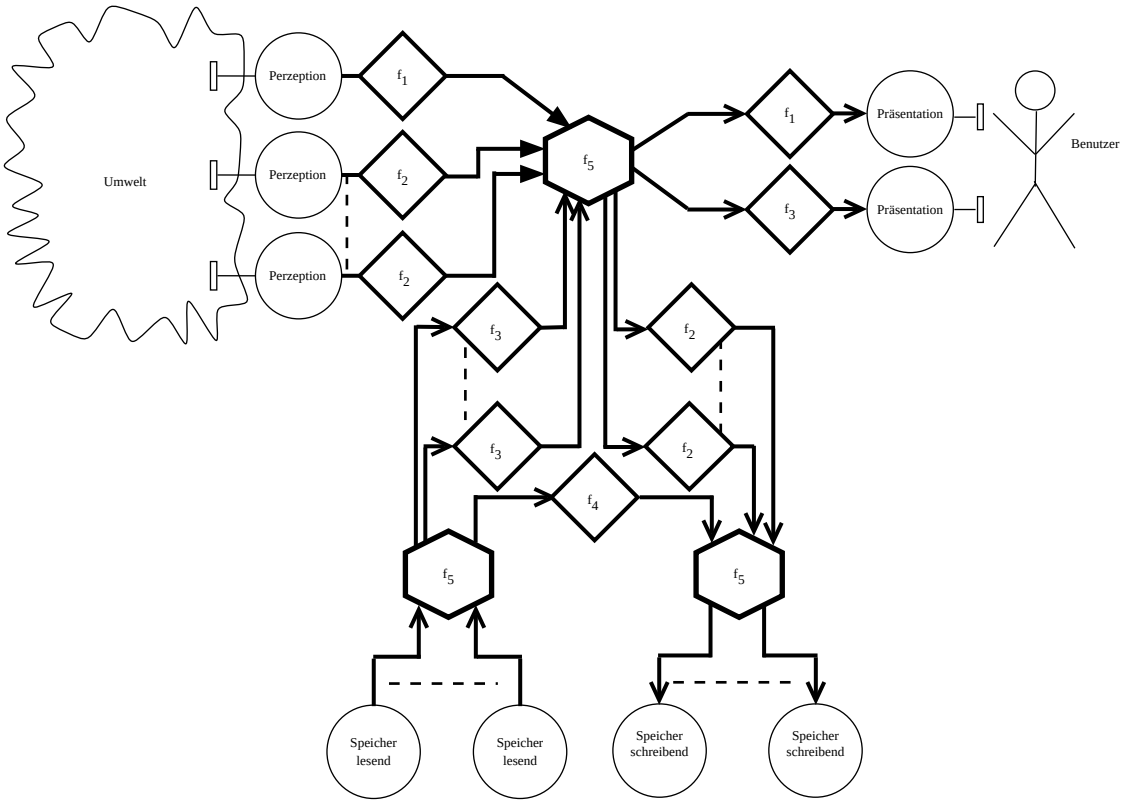


Abb. 17 Autorensystem

3.3.5 zusammengesetzte Verarbeitungsfunktionen

Bei der graphischen Beschreibung von Anwendungen unter Verwendung der im obigen Abschnitt eingeführten Regeln, können Ketten von Verarbeitungsfunktionen gleichen Typs entstehen, die zwei Grundfunktionen oder eine Grundfunktion und eine Verarbeitungsfunktion vom Typ f_5 miteinander verbinden. Es kann zum Beispiel folgendes Bild entstehen:

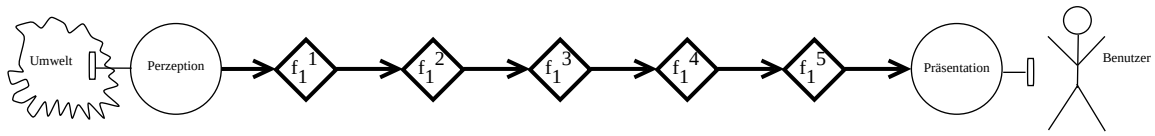
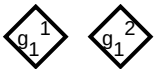


Abb. 18 eine Kette von Funktionen

In diesem Beispiel sind fünf Verarbeitungsfunktionen vom Typ f_1 miteinander verbunden. Diese Funktionen werden nacheinander auf die Daten der Perzeption angewandt:

$$f_1^5 (f_1^4 (f_1^3 (f_1^2 (f_1^1 (D))))) = f_1^5 \text{ o } f_1^4 \text{ o } f_1^3 \text{ o } f_1^2 \text{ o } f_1^1 (D)$$

Da die Verarbeitungsfunktionen sowohl Bausteine für die Integration kontinuierlicher Medien liefern als auch Elemente der Verteilung darstellen sollen, ist es erforderlich, die Verarbeitungsfunktionen zusammenzufassen, die an einem Ort oder zusammen durchgeführt werden sollen. Voraussetzung dafür ist die Typgleichheit der Verarbeitungsfunktionen.

Im obigen Beispiel sollen jetzt die Funktionen f_1^1 und f_1^2 sowie f_1^4 und f_1^5 zusammengefaßt werden. In der Abfolge der Funktionen entstehen somit neue Funktionen g in folgender Weise: $g_1^2 \text{o } f_1^3 \text{o } g_1^1$ mit $g_1^1 = f_1^2 \text{ o } f_1^1$ und $g_1^2 = f_1^5 \text{ o } f_1^4$. In der graphischen Darstellung können nun g_1^1 und g_1^2 als neue Funktionen  geschrieben oder, wie in der folgenden Abbildung, als Zusammenfassung hervorgehoben werden.

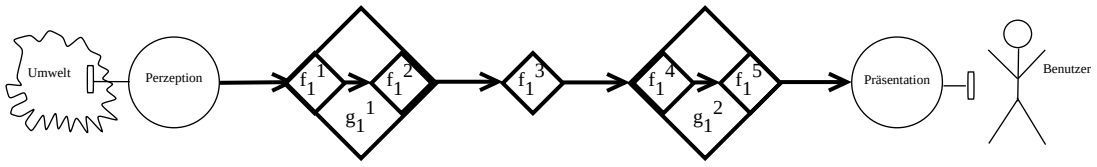


Abb. 19 zusammengesetzte Verarbeitungsfunktionen

Neben den Verarbeitungsfunktionen der Typen f_1 bis f_4 können auch die Funktionen vom Typ f_5 zusammengesetzt werden. Eine einfache Zusammensetzung von Funktionen vom Typ f_5 ist die bereits zuvor beschriebene Verkettung von f_5^* und f_5°. Unter den gleichen Voraussetzungen, der Anschlußkompatibilität $Out(f_5^\circ)=Out^a(f_5^\circ)=In(f_5^*)$, kann eine einfache Erweiterung vorgenommen werden. Diese Verkettung ist in Abbildung 20 dargestellt. Die Verkettungsoperationen lassen sich auf die entsprechenden Matritzenoperationen zurückführen.

Prinzipiell können alle zusammengesetzten Funktionen auch wieder zerlegt werden und umgekehrt. Interessant kann die Zerlegung vor allem für die Funktionen vom Typ f_5^* werden, da durch eine Zerlegung die Anforderungen der Anschlußkompatibilität $Out(f_5^\circ)=Out^a(f_5^\circ)=In(f_5^*)$ für die Verkettung von f_5^* und f_5° heruntergesetzt werden kann. Ein Beispiel dazu enthält die folgende Abbildung.

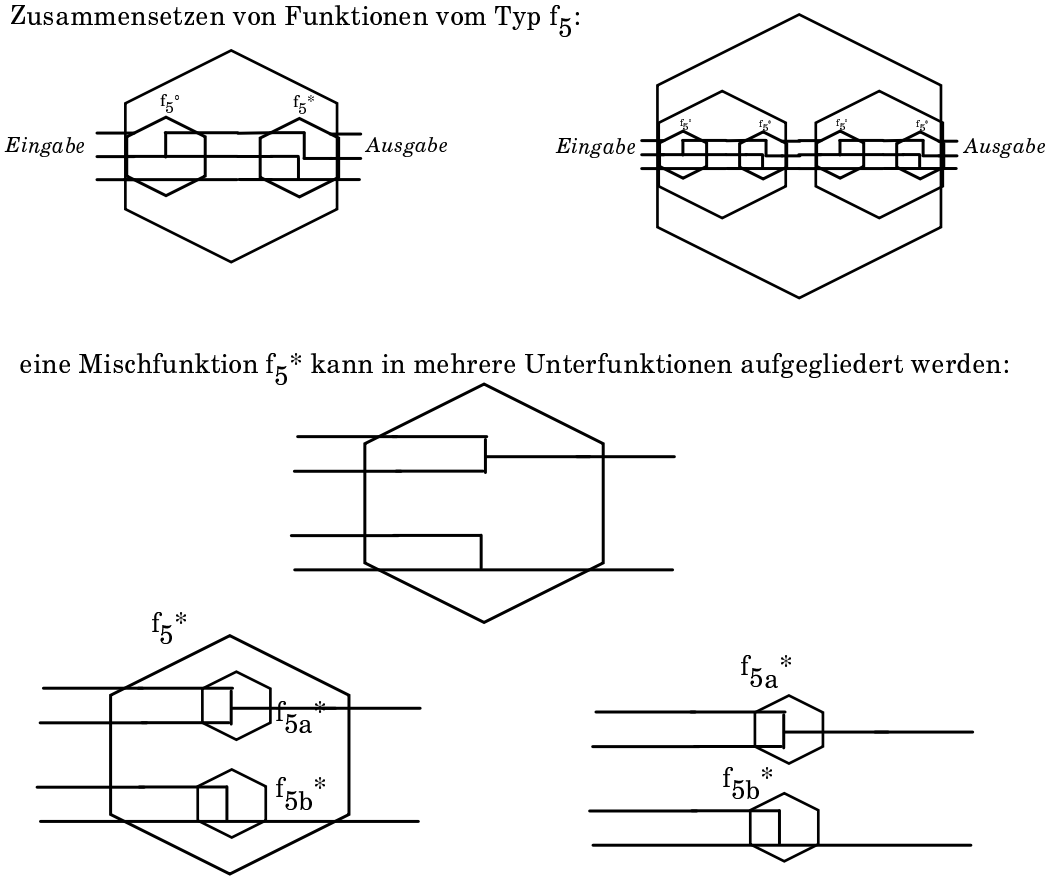


Abb. 20 zusammengesetztes und zerlegtes Verteilen und Mischen

Die Entscheidung, welche Verarbeitungsfunktionen zusammengefaßt werden sollen, kann nicht generell vorgenommen werden, sondern bleibt stets dem Anwender überlassen. Generell gilt, daß eine Funktion, auch eine zusammengefaßte, als ein Baustein beschrieben wird, der nur als Ganzes in die Anwendung integriert und bei der Verteilung nur als Ganzes einem Rechner zugeordnet werden kann.

3.4 Umgebungen für Funktionen

Die Grundfunktionen und die Verarbeitungsfunktionen f_1 bis f_5 werden zur Realisierung in Umgebungen eingebettet. Es werden vier Umgebungstypen unterschieden; ein Typ für jeden Grundfunktionstyp und ein Umgebungstyp für die Verarbeitungsfunktionen f_1 bis f_5. Dabei gelten folgende Grundsätze:

- jede Funktion läßt sich einem bestimmten Umgebungstyp zuordnen;
- jede Funktion wird in einer Umgebung realisiert;
- jeder Rechner, der an einer verteilten Anwendung teilnimmt, realisiert eine Umgebung jeden Typs.

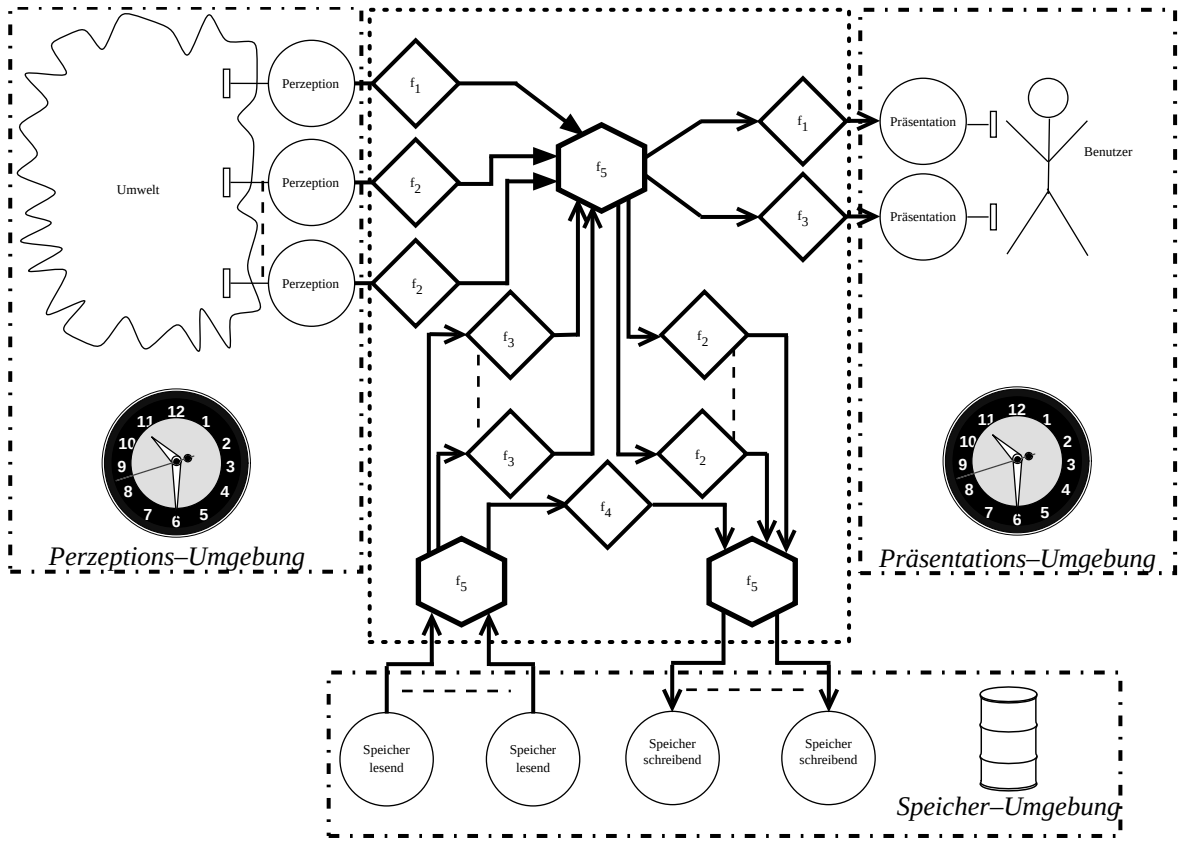


Abb. 21 Funktionen und Umgebungen am Beispiel des Autorensystems

3.4.1 Einbettung der Grundfunktionen

Die Unterscheidung der Umgebungen für Grundfunktionen von den Verarbeitungsfunktionen f_1 bis f_5 bietet sich an, da die Grundfunktionen und ihre Umgebungen die Randbedingungen für die Verarbeitungsfunktionen f_1 bis f_5 festlegen. Die Umgebungen der Grundfunktionen gliedern sich zunächst in die Perzeptions–, die Präsentations– und die Speicherumgebung, in einem zweiten Schritt erkennt man, daß sich die Perzeptions– und die Präsentationsumgebung gemeinsam beschreiben lassen. Diese beiden Umgebungen sind durch die strikte Abhängigkeit von der Zeit charakterisiert. Die Funktionen Perzeption und Präsentation sind im klassischen Sinn Realzeitfunktionen, da sie an ihren Schnittstellen zur Umwelt und zum Benutzer Daten mit einer festen Rate einlesen oder ausgeben. Die Umgebungen müssen daher ein Zeitbezugssystem enthalten. Wie sich aus Studien der Zeitanforderungen und der Übertragung [Stein94] [Ferrari91] ergibt, können geeignete Umgebungen diese harten Zeitbedingungen an der Schnittstelle zu anderen Funktionen puffern.

Zur Realisierung der Umgebungen für die Perzeption und die Präsentation werden verschiedene Wege gegangen. Einerseits wird die Umgebung in der Hardware mit integriert und beeinflußt so auch die Schnittstelle der Bausteine (zum Beispiel die Einschubkarten für die erste Ebene der Unterstützung, vgl. DVI); andererseits werden Realzeiterweiterungen für die verwendeten Betriebssysteme vorgenommen [Fisher92] [CouBaRo93].

Die Umgebungen für die Speicherfunktionen integrieren die klassischen Datenverwaltungssysteme. Die Speicherfunktionen für kontinuierliche Medien müssen aber deutlich von den Schnittstellen der Datenverwaltungssysteme unterschieden werden. Datenverwaltungssysteme stellen vor allem Operationen zur Beschreibung und Ordnung sowie zur Auswahl von Daten bereit. Der tatsächliche Zugriff wird in der Multimediagrundfunktion realisiert.

Den Umgebungen für die Grundfunktionen ist gemeinsam, daß sie auf jedem Rechner neu realisiert werden müssen, und es keine genormte Schnittstelle für sie gibt, im allgemeinen sogar auch keine klare Definition der Umgebung festgelegt wird. Daher können auch keine Standardschnittstellen der Grundfunktionen zu den Umgebungen angegeben werden. Ziel ist vielmehr die Schnittstellen der Funktionen untereinander so zu beschreiben, daß sie als Bausteine verwendet werden können. Damit kann die Art der Realisierung der Umgebung und die Einbettungsweise der Grundfunktionen vor dem Benutzer verborgen werden. Wird eine Grundfunktion für eine Anwendung instanziiert, kann die Umgebung zu diesem Zeitpunkt entscheiden, ob die Grundfunktion realisiert werden kann, das heißt, ob alle nötigen Ressourcen für sie bereitgestellt werden können.

3.4.2 Einbettung der Verarbeitungsfunktionen

Vergleicht man die Abbildung 17 mit der Abbildung 2, so kann man die Umgebung der Verarbeitungsfunktionen f_1 bis f_5 als Netz betrachten, das die Übertragung zwischen den Grundfunktionen realisiert. Im Unterschied zu den Kommunikationsnetzen, die aus der Datenverarbeitung bekannt sind, ist in diesem Netz eine benutzergesteuerte Veränderung der transportierten Daten möglich. Bei der Betrachtung des Netzaufbaus zeigt sich, daß die typischen Netzfunktionen der Verteilung und Weiterleitung von den Verarbeitungsfunktionen f_5 vorgenommen werden, die die Netzknoten darstellen. Die Verarbeitungsfunktionen f_1 bis f_4 sind dagegen die netzuntypischen Veränderungspunkte.

Die Ähnlichkeit des Modellierungskonzepts der Verarbeitungsfunktionen f_5 mit Netzknoten legt es nahe, sie zumindest teilweise in die vorhandenen Kommunikationssysteme zu integrieren, oder vermittels diesen zu realisieren. Die Verteilungsfunktion $f_5°$ kann beispielsweise durch Multicast–Verbindungen realisiert werden.

Die Umgebung der Verarbeitungsfunktionen f_1 bis f_5 kann im Gegensatz zu den Grundfunktionsumgebungen nicht weiter beschrieben werden, sondern ist allgemein das Betriebssystem. Auch hier gilt, daß das Bausteinkonzept und die Schnittstellen der Funktionen von der Realisierung der Einbettung abstrahieren.

Je nachdem, ob die Umgebung für alle Funktionen gleich ist, das heißt alle Funktionen sind auf dem selben Rechner realisiert, oder, ob die Funktionen in verschiedenen Umgebungen auf unterschiedlichen Rechnern realisiert sind, kann man für die Kommunikation lokale Mechanismen einsetzen, oder man muß die Kommunikation im Rechnernetz verwenden.

3.4.3 Kopplung von Umgebungen

Die Kopplung von Umgebungen beschreibt nicht das Zusammenspiel der verschiedenen Umgebungstypen, sondern die Möglichkeit einen Umgebungstyp in zwei oder mehrere Umgebungen zu spalten. Das ist immer dann der Fall, wenn eine Anwendung tatsächlich über mehrere Rechner verteilt ist. Bausteine, die in Umgebungen unterschiedlichen Typs und auf verschiedenen Rechnern liegen, verwenden die Kommunikation im Rechnernetz und stellen keine Kopplung der Umgebungen dar.

Umgebungen sind miteinander gekoppelt, wenn sie ein gemeinsames Bezugssystem zur Zeit oder zur Datenhaltung haben. Man unterscheidet zwischen der engen und der losen Kopplung. In der engen Kopplung benutzen alle Funktionen eines Umgebungstyps unabhängig von der Verteilung auf die Rechner das selbe Bezugssystem zur Zeit oder zur Datenhaltung. Im allgemeinen heißt das, daß sie eine gemeinsame Uhr oder ein gemeinsames Filesystem oder eine gemeinsame verteilte Datenbank haben. In der losen Kopplung hat jede Umgebung ihr eigenes Bezugssystem zur Zeit oder zur Datenhaltung, hat darüber hinaus aber Kenntnisse über die Bezugssysteme der gekoppelten Umgebungen. Diese Kenntnisse können zum Beispiel die Differenz zwischen den Uhren sein, die in längeren Abständen immer wieder neu bestimmt wird, oder die Namen von Fileservern oder Datenbankagenten der gekoppelten Umgebungen.

4 Ein Datenmodell für die Verarbeitung kontinuierlicher Medien

In diesem Kapitel wird gezeigt, daß die Handhabung der Zeit für die Daten multimedialer Systeme auf einen einfachen Ansatz unter der Verwendung von Ganzzahlarithmetik zurückgeführt werden kann. Grundlage dafür ist, daß die Zeit nicht nur eine Restriktion der Datenpräsentation ist, sondern daß jeder Datenwert kontinuierlicher Medien spezielle zeitabhängige Eigenschaften hat. Nach einer kurzen theoretischen Betrachtung von Folgen zeitabhängiger Daten wird ein Datenmodell durch Struktur und Operationen des Modells definiert. Die Integration des Werts und der zeitabhängigen Eigenschaften in eine gemeinsame Repräsentation ist die Aufgabe des entwickelten Datenmodells. Auf der Grundlage dieses Datenmodells werden die im vorigen Kapitel eingeführten Funktionen näher beschrieben.

4.1 Datenmengen kontinuierlicher Medien

Perzeptionsmedien bestimmen ihre Daten zu einem bestimmten Zeitpunkt aus Parametern der Umwelt. Die ***Werte*** der Parameter können in der Zeit davor und danach variieren. Daher gibt es zu jedem Wert einen ***Zeitpunkt***, zu dem er entstanden ist. Wie aus Kapitel 1 bekannt, werden zeitabhängige und zeitunabhängige Medien unterschieden. Die Unterscheidung zwischen zeitabhängigen Medien und zeitunabhängigen Medien kann auf die Daten übertragen werden. Sie lassen sich danach in zwei Klassen unterteilen, die zeitabhängigen Daten und die zeitunabhängigen Daten. Die Klasse der zeitabhängigen Daten enthält außer dem Entstehungszeitpunkt eines Datums auch seine ***Gültigkeitsdauer***.

4.1.1 Integrierte Beschreibung der Eigenschaften von Einzeldaten

Ein Mediendatum eines kontinuierlichen Medii ist durch die drei Eigenschaften Wert, Zeitpunkt und Dauer gekennzeichnet. Alle Parameter haben direkten Einfluß auf die repräsentierte Information. Diese Definition ist konform zu [Herrtw90] und [Gibbs94]. Eine wichtige Information, die aus dem Zeitpunkt abgeleitet werden kann, ist die Reihenfolge der Daten einer Datenmenge eines zeitabhängigen Medii.

Einzeldatum:

Ein einzelnes Mediendatum, ein Einzeldatum d, setzt sich aus dem Tripel (w,T,t) zusammen. Dabei gelten folgende Definitionen und Schreibweisen:

- *der Wert w:* $d.w$;
- *der Zeitpunkt T:* $d.T$;
- *die (Gültigkeits–)Dauer t:* $d.t$.

Die folgenden Annahmen über die Wertemengen der Datentripel werden vorausgesetzt:

- $d.w \in W$ *und auf der Menge W ist die Gleichheit definiert und entscheidbar;*
- $d.T \in \mathsf{Zeit}$ *und* $d.t \in \mathsf{Zeit}$ *und auf der Menge* Zeit *sind Gleichheit und eine Ordnungsrelation $<$ definiert und entscheidbar.*

Für die grundlegenden Aussagen, die in diesem und dem folgenden Abschnitt getroffen werden, sind keine weiteren Einschränkungen nötig. Eine tiefergreifende Analyse der Zeit und der Zeitdomäne für Multimedia wird in [MiViFe95] vorgestellt. Weitere interessante Ergänzungen liefern [Milner83] und [NRSV90]. In [Tenn95] werden die Vor– und Nachteile von diskreten und kontinuierlichen Domänen diskutiert.

Beispiele für Daten kontinuierlicher Medien sind einzelne (Audio–) Samples oder auch die Sektoren einer Audio CD sowie Einzelbilder oder die Bildergruppen von MPEG (kurz: MPEG–GOP). Insbesondere bei letzteren wird wegen ihrer unterschiedlichen Anzahl von Bildern verständlich, warum die Dauer eine eigene Eigenschaft der Daten ist und nicht schon allgemein durch den Datentyp oder die Codierung der Werte vorgegeben ist.

Die Präsentation oder Interpretation von Multimediadaten muß unter Berücksichtigung der Zeitparameter erfolgen. Ein Wert erhält zum Zeitpunkt T_0 seine Gültigkeit, d. h. seine Präsentation beginnt. Die Präsentation dauert eine bestimmte Zeit, die zunächst unabhängig von der Gültigkeitsdauer ist, und endet zum Zeitpunkt $T_1 > T_0$. Das Verhältnis der Gültigkeitsdauer zur Präsentationsdauer ist anwendungsabhängig, und kann vom Autor als Gestaltungsmittel verwendet werden. Zur Vereinfachung seien zunächst die Präsentations– und die Gültigkeitsdauer gleich, später wird diese Annahme wieder fallenlassen.

Die Daten können mittels der Funktionen f_1 bis f_5 bearbeitet werden. Dabei wird im allgemeinen der Wert $d.w$ eines Medii verändert. Einige Funktionen können allerdings auch die Zeitparameter $d.T$ oder $d.t$ verändern. Dies wiederum kann indirekt Auswirkungen auf den Wert $d.w$ haben. Das Datenmodell wurde in [Fritzsche95] veröffentlicht.

4.1.2 Eigenschaften von Datenmengen

Aus der Definition der Einzeldaten kontinuierlicher Monomedien lassen sich Eigenschaften von Mengen dieser Einzeldaten ableiten. $\wp$ sei eine solche Menge. Im allgemeinen ist $\wp$ endlich; die Kardinalität $n=|\wp|$ von $\wp$ kann bei der Perzeption aber erst ex post angegeben werden, man sagt $\wp$ ist potentiell unendlich. Da $\wp$ aus Einzeldaten besteht, ist $\wp$ immer abzählbar. Die Elemente von $\wp$ lassen sich damit als Datenfolge $F = \langle d_i \rangle$ aufzählen. Dies ist die Aussage des nachfolgenden Lemma 1, das durch Konstruktion der Folge bewiesen wird.

Lemma 1

Die Menge $\wp$ der Einzeldaten eines zeitabhängigen oder kontinuierlichen Medii läßt sich zu einer Folge ordnen.

Beweis:
$\wp$ wird nach den Entstehungszeiten T der Werte geordnet. Da die Werte zu aufeinanderfolgenden Zeitpunkten gewonnen werden, gilt für alle d und d' aus $\wp$ die Voraussetzung:

$$d \neq d' \Rightarrow d.T \neq d'.T \wedge \left(d.T > d'.T \vee d.T < d'.T\right)$$

Ist $\wp$ endlich, kann die Konstruktion der Folge durch Minimum–Elimination nach d.T aus $\wp$ erfolgen. Ist $\wp$ potentiell unendlich, so muß die Perzeption zu einem Zeitpunkt T_0 begonnen haben. Damit gibt es einen Wert d_1 aus $\wp$ für den gilt:

$$T_0 \leq d_1.T \wedge \forall\, d \in \wp : d_1 \neq d \Rightarrow d_1.T < d.T$$

dann sei $F_1 = \langle d_1 \rangle$, $\wp_1 = \wp \setminus \{d_1\}$ und allgemein mit $\wp_0 = \wp$ und $1 \le i \le n$:

$$\left(d_i \in \wp_{i-1} \wedge \forall d \in \wp_{i-1}: d_i \neq d \Rightarrow d_i.T < d.T\right) \Rightarrow F_i = \langle d_1, \ldots, d_i \rangle, \wp_i = \wp \setminus \{d_1, \ldots, d_i\}$$

Analog zum endlichen Fall wird auch bei potentiell unendlichem $\wp$ die Folge durch Minimum–Elimination nach d.T konstruiert; in diesem Fall ist aber $\wp$ im allgemeinen bereits nach den Entstehungszeitpunkten geordnet gegeben. ■

Wie in Lemma 1 angegeben, haben verschiedene Daten im Normalfall auch verschiedene Zeitpunkte und lassen sich damit chronologisch ordnen. Durch Bearbeitung einer endlichen Datenmenge eines kontinuierlichen Medii kann es geschehen, daß verschiedene Daten den selben Zeitpunkt T erhalten. Für diese gilt nur die schwache Chronologie, entsprechend zu den folgenden Definitionen.

Def. schwach chronologisch

*Sei $F = \langle d_i \rangle_n$ eine Datenfolge eines kontinuierlichen Medii mit der Elementanzahl n und $1 \le i \le n$. Falls für $F = \langle d_i \rangle_n$ gilt, daß $d_i.T \le d_{i+1}.T$, für alle $1 \le i < n$, so heißt die Folge **schwach chronologisch** geordnet.*

Def. chronologisch

*Sei $F = \langle d_i \rangle_n$ eine Datenfolge eines kontinuierlichen Medii mit der Elementanzahl n und $1 \le i \le n$. Falls für $F = \langle d_i \rangle_n$ gilt, daß $d_i.T < d_{i+1}.T$, für alle $1 \le i < n$, so heißt die Folge **chronologisch** geordnet. Im Unterschied zu schwach chronologisch spricht man auch von streng chronologisch.*

Im folgenden steht häufig vereinfachend für $d_i.T$ nur noch T_i, für $d_i.t$ nur noch t_i und für $d_i.w$ nur noch w_i. Bezogen auf die Mengen von Daten eines kontinuierlichen Medii wird nach Lemma 1 überwiegend von Datenfolgen eines kontinuierlichen Medii gesprochen. Dazu wird eine Reihe von Normalformen entwickelt. Die erste Normalform ergibt sich aus der Konstruktion von Lemma 1.

Def. 1. Normalform der Datenfolge

Eine Datenfolge eines kontinuierlichen Medii ist genau dann in erster Normalform (1NF), wenn die Datenfolge chronologisch geordnet ist.

Im allgemeinen liegen die Daten eines kontinuierlichen Medii in 1NF vor. Diese chronologische Ordnung kann aber durch eine Bearbeitung verlorengehen. Sie sollte vor der Präsentation unbedingt wiederhergestellt werden, da man sonst stets die gesamte Datenfolge zur Verfügung haben muß. Das aber ist bei den oft sehr großen Datenmengen kontinuierlicher Medien praktisch nicht möglich. Das heißt, daß die erste Normalform der Datenfolge eine Voraussetzung für die Präsentation ist. Dennoch sind die Anforderungen an eine Präsentationsfunktion für Datenfolgen in 1NF hoch, da in einer Datenfolge in erster Normalform immer noch sowohl Überschneidungen der Gültigkeitszeiten der Werte als auch Lükken zwischen diesen auftreten können. Daher muß eine Präsentationsfunktion für Datenmengen in erster Normalform im schlimmsten Fall in der Lage sein, alle Werte gleichzeitig zu

präsentieren und die entsprechenden Mischfunktionen zu berechnen, sowie die verschiedenen Endzeitpunkte der Präsentation der Einzelwerte bestimmen und einhalten. Darüber hinaus muß sie ein Verfahren zur Überbrückung von den Zeiten bereitstellen, in denen kein Wert gültig ist.

Die folgenden Definitionen beziehen sich auf einige ausgezeichnete Formen der Folgen in 1NF. Daraus werden mögliche zeitliche Verhältnisse von direkt aufeinanderfolgenden Werten einer Folge in erster Normalform abgeleitet. Eine besondere Form der Datenfolge entsteht, wenn die Distanzen der Zeitpunkte und die Dauern aller Daten gleich sind. Diese Gleichmäßigkeit wird durch den Begriff Datenstrom ausgedrückt. In [Herrtw90] heißt eine Datenfolge mit dieser Eigenschaft periodischer Datenstrom.

Def. Datenstrom

Sind in einer Datenfolge in erster Normalform für alle $1 \le i < n$

1. *die Differenzen $T_{i+1} - T_i = \Delta T$ alle gleich groß, d. h. die einzelnen Entstehungszeitpunkte äquidistant, und*

2. *die Gültigkeitsdauern $t_i = t_n$ der Werte alle gleich lang,*

*so nennt man die Folge **Datenstrom**.*

Ein Datenstrom in dieser Sicht entspricht nicht notwendigerweise einer Datenübertragung mit fester Datenrate. Für einen Datenstrom lassen sich aber präzise Aussagen bezüglich der erforderlichen Verarbeitungs– oder Übertragungsgeschwindigkeit treffen. Zum Beispiel muß jeder Wert $d_i.w$ spätestens zum Zeitpunkt $d_i.T$ bereitgestellt sein, das heißt es müssen mindestens $1/\Delta T$ Daten pro Zeiteinheit verarbeitet werden. Beispiele für Datenströme sind die Sample-Folgen in der PCM–Technik, die Sektorfolgen auf einer Audio–CD oder die Bilderfolgen eines Films. Dabei treten zwischen den Gültigkeitszeiten der Bilder eines Films Lükken auf, da in diesen Lücken der Film zum nächsten Bild weitertransportiert werden muß.

Def. kontinuierliche Folge, diskrete Folge, Überschneidung

Eine chronologische Folge mit der Eigenschaft, daß

- *für alle $1 \le i < n$ gilt:*
  $T_i + t_i = T_{i+1}$, *heißt **kontinuierliche Folge**,*
- *für alle $1 \le i < n$ gilt:*
  $T_i + t_i < T_{i+1}$, *heißt **diskrete Folge**,*
- *für alle $1 \le i < n-2$ gilt:*
  $T_{i+1} < T_i + t_i < T_{i+2} < T_{i+1} + t_{i+1}$, *heißt **einfache Überschneidung**.*

Eine kontinuierliche Folge $F = \langle d_i \rangle_n$ bildet in Bezug auf ihr Wert–Zeitverhalten, beschrieben durch die Gültigkeitsdauern t_i und die Entstehungszeitpunkte T_i, ein Zeitkontinuum, da zu jedem $t' \in [T_1, .., T_n + t_n]$ sofort ein eindeutiger Wert w_i angegeben werden kann.

Da hier statt der technischen mehr die logischen Gesichtspunkte in den Vordergrund gerückt werden sollen, wird von den technischen Details, wie der Austastungslücke beim Videosignal oder dem Filmtransport bei der Projektion abstrahiert und die entsprechenden Folgen können auch als kontinuierlich betrachtet werden, da die Präsentationsfunktionen diese Details verbergen. Ist die Folge kontinuierlich, so ergibt sich bei der Präsentation der Eindruck einer natürlichen, zusammenhängenden Wiedergabe.

Überschneidungen und Lücken treten auch logisch auf und kennzeichnen eine gestalterische Idee. Sind die Lücken in einer diskreten Folge so groß, daß sie deutlich wahrnehmbar sind, so entsteht zum Beispiel bei Einzelbildern oder kurzen Bildfolgen ein Stroboskopeffekt. Überschneidungen liefern dagegen einen weichen Übergang, es liegt eine Art Halleffekt oder eine Überblendung vor. Abbildung 22 stellt die drei Eigenschaften für einen Datenstrom graphisch dar.

Sowohl für die diskrete Folge als auch für die einfache Überschneidung existiert eine kontinuierliche Folge, die die gleiche Präsentation beschreibt. Zwei Folgen, die die gleiche Präsentation beschreiben, heißen einander *äquivalent.*

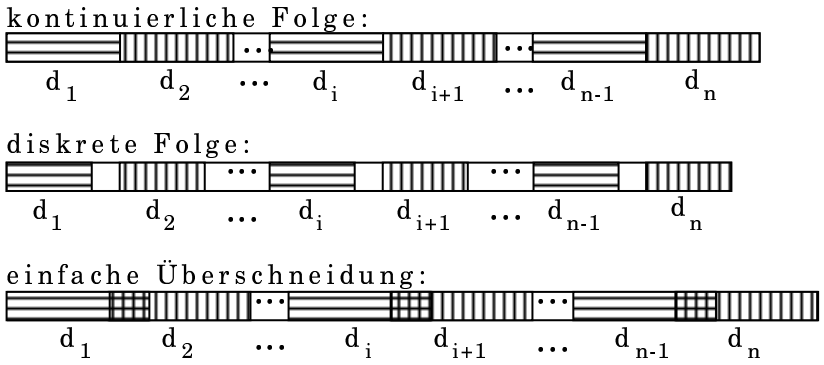


Abb. 22 Beispiele für Datenströme mit möglichen Lücken und Überschneidungen

In einer diskreten Folge treten während der Präsentation in den Zeiten zwischen zwei aufeinanderfolgenden Datenwerten undefinierte Werte auf. Da an einem Präsentationsgerät grundsätzlich kein undefinierter Wert dargestellt werden kann, muß für diesen Fall festgelegt werden, welcher Wert dargestellt werden soll. Es sind zwei Varianten möglich:

1. der zuletzt dargestellt Wert $d_i.w$ wird weiter dargestellt;
2. ein voreingestellter Standardwert $d.w_{stand}$ wird dargestellt.

Im Fall 1 bedeutet das einerseits geringen Aufwand, andererseits aber eine Veränderung der Daten, denn es gilt dann für $t_i=T_{i+1} - T_i$. Im Fall 2 muß dagegen der Standardwert definiert werden. Beide Varianten treten in existierenden Systemen auf, wofür noch einmal die technischen Verfahren als Beispiel herangezogen werden. Zwischen den (Halb–)Bildern des Fernsehens gibt es eine Lücke, die dazu dient, den Elektronenstrahl an den linken oberen Punkt zurückzusetzen. Während dieser sogenannten Austastungslücke wird der Strahl dunkelgetastet und erhält somit einen festen Wert. In der Präsentation aber, das heißt auf dem Schirm, bleibt durch Nachleuchten der Beschichtung das alte Bild erhalten, bis es überschrieben wird, seine Präsentationszeit verlängert sich also. Bei der Filmprojektion ist in der

Zeit des Bildwechsels, zu der der Film sich bewegt, der Lichtstrahl durch eine Blende abgedeckt. In diesen Sekundenbruchteilen wird nichts auf die Leinwand projiziert, oder genauer: das Bild ist ganz schwarz.

Die beiden Fälle zur Ersetzung der Lücken geben offensichtlich einen Konstruktionshinweis für die äquivalente kontinuierliche Folge. Der Vorteil des Falls 1 ist, daß sich das Datenvolumen nicht verändert. Im Fall 2 dagegen müßte bei genauer Beschreibung das doppelte Datenvolumen verbraucht werden. Eine Möglichkeit, daß die Daten nicht unbedingt verändert werden müssen, besteht, wenn nur das Verfahren und eventuell der Standardwert festgelegt sind und vom Präsentationssystem die Lücken eigenständig gefüllt werden, wie bei den o. g. Beispielen. Beiden Fällen gemeinsam ist, daß die diskrete Folge durch eine kontinuierliche Folge beschrieben werden kann. Diese Tatsache beschreibt der folgende Satz.

Satz 1 "Keinen Wert gibt es nicht"

Eine diskrete, chronologische Datenfolge $F = <d_i>_n$ kann durch eine äquivalente, kontinuierliche, chronologische Folge $F' = <d'_j>_m$ dargestellt werden, wenn die Ersetzung für die undefinierten Werte bekannt ist.

Beweis
durch Konstruktion von $F' = <d'_j>_m$:
O. B. d. A. ist das Ersetzungsverfahren das Einsetzen von Standardwerten.

F': $d'_1.w := d_1.w$, $d'_1.T := d_1.T$, $d'_1.t := d_1.t$
der erste Datenwert wird mit den Zeiten übernommen;
$d'_2.w := d.w_{stand}$, $d'_2.T := (d'_1.T + d'_1.t)$, $d'_2.t := (d_2.T - d'_2.T)$,
der zweite Datenwert wird der Standardwert, sein Startzeitpunkt liegt zum Ende des ersten Datenwerts (kontinuierliche Folge), die Dauer ist die Differenz zum nächsten Startzeitpunkt.

Allgemein gilt für $1 \le i \le n$; $1 \le j \le 2n - 1 = m$:
falls j ungerade: $j = 2i-1$
$d'_j.w := d_i.w$, $d'_j.T := d_i.T$, $d'_j.t := d_i.t$,
die ungeraden Folgenglieder d'_j enthalten die Folgenglieder d_i,
falls j gerade: $i < n$, $j = 2i$
$d'_j.w := d.w_{stand}$, $d'_j.T := (d'_{j-1}.T + d'_{j-1}.t)$, $d'_j.t := (d_{i+1}.T - d'_j.T)$,
zwischen die Folgenglieder von F werden Standardwerte entsprechender Dauer eingeschoben.

Nach dem letzten Datenwert von F braucht kein Wert eingesetzt zu werden. Daher enthält F' nur 2n-1 Elemente. Bei einem anderen Ersetzungsverfahren ist lediglich für $d.w_{stand}$ jeweils der entsprechende Wert einzusetzen.■

Analog zu Satz 1 kann man eine Aussage für die Überschneidungen formulieren. Dazu muß jedoch noch einige Vorarbeit geleistet werden. Es werden nur sich überschneidende Daten betrachtet. Daher reicht es hier aus, endliche Teilfolgen zu untersuchen. Eine Überschneidung ist also stets als endliche Folge $F = <d_i>_n$ angebbar. Neben der einfachen Überschneidung existiert auch die mehrfache, oder genauer k–fache Überschneidung.

Def. k–fache Überschneidung

Sei $F = \langle d_i \rangle_n$ eine schwach chronologische Datenfolge, dann heißt F eine k–fache Überschneidung, wenn für alle $1 \le i \le n\text{-}k$, $1 \le j \le k$, d_i und d_{i+j} existieren, und es gilt: $d_i.T + d_i.t > d_{i+j}.T$.

Zunächst sollen jedoch nur die möglichen Überschneidungen zweier direkt aufeinanderfolgender Daten d_i und d_{i+1} betrachtet werden; Abbildung 19 stellt die möglichen Überschneidungen dar. Es sind drei Zeiträume α, β, und γ zu unterscheiden. Im Zeitraum α ist nur d_i gültig. In dem Zeitraum β, über den beide Werte gleichzeitig gültig sind, ist ein Mischwert aus den Werten $d_i.w$ und $d_{i+1}.w$ zu betrachten. Die Bildung dieses Mischwertes wird durch die Funktion $mix(d_i.w, d_{i+1}.w)$ beschrieben. Im Zeitraum γ ist nur noch der Datenwert gültig, dessen Gültigkeitsdauer später endet.

Den drei Zeiträumen entsprechend werden drei Daten $d°_i$, $d°_{i+1}$ und d_{n+1} definiert, die jeweils einen eindeutigen Wert haben und die Daten d_i und d_{i+1} ersetzen, so daß bezüglich der Präsentation die Teilfolgen $\langle d_i, d_{i+1} \rangle$ und $\langle d°_i, d°_{i+1}, d_{n+1} \rangle$ gleich sind. Dazu wird folgende Konstruktion gegeben:

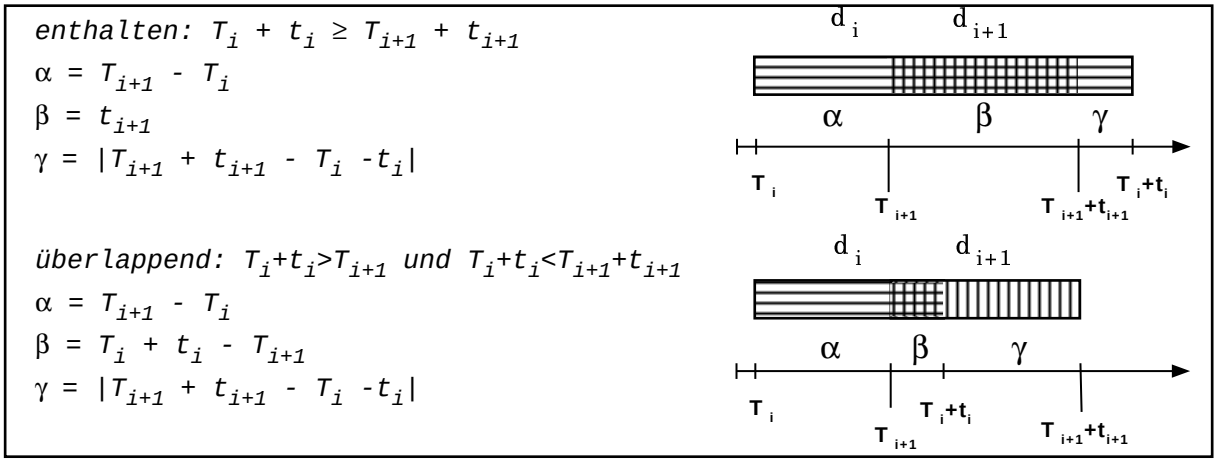


Abb. 23 mögliche Überschneidungen zweier aufeinanderfolgender Werte

Konstruktion zur Elimination von Überschneidungen zweier Daten:

Gegeben d_i und d_{i+1} einer schwach chronologischen Folge, mit:

- *$T_i \le T_{i+1}$, da schwach chronologisch*
- *$T_i + t_i > T_{i+1}$, da Überschneidung*

dann kann d_i und d_{i+1} durch $d°_i$, $d°_{i+1}$ und d_{n+1} so ersetzt werden, daß die neue Folge chronologisch und überschneidungsfrei ist.

Es sei: $\alpha := T_{i+1} - T_i$

$\gamma' := T_{i+1} + t_{i+1} - T_i - t_i$; $\gamma := |T_{i+1} + t_{i+1} - T_i - t_i|$

1. Abtrennen von d_{n+1}; aus $<d_i, d_{i+1}>$ wird $<d'_i, d'_{i+1}, d_{n+1}>$ mit:

d_{n+1}: $T_{n+1} := max\{ (d_k.T+d_k.t) \mid k \in \{i, i+1\} \} - \gamma$

$t_{n+1} := |\gamma'| = \gamma$

Falls $\gamma' > 0$: $w_{n+1} := w_{i+1}$,

Falls $\gamma' < 0$: $w_{n+1} := w_i$,

dann ist $\beta := T_{n+1} - T_{i+1}$

und es bleiben die beiden Daten mit gleichzeitigem Ende:

d'_i : $t'_i := \alpha+\beta$, $T'_i := T_i$, $w'_i := w_i$

d'_{i+1} : $t'_{i+1} := \beta$, $T'_{i+1} := T_{i+1}$, $w'_{i+1} := w_{i+1}$

2. Abtrennen von $d°_i$; aus $<d'_i, d'_{i+1}, d_{n+1}>$ wird $<d°_i, d''_i, d'_{i+1}, d_{n+1}>$ mit:

$d°_i$: $t°_i := \alpha$, $T°_i := T_i$, $w°_i := w_i$

und es bleiben die Daten mit gleichen Zeitparametern:

d''_i: $t''_i := \beta$, $T''_i := T_{i+1}$, $w''_i := w_i$

d'_{i+1}: wie unter 1. bestimmt.

3. Bestimmen von $d°_{i+1}$; aus $<d°_i, d''_i, d'_{i+1}, d_{n+1}>$ wird $<d°_i, d°_{i+1}, d_{n+1}>$:

$d°_{i+1}$: $t°_{i+1} := \beta$, $T°_{i+1} := T_{i+1}$, $w°_{i+1} := mix(d_i.w, d_{i+1}.w)$

Nach Konstruktion kann eine schwach chronologische Überschneidung $<d_i, d_{i+1}>$ durch eine chronologische, kontinuierliche Folge $<d°_i, d°_{i+1}, d_{n+1}>$ ersetzt werden, die die gleichen Werte präsentiert und gleiche Dauer hat. Abhängig von der Überschneidung treten folgende Varianten auf:

- *falls* $\alpha = 0, \gamma = 0$ *wird aus* $<d_i, d_{i+1}>$: $<d°_{i+1}>$
- *falls* $\alpha > 0, \gamma = 0$ *wird aus* $<d_i, d_{i+1}>$: $<d°_i, d°_{i+1}>$
- *falls* $\alpha = 0, \gamma > 0$ *wird aus* $<d_i, d_{i+1}>$: $<d°_{i+1}, d_{n+1}>$
- *falls* $\alpha > 0, \gamma > 0$ *wird aus* $<d_i, d_{i+1}>$: $<d°_i, d°_{i+1}, d_{n+1}>$

Unter Verwendung der vorangegangenen Konstruktion wird der folgende Satz bewiesen.

Satz 2

Eine Überschneidung F= $<d_i>_n$ kann durch eine äquivalente, kontinuierliche Folge F'= $<d'_l>_m$ dargestellt werden, wenn die Mischfunktion mix($d_i.w$, $d_{i+1}.w$) bekannt ist.

Beweis
durch schrittweise Konstruktion von F'= $<d'_l>_m$.

1. Beginnend bei Index 1 wird die erste Überschneidung zweier aufeinanderfolgender Daten eliminiert (s.o.). Es gelte folgende neue Indizierung:
   sei $<d_i, d_{i+1}>$ die erste Überschneidung in F dann ist F'= $<d'_l>_{n+1}$:

$l = 1, \dots, i-1$	: $d'_l = d_l$;	also $<d'_1, \dots, d'_{i-1}>$	$= <d_1, \dots, d_{i-1}>$
$l = i, i+1$	: $d'_l = d°_l$;	also $<d'_i, d'_{i+1}>$	$= <d°_i, d°_{i+1}>$
$l = i+2, \dots, n$	: $d'_l = d_i$;	also $<d'_{i+2}, \dots, d'_n>$	$= <d_{i+2}, \dots, d_n>$
$l = n+1$	: $d'_{n+1} = d_{n+1}$		

   oder:
   $F' = <d_1, \dots, d_{i-1}, d°_i, d°_{i+1}, d_{i+2}, \dots, d_n, d_{n+1}>$
2. Die Folge F' mit der kontinuierlichen Teilfolge $< d_1, \dots, d°_{i+1}>$ wird als Folge F in schwache chronologische Ordnung gebracht, da diese durch 1. nach der Stelle i+1 verloren sein kann.
3. Die neue Folge F aus Punkt 2. hat nach Konstruktion eine Überschneidung weniger als die Eingangsfolge. Es sind folglich die Punkte 1. und 2. zu wiederholen bis keine Überschneidung mehr vorliegt.

Das Verfahren endet, da eine endliche Datenmenge nur endlich viele Überschneidungen haben kann, und diese Stück für Stück verschwinden. ■

Nach den Konstruktionen aus den Sätzen 1 und 2 enthalten die kontinuierlichen, chronologischen Folgen bezüglich der Präsentation die selben Daten wie die diskreten, chronologischen bzw. die Überschneidungen. Zum Beweis, daß die Folgen tatsächlich äquivalent sind, fehlt noch der Nachweis, daß die Gesamtdauer nicht verändert wurde. Den Nachweis übernimmt das folgende Lemma 2.

Lemma 2

Bei der Konstruktion einer kontinuierlichen Folge F'= $<d'_j>_m$ aus einer diskreten Folge F= $<d_j>_m$ oder einer Überschneidung verändert sich die Gesamtdauer der Folge nicht.

Beweis:
Die Dauer einer diskreten Folge F ist: $T_n + t_n - T_1$
Die Dauer einer Überschneidung F ist: $\max\{ (T_k + t_k) \mid 1 \le k \le n\} - T_1$
Die Dauer der kontinuierlichen Folge F' ist: $T'_m + t'_m - T'_1$

Nach Konstruktion gilt für F':

- bei diskreter Folge F:
  $T'_1 := T_1$, $T'_m = T'_{2n-1} := T_n$, $t'_m = t'_{2n-1} := t_n$,
  damit ist $F' = T'_m + t'_m - T'_1 = T_n + t_n - T_1$
- bei Überschneidungen in F:
  $T'_1 := T_1$,
  $T'_m := \max\{ (T_k + t_k) \mid 1 \le k \le n\} - \gamma$ durch Abtrennen von d_{j+1} in der Konstruktion
  $t'_m := \gamma$
  damit ist
  $F' = T'_m + t'_m - T'_1 = \max\{ (T_k + t_k) \mid 1 \le k \le n\} - \gamma + \gamma - T_1 = \max\{ (T_i + t_i) \mid 1 \le i \le n\} - T_1$. ■

Betrachtet man beliebige, chronologische Folgen, so können sich Überschneidungen und Lücken in bunter Reihenfolge abwechseln. Nach Satz 2 lassen sich die Teilfolgen, die Überschneidungen bilden, durch kontinuierliche Folgen ersetzen, die nach Lemma 2 die gleiche Dauer haben. Als Ergebnis erhält man eine chronologische Folge mit Lücken, die nach Satz 1 entfernt werden können. Somit kann man jede beliebige chronologische Folge in eine äquivalente, kontinuierliche Folge transformieren. Daraus leitet sich die zweite Normalform ab.

Def. 2. Normalform

Eine Datenfolge eines kontinuierlichen Medii ist genau dann in zweiter Normalform (2NF), wenn sie in 1. Normalform und kontinuierlich ist.

Es gilt: $\forall\, d_i, d_{i+1} \in \wp : d_i.T + d_i.t = d_{i+1}.T$

Die zweite oder auch starke zweite Normalform erlaubt eine Senkung der Anforderungen an die Präsentationsfunktion, da die Mischfunktion bereits berechnet ist und somit zu einem Zeitpunkt nur genau ein Datum zu präsentieren ist. Es muß nun nur noch der Anfang und die Dauer eingehalten werden. Typischerweise wird dabei eine 1-Vorausschau auf die Daten vorgenommen. In vielen Fällen reicht es aus, wenn die Datenfolge chronologisch und überschneidungsfrei ist, aber Lücken enthält. Dazu wird die schwache 2. Normalform definiert.

Def. Die schwache 2. Normalform

Eine Datenfolge eines kontinuierlichen Medii ist genau dann in schwacher zweiter Normalform (w2NF), wenn sie in 1. Normalform und diskret ist.

Die schwache zweite Normalform reduziert die Anforderungen an die Präsentation gegenüber der ersten Normalform deutlich, da nur noch ein Wert oder eine Lücke betrachtet werden muß. Die (starke) zweite Normalform bietet sogar die Möglichkeit der direkten Datenauswertung, ohne Lücken berücksichtigen zu müssen. Wie bereits erwähnt wird dieser Vorteil durch ein erhöhtes Datenvolumen erkauft.

In der Verarbeitung der Daten bringt die zweite Normalform eine Einschränkung mit sich, da eine einmal berechnete und eingesetzte Mischung im allgemeinen Fall nicht wieder rückgängig gemacht, oder nur durch zusätzliches Wissen wiedergewonnen werden kann. Dagegen läßt sich eine Überschneidung einer Folge in 1NF durch Ändern der Zeitparameter der beteiligten Daten stets beliebig vergrößern oder verkleinern oder auch beheben. Also lassen sich Folgen in 1NF bezüglich der Präsentation leichter und vielseitiger ändern als Folgen in 2NF oder w2NF, wenn die Normalform erhalten werden soll. In einem System, das die Einhaltung der 2NF erzwingt, dürfen die Zeitpunkte T_i der Daten nicht direkt vom Benutzer verändert werden. Da solch ein System die Zeitpunkte aus $T_i+t_i=T_{i+1}$ berechnen kann und selbst berechnen muß, brauchen diese Zeitpunkte auch nicht gespeichert werden.

Durch die unterschiedlichen Präsentationsdauern der Einzelwerte einer Datenfolge in 2NF werden immer noch hohe Ansprüche an die Präsentationsfunktion gestellt. Die meisten verbreiteten Präsentationsfunktionen arbeiten daher nach einem anderen Prinzip. Statt einzelne Dauern zu überprüfen, nehmen sie stets die gleiche Dauer für die Daten an und unterstellen äquidistante Zeitpunkte, also einen Datenstrom, der in zweiter Normalform ist. Dies ist die dritte Normalform der Daten.

Def. 3. Normalform, kontinuierlicher Datenstrom

Eine Datenfolge eines kontinuierlichen Medii ist genau dann in dritter Normalform (3NF), wenn sie in 2. Normalform und ein Datenstrom ist. Man spricht auch von einem kontinuierlichen Datenstrom.

Ein kontinuierlicher Datenstrom wird nach [Herrtw90] auch als periodischer, kontinuierlicher Datenstrom bezeichnet. Die einfache Präsentation eines kontinuierlichen Datenstroms ist durch den festen, stets gleichen Ablauf, bei dem gleichmäßig zu festgesetzten Zeitpunkten das nächste Datum verarbeitet wird, ein wichtiger Vorteil der dritten Normalform. Ein anderer wichtiger Vorteil ist die mögliche Datenreduktion. Da für jedes Datum seine Dauer bekannt und sein Zeitpunkt direkt berechenbar ist, können diese Zeitparameter eingespart werden. Die Zeitparameter kontinuierlicher Datenströme werden daher für den gesamten Strom einmal durch Raten beschrieben.

Man beschreibt eine Datenfolge $F= <d_j>_n$ in 3NF durch folgende Parameter:

- die Dauer eines Einzelwertes $\Delta t = d_1.t$, oder seine Rate $1/\Delta t$,
- den Anfangszeitpunkt $T^0 = d_1.T$, Startzeitpunkt des ersten Datenwerts,
- die Anzahl n der Datenwerte, die er enthält.

Somit ist in 3NF $F= <d_j>_n = S(\Delta t, T^0, n, <w_i>_n)$. Die Dauer t von S berechnet sich nach $t= n \cdot d_1.t$. Die Elemente w_i von S sind nur noch die Werte $d_i.w$ von F.

In einer verkürzten Schreibweise kann der Datenstrom auch durch $S(\Delta t, T^0, <w_i>)$ dargestellt werden. In diesem Fall bleibt die Anzahl der Datenwerte unberücksichtigt, die Datenfolge ist potentiell unendlich.

Die einfache Präsentation und vor allem die Reduktion der Daten durch die Speicherung eines kontinuierlichen Datenstroms als $S(\Delta t,T^0,n,<w_i>_n)$ wird mit einschneidenden Einschränkungen bei der Verarbeitung der Daten erkauft. Alle auf dem Datenstrom ausgeführten Operationen müssen normalformerhaltend sein. Ein Verlängern oder Verkürzen der Dauer eines Datums kann daher nur durch Einfügen beziehungsweise Entfernen von Daten er-

reicht werden. Neben dem Nachteil, daß dazu ganze Teilfolgen verschoben werden müssen, ergibt sich auch, daß die Zeitdifferenzen, um die verlängert oder verkürzt werden kann, nur Vielfache von Δt sein können. Ein weiterer Nachteil bei der Verwendung der Darstellung als $S(\Delta t, T^0, n, <w_i>_n)$ liegt darin, daß die einzufügenden Werte wesentlich größere Datenmengen enthalten können, als die Zeitparameter darstellen. Dies ist zum Beispiel für Videobilder der Fall.

Durch Bearbeitung bzw. Veränderung der Datenströme in der Form $F= <d_j>_n$, insbesondere ihrer Zeitparameter, können stets wieder beliebige Folgen entstehen. Aufgrund der Aussagen in diesem Kapitel kann man jedoch immer wieder kontinuierliche Folgen erstellen. Nach den Sätzen 1 und 2 kann jede Datenmenge als kontinuierliche Datenfolge dargestellt werden. Damit eine beliebige, kontinuierliche Folge in einen kontinuierlichen Datenstrom gewandelt werden kann, ist zu zeigen, wann die Entstehungszeitpunkte äquidistant sind oder wie sie so gemacht werden können, und wie man gleiche Gültigkeitsdauern erreicht. Dazu müssen die Eigenschaften aller Datenparameter berücksichtigt werden. Grundlage zu diesen Umformungen ist die Umrechnung von Sampling-Raten in der PCM–Technik; weiterführende Überlegungen und Berechnungen wurden im Rahmen der Vorarbeiten zur Erstellung eines Audioeditors in [Bast93] durchgeführt. Eine genaue Beschreibung der Zusammenhänge führt über den Rahmen dieser Arbeit hinaus und bringt keine neuen Erkenntnisse.

4.1.3 Eigenschaften der Datenparameter in digitalen Rechnersystemen

In der Umwelt liegen alle Datenparameter, der Wert, der Startzeitpunkt und die Dauer in beliebig feiner Auflösung vor, das heißt die Werte der Parameter sind kontinuierlich. Beim Übergang zu digitalen Rechnersystemen müssen bei der Perzeption diese Werte diskretisiert werden. Es gibt damit für die Werte Schritte, die einem Wertebereich in der Umwelt zugeordnet werden, und zwischen diesen Schritten gibt es keine Werte. Dieser Übergang muß für alle drei Parameter vorgenommen werden. Dabei stellt sich für jeden Parameter die Frage, was geht bei der Diskretisierung durch die eingeführte Ungenauigkeit verloren, und was kann man durch sie gewinnen?

Als Beispiel kann die Puls–Code–Modulation (PCM) dienen, bei der genau diese Schritte vorgenommen werden und gut untersucht sind. In der PCM–Technik wird zunächst die Zeit diskretisiert, indem die Signalwerte nicht mehr kontinuierlich sondern zu einzelnen Abtastzeiten bestimmt werden. Die Grundlage hierfür bildet das Abtasttheorem, das das Verhältnis zwischen dem Ursprungssignal und der Abtastfrequenz herstellt [Skritek88]. Es geht davon aus, daß sich das Ursprungssignal aus Schwingungen unterschiedlicher Frequenzen und wechselnder Amplitude zusammensetzt. Jetzt kann man die Festlegung der Abtastfrequenz aus zwei Richtungen betrachten:

1. Man wählt die Abtastfrequenz fest zu einem Wert f_{ab}, dann gehen vom Ursprungssignal alle Frequenzen, die höher als $½f_{ab}$ sind, verloren.
2. Sollen alle Frequenzen des Ursprungssignals bis zu einer Maximalfrequenz f_{max} berücksichtigt werden, so muß die Abtastfrequenz mindestens doppelt so hoch also $2 \cdot f_{max}$ sein.

Damit ist zunächst ein Teil der Frage nach den Verlusten beantwortet. Die diskretisierte Zeit ist nun als Folge ganzer Zahlen von Abtastzeitpunkten beschreibbar. Die Abtastzeitpunkte entsprechen dem Parameter Zeitpunkt der zeitabhängigen Daten, ihre Abstände geben die Dauer der Werte an. Im allgemeinen werden die einmal festgelegten Abtastfrequenzen nicht mehr verändert und die erhaltenen Wertemengen sind damit in dritter Normalform.

Das bisher beschriebene Verfahren stellt nur die Puls–Amplituden–Modulation dar. Es fehlt noch die Codierung. Diese ordnet die bisher noch kontinuierlich gemessenen Abtastwerte festen Intervallen zu. Hierbei entsteht wieder ein Datenverlust, der abhängig von der Anzahl der verwendeten Intervalle ist. Diese Verfahren nennt man Quantisierung, den entstehenden Fehler Quantisierungsfehler. Da bei gleichgroßen Quantisierungsintervallen der Quantisierungsfehler betragsmäßig kleiner Werte relativ groß ist, wählt man häufig die Quantisierungsintervalle verschieden groß, so daß Intervalle, die betragsmäßig kleine Werte repräsentieren, auch klein und Intervalle, die betragsmäßig große Werte repräsentieren, auch groß sind. Einem Datum ordnet man anstatt seines Wertes die Nummer des Quantisierungsintervalles zu, also auch ganze Zahlen.

Die PCM–Technik wird für verschiedene Zwecke eingesetzt und ist vor allem in der Audio–Technik verbreitet. Sie dient zum Beispiel der digitalen Sprachübertragung oder der Erstellung von Audio–Compact–Discs (CD's). Vergleichbar zum Abtastvorgang im Audiobereich wird im Bereich Video mit einer Aufnahmerate von Einzelbildern gearbeitet. Geht man dabei im Audiobereich von Schwingungen aus, so sind es im Videobereich Bewegungen. Die Genauigkeit, mit der eine Bewegung wiedergegeben werden soll, bestimmt die Frequenz für die Einzelbilder. Für Video oder Film sind 25 Bilder pro Sekunde beim Fernsehbild, 24 Bilder pro Sekunde für Film und wahlweise 24 oder 18 Bilder pro Sekunde beim Schmalfilm ausreichend. Die Berechnung der Datenwerte geschieht analog zur Quantisierung von PCM, ist allerdings ungleich komplizierter. Sie wird bei einigen Geräten, wie zum Beispiel Photo-CD, schon in Hardware angeboten. Das Ergebnis ist auch nicht ein einzelner Wert sondern eine Matrix aus Bildpunkten. Zu jedem Bildpunkt werden entweder nur Helligkeitswerte, für Schwarz-Weiß–Bilder, oder zusätzliche Farbwerte für bunte Bilder angegeben. Auch hier bestimmt die Feinheit der Abstufungen die Qualität; 256 verschiedene Farben sind für viele Anwendungen ausreichend, aber die Auflösungsmöglichkeiten des menschlichen Auges liegen weit höher bei über 16 Millionen Farben.

Ein wichtiger Unterschied zwischen den vorgestellten Verfahren für Audio und für Video liegt darin, daß Audiowerte (Samples) für sich allein keinen erkennbaren Laut oder Ton darstellen, Videowerte dagegen schon Einzelbilder, also erkennbare Einheiten. Ein neuerer Ansatz geht davon aus, für Audio nicht mehr einzelne Samples zu betrachten, sondern entweder wie bei MIDI eine Nummer, der eine Folge von nacheinander zu spielenden Samples zugeordnet wird, oder ein Spektrum von Frequenzen und deren Amplituden, die zu einem Zeitpunkt gelten [MPEG93]. In beiden Fällen entspricht im Audiobereich der Wert nicht mehr einfach einer ganzen Zahl, sondern eine Gruppe von Zahlen.

Faßt man die Aussagen zur Digitalisierung der kontinuierlichen Medien zusammen, so zeigt sich, daß für die Wertebereiche der drei Parameter Wert, Zeitpunkt und Dauer jeweils eine kleinste Differenz der Werte ε_W bzw. Zeit ε_Z zwischen zwei Werten festgelegt werden kann, unterhalb der die Werte der Parameter nicht mehr unterschieden werden. Ausgehend von einem festzulegenden, absolut kleinsten Wert lassen sich dann alle Parameter durch ganzzahlige Vielfache von ε_W bzw. ε_Z darstellen. Die Größe der ε's stellt den Bezug zur

Realität her und muß für jede Datenfolge ersichtlich sein. Auf diese Weise erreicht man zuerst eine Digitalisierung der Zeit und dann der Werte. Die Zeit ist dabei stets linear, die Werte können Einzelwerte, Felder oder Matrizen sein. Die bei der Digitalisierung auftretenden Verluste sind durch die Festlegung der ε's unerheblich. Der Gewinn liegt in den beliebig oft herstellbaren unverfälschten Kopien der Daten, also der gleichbleibenden Qualität der Kopien.

Bei dieser Digitalisierung während der Perzeption kontinuierlicher Medien entstehen zunächst kontinuierliche Datenströme mit $S(\varepsilon_Z, T^0, n, <w_i>_n)$. Eine Variante ist, jeweils nach einer Zeit von ε_Z einen Test auf eine Veränderung durchzuführen und nur bei einer Veränderung einen neuen Wert zu erstellen. Dann erhält man eine Datenfolge in 2NF. Diese Methode hat allerdings den Nachteil, daß die Zeitparameter erst ex post bestimmt werden und so ihre Werte erst bekommen, wenn die Gültigkeit vorbei ist. Daher ist dieses Verfahren bei der Live–Präsentation nur bedingt einsetzbar.

Die Parameter der Daten können aber auch bereits im Rechner durch Simulation oder Animation berechnet werden. Dann ist diese Berechnung Grundlage für die Genauigkeit der Darstellung und die Erzeugung einer Normalform.

Bei den Werten der Daten gibt es insbesondere für Videodaten verschiedene Kompressionsverfahren. Neben der internen Kompression, die nur versucht Einzelbinder mit weniger Daten zu speichern, gibt es Algorithmen, die die Veränderung von einem Bild zum nächsten berechnen und nur diese Änderungen speichern (vgl. Kap. 2). Dafür müssen die Datenfolgen mindestens in w2NF vorliegen. Im Gegensatz zu den Zeitparametern lassen sich die Wertparameter nicht generell beschreiben, da für die Repräsentation von Tönen und Bildern oder anderen Werten kontinuierlicher Medien und deren Interpretation bei der Präsentation schon verschiedene Datenformate existieren. Somit ergeben sich aus den unterschiedlichen Anforderungen an die Daten beziehungsweise an die kontinuierlichen Medien verschiedene Datentypen für die Datenfolgen. Im weiteren Verlauf der Arbeit muß dies immer wieder berücksichtigt werden, aber es ist trotzdem eine weitgehend gemeinsame Behandlung möglich.

4.1.4 Grundlagen des Zeitbezugssystems

In diesem Abschnitt sollen die Repräsentation der Zeit und die Interpretation der Zeitparameter kontinuierlicher Medien behandelt werden. Beide werden durch das Zeitbezugssystem bestimmt. Die Zeitparameter sollen einfach aufgebaut und leicht interpretierbar sein, gleichzeitig aber auch eine breite Palette von Bearbeitungsfunktionen ermöglichen. Zunächst wird die beobachtete Parametergewinnung der Daten kontinuierlicher Medien, insbesondere die der Dauer, als Grundlage verwendet. Das Zeitbezugssystem wird zu einem Teil in den Perzeptions– und Präsentations–Umgebungen und zum anderen Teil in den dort enthaltenen Perzeptions– und Präsentations–Funktionen realisiert.

Wie im vorigen Abschnitt beschrieben, lassen sich die Zeitparameter als ganze Zahlen darstellen. Dabei wird als Grundraster jeweils ein Schritt in der Größe ε_Z festgelegt. Da ε_Z die feinste zeitliche Unterscheidung zwischen den Daten ist, kann ein Datum minimal einen Schritt lang sein, aber auch mehrere Schritte dauern. Die Dauern der Daten sind somit in ganzzahligen Schritten von ε_Z angegeben. Ein Datum d_i einer Folge *F* beginnt zu einem be-

stimmten Schritt T_i, relativ zum Anfang der Folge beziehungsweise deren erstem Element d_1, und dauert eine gewisse Anzahl von Schritten t_i. Damit sind die Domänen der T_i und der t_i auf die natürlichen Zahlen inklusive der Null festgelegt.

Da verschiedene Schrittlängen, ε_Z's, deren Größe von der gewünschten Genauigkeit abhängt, benutzt werden, es ist nötig, die Schrittlänge messen und angeben zu können. Im digitalen Rechner wird dazu wieder ein allgemeines Grundraster verwendet, dessen Auflösung deutlich feiner als das feinste zu erwartende Schrittraster sein muß. Die Elemente dieses feinen Rasters bilden dann eine Uhr, die die Schrittlängen bestimmt. Übereinstimmend mit dem Uhrenbegriff und den kleinsten diskreten Schritten einer Uhr wird das feinste Element ein ***Tick*** genannt.

Das entsprechende Zeitbezugssystem hat mindestens zwei Zeitachsen, eine, die in Ticks und mindestens eine weitere, die in Schritten gemessen wird. Die Ticks erzeugt ein Ticker und die Schritte ein Schrittgeber. Der Ticker ist systembezogen und an eine Umgebung gebunden; der Schrittgeber ist dagegen mit einer Perzeptionsfunktion oder einer Präsentationsfunktion verbunden. Somit können in einer Umgebung mehrere Schrittgeber oder Schrittachsen existieren, die alle auf ein und denselben gemeinsamen Ticker bezogen sind. Jeder Schritt dauert eine zunächst feste Anzahl von Ticks [Unge92]. Das Zeitbezugssystem wird in der Abbildung 24 veranschaulicht.

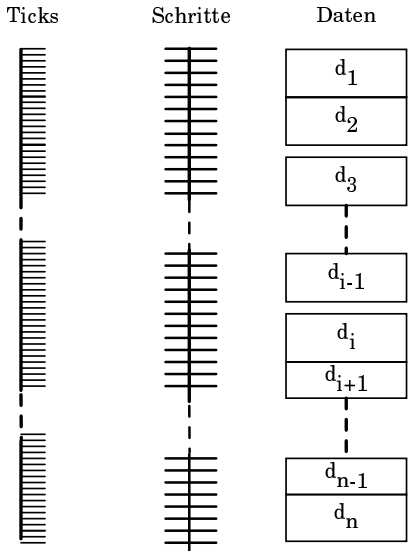


Abb. 24 Das Zeitbezugssystem

Durch diese Beziehungen sind sowohl die Dauern der Daten als auch der Schritte bestimmt, das Problem wurde allerdings auf die Dauer eines Ticks verlagert. Die Festlegung der Dauer eines Ticks vereinfacht sich aber, da er im Gegensatz zu den Schritten (ε_Z's) nicht mehr mit einer bestimmten Codierung der Daten verbunden ist. Es bietet sich daher an, die Tickdauer zu standardisieren. Da aber kein Standard existiert, müssen für die kontinuierlichen Daten die Tickdauern angegeben werden, und es ist zu untersuchen, welche Tickdauern die meisten Schrittdauern unterstützen.

Da Ticks und Schritte ein Raster liefern, ist es möglich, statt der Dauern auch die Rate anzugeben, mit der sie auftreten. Raten haben die Vorteile, daß sie bereits eine breite Verwendung haben, und daß sie im allgemeinen in ganzen Zahlen angegeben werden. Der Vorteil dieses Systems ist die Ausnutzung der diskretisierten Zeit in der ganzzahligen Darstellung, so wird der Umgang mit der Zeit einfacher. Die Tickrate berechnet sich dann aus den auftretenden Schrittraten als das kleinste gemeinsame Vielfache. In den Perzeptions– und Präsentations–Umgebungen muß die Tickrate und für die Perzeptions– und Präsentations–Funktionen die Schrittrate entsprechend eingestellt werden.

In [Hesme93] werden gebräuchliche Schrittraten und geeignete Tickraten untersucht. Dabei ergibt sich, daß eine Tickrate von 600 Hz sowohl eine breite Palette an Schrittraten abdeckt, als auch in einem Rechner (auch in Software) einfach zu realisieren ist. Weiterhin wird gezeigt, daß es nicht für alle Schrittraten zweckmäßig ist, einen gemeinsamen Ticker in ganzzahliger Teilung zu verwenden. Die Tickraten würden dann so groß, daß sie die Verarbeitung der kontinuierlichen Daten stören oder sogar verhindern würden. In [Hesme93] werden verschiedene Verfahren aufgezeigt, wie auf der Basis von ganzzahligen Schrittraten und geeigneten Korrekturmaßnahmen dennoch ein gemeinsamer Ticker verwendet werden kann. In der Tabelle 3 werden für verschiedene gebräuchliche Codierungen ihre Schrittraten und die Teiler zur Tickrate von 600 Hz angegeben.

Codierung		Schrittrate		Teiler
Audio	CD–DA	75	Sectors/s	8
	MPEG I Layer I	125	Frames/s	24/5
	MIDI Clock 60 bpm	24	Clocks/s	25
	MIDI Clock 100 bpm	40	Clocks/s	15
Video	Film	24	Bilder/s	25
	PAL	25	Frames/s	24
	NTSC	30	Frames/s	20

Tab. 3 *Verhältnis der Schrittraten zum 600 Hz Ticker*[1]

Da abhängig von der Implementierung der Ticker und der gewünschten Breite ihrer Anwendbarkeit und Genauigkeit auch andere Tickraten auftreten können, müssen sowohl die Tickrate als auch die Ticks pro Schritt zu einer Datenfolge angegeben werden oder es werden die Schrittrate und die verwendete Tickrate angegeben. Im folgenden wird davon ausgegangen, daß jeweils die Raten für den Ticker und den Schrittgeber angegeben sind. Für ein geeignetes Verhältnis der Raten zueinander ist der Benutzer verantwortlich.

4.1.5 Ticker und Schrittgeber

So, wie das Zeitbezugssystem bisher vorgestellt wurde, unterstützt es die Perzeption und die Präsentation zu den vorgegebenen Zeitbedingungen. Dazu werden entsprechende Ticker und Schrittgeber gestartet, die die Zeitparameter bestimmen beziehungsweise interpretieren.

[1] MIDI Clock wird in battements per minute (bpm) gemessen

Somit läßt sich das Zeitbezugssystem abstrakt durch die Raten von Ticker und Schrittgeber sowie Start–Stop–Operationen auf diesen beschreiben. Ähnliche Ansätze sind die Uhren der Zeitkapseln in [Herrtw90] und die Uhren in Tactus [Dann92]. Ein solches System realisiert den statischen Fall, in dem zum Beispiel keine Geschwindigkeitsänderung im Zeitbezugssystem durchgeführt werden kann, sondern nur direkt über die Zeitparameter der Daten möglich ist, also durch Datenänderung. Erweitert man die Menge der Operationen auf dem Zeitbezugssystem um Änderungen der Tickrate und der Schrittrate, und deren Domänen auf die positiven und negativen ganzen Zahlen, so erhält man eine flexible Steuerung der Zeitbedingungen.

Im folgenden werden Operationen auf dem Zeitbezugssystem vorgestellt, die nicht nur statisch die vorgegebene Interpretation der Zeitparameter der Daten erlauben, sondern dynamisch änderbar auch eine erweiterte Interpretation ermöglichen. Grundlage hierfür ist die Unabhängigkeit der Änderungen der Tick– und Schrittraten im Zeitbezugssystem von den Zeitparametern der Daten.

In einem Zeitbezugssystem gibt es einen Ticker und einen oder mehrere Schrittgeber. Jedes dieser Elemente hat eine änderbare Rate. Die Operationen auf dem Ticker sind unabhängig von den Operationen auf den Schrittgebern. Die Schrittgeber sind vom Ticker abhängig, haben aber eigene Operationen. Da sowohl Ticker als auch Schrittgeber Uhren darstellen, entsprechen sich ihre Operationen weitgehend.

Der Ticker

Der Ticker ist ein in sich abgeschlossener Teilprozeß in einer Perzeptions– oder Präsentationsumgebung. Seine Ausgaben sind Ticks, die den Schrittgebern der in der Umgebung eingebetteten Grundfunktionen zugänglich gemacht werden. Die Ticks sind ganze Zahlen, die fortlaufend und im allgemeinen aufsteigend erzeugt werden. Ein Parameter des Tickers ist seine Tickrate, die in ganzen Zahlen pro Sekunde angegeben wird. Damit ist die kleinste mögliche Rate ein Tick pro Sekunde, alle anderen Raten liefern „schnellere" Ticks; eine Rate von Null liefert keine Ticks. Da der Ticker deutlich schneller als alle Schrittgeber arbeiten soll, ist eine Rate von einem Tick pro Sekunde extrem langsam.

Der Ticker fängt sofort bei der Initialisierung an zu ticken. Dazu müssen bei der Initialisierung die Parameter für die Tickrate und den ersten Tickwert, der von Null verschieden sein kann, angegeben werden. Ein Ruhen des Tickers kann durch eine Rate von Null erzeugt werden. Die Zählrichtung kann aufwärts oder abwärts sein, abhängig davon, ob eine positive oder negative Rate angegeben wurde. Die Zählgeschwindigkeit wird durch den Betrag der Rate eingestellt und kann auch abgefragt werden. Der aktuelle Tickwert kann durch gezieltes Setzen verändert werden. Die Veränderung erzeugt Diskontinuitäten der Daten und sollte nur zu Ruhezeiten des Tickers zu verwenden. Diese Funktion ermöglicht es, die kontinuierlichen Medien in diskreten Abschnitten zu untersuchen, wie zum Beispiel eine Einzelbildfortschaltung beim Film. Ein Stoppen des Tickers beendet ihn so, daß er nur durch eine neue Initialisierung wieder gestartet werden kann.

Die Operationen auf dem Ticker mit den Parametern ***rate*** und ***tick*** sind:

- initialisieren des Tickers:
  init (rate, tick): ***INTEGER × INTEGER*** → ***Ø;***
- setzen der Tickrate:
  set_rate (rate): ***INTEGER*** → ***Ø;***

- lesen der Tickrate
  get_rate (): ***Ø*** → ***INTEGER;***
- setzen des Tickwerts:
  set_tick (tick): ***INTEGER*** → ***Ø;***
- lesen des Tickwerts:
  get_tick (tick): ***Ø*** → ***INTEGER;***
- beenden:
  stop (): ***Ø*** → ***Ø;***

Der Schrittgeber

Ein Schrittgeber ist ein Teilprozeß in einer Perzeptions– oder Präsentationsfunktion. Er zählt Schritte, die er in einer Perzeptionsfunktion dem Datenwert zuweist, in einer Präsentationsfunktion bestimmt er aus ihnen das zu präsentierende Datum. Für jeden Zählschritt wartet er die Anzahl von Ticks, die sich aus dem Verhältnis der Schritt– zur Tickrate ergibt. Diese Ratendefinition bietet eine gute Grenze zwischen kontinuierlichen, zeitabhängigen und diskreten, zeitunabhängigen Medien, die in größeren Abständen Veränderungen haben. Diskrete Medien lassen sich durch eine Rate von Null und einzelne Schrittweiterschaltungen durch explizites setzen der Schritte beschreiben. Das Setzen der Schritte entspricht dann dem Eintreffen eines Ereignisses.

Der Schrittgeber hat zunächst dem Ticker analog die Parameter Schritt und Schrittrate. Der Tick bestimmt den Schritt nach folgender Formel, wobei eine Tickrate von 0 nicht berücksichtigt wird, es wird jeweils das letzte gültige Verhältnis (Schrittrate / Tickrate) verwendet:

$$\text{Schritt} = \lfloor \text{Tick} \cdot (\text{Schrittrate} / \text{Tickrate}) \rfloor$$

Nach dieser Formel müssen alle Schrittgeber gemeinsam mit dem Ticker starten. Es soll aber ermöglicht werden, den Schrittgeber erst zu einem bestimmten Tick zu starten. Daher wird ein weiterer Parameter benötigt, der angibt, wann der Schrittzähler anfangen soll zu zählen. Dieser neuer Parameter heißt Starttick. Da die Zeitparameter der Daten nur positive ganze Zahlen sind, genügt es, wenn ein Schritt entweder eine positive ganze Zahl oder Null ist. Somit bestimmen die Tick– und die Schrittrate und die Differenz (Tick - Starttick) den aktuellen Schrittwert [Hesme93]. Die Formel, nach der sich der Schritt berechnet, ist dann:

Falls (Tick - Starttick) < 0, dann ist Schritt = 0; sonst:

$$\text{Schritt} = \lfloor (\text{Tick} - \text{Starttick}) \cdot (\text{Schrittrate} / \text{Tickrate}) \rfloor$$

Durch den Starttick kann auch bei ruhendem Ticker das Datum verändert werden. Bei der Perzeption ändern sich dadurch die Zeitparameter, bei der Präsentation ist das entsprechende Datum darzustellen.

Die Operationen auf einem Schrittgeber mit den Parametern ***start_tick***, ***rate*** und ***step*** sind:

- initialisieren des Schrittgebers mit Starttick und Schrittrate:
  init (start_tick, rate): ***INTEGER × INTEGER*** → ***Ø;***
- setzen der Schrittrate:
  set_rate (rate): ***INTEGER*** → ***Ø;***
- lesen der Schrittrate
  get_rate (): ***Ø*** → ***INTEGER;***

- setzen des Startticks:
  set_start_tick (tick): ***INTEGER*** → ***Ø;***
- lesen des Startticks:
  get_start_tick (): ***Ø*** → ***INTEGER***;
- lesen des Schrittwerts
  get_step (): ***Ø*** → ***INTEGER;***
- beenden:
  stop (): Ø → ***Ø;***

Der Parameter ***step*** wird durch die obige Formel bestimmt und nicht auf andere Weise verändert. Die Verbindung zwischen Ticker und Schrittgeber ist die Tickverteilung. Sie hat dafür zu sorgen, daß alle Schrittgeber die benötigten Ticks regelmäßig und exakt erhalten.

Ergänzungen zum Ticker

In den vorgestellten Formeln zur Berechnung der Schritte aus den Ticks spielt das Verhältnis von Schritt– zur Tickrate eine wichtige Rolle. Eine Änderung der Schrittrate bewirkt *eine* Veränderung der Geschwindigkeit der Schritte, eine Änderung der Tickrate bewirkt *keine* Veränderung der Geschwindigkeit der Schritte, da die Ticks auch entsprechend öfter kommen. Nun soll es ermöglicht werden, durch eine Änderung der Tickrate alle mit dem Ticker verbundenen Schrittgeber im gleichen Verhältnis zu beschleunigen oder zu verlangsamen. Dazu wird der Parameter Tickrate des Ticker in einen internen und einen externen Parameter aufgeteilt und es werden zwei neue Operation eingeführt. Diese Erweiterung führt auch aus dem Dilemma, daß eine Tickrate von 0 zulässig ist, aber im Verhältnis (Schrittrate / Tickrate) nicht berücksichtigt werden kann. In diesem Verhältnis wird immer die externe Tickrate verwendet, die jetzt nicht mehr auf 0 gesetzt werden darf. Angehalten wird der Ticker über die interne Tickrate mit einem Wert von 0.

Die Operationen auf dem ergänzten Ticker mit den Parametern ***internal_rate***, ***external_rate*** und ***tick*** sind:

- initialisieren des Tickers:
  init (rate, tick): ***INTEGER × INTEGER*** → ***Ø;***
  es werden die interne und die externe Tickrate auf den Wert von ***rate*** gesetzt; der aktuelle Tickwert wird der von ***tick***.
- setzen Tickrate:
  set_rate (rate): ***INTEGER*** → ***Ø;***
  es werden sowohl die interne als auch die externe Rate verändert.
- lesen Tickrate
  get_rate (): ***Ø*** → ***INTEGER;***
  es wird der Wert der externen Rate zurückgegeben.
- setzen des Tickwerts:
  set_tick (tick): ***INTEGER*** → ***Ø;***
- lesen des Tickwerts:
  get_tick (tick): ***Ø*** → ***INTEGER;***
- beenden:
  stop (): ***Ø*** → ***Ø;***

Bis hierher sind das die selben Operationen, die nur leicht angepaßt wurden.

- setzen interne Tickrate:
  set_intern_rate (rate): ***INTEGER*** → ***Ø;***
  mit der Eigenschaft, daß die Tickrate geändert wird, ohne daß ***get_rate()*** diese Rate zurückgibt.
- lesen interne Tickrate:
  get_intern_rate (): ***Ø*** → ***INTEGER;***

4.1.6 Ein Datentyp für kontinuierliche Medien

Aus den bisher gesammelten Eigenschaften kontinuierlicher Medien läßt sich ein abstraktes Datenmodell herleiten, dem semantisch eine strenge Interpretation im Ticker–Schrittgeber–Modell zugeordnet ist. Der daraus resultierende Datentyp ***sequence*** wird im folgenden beschrieben.

Prinzipiell wird eine potentiell unendliche Folge von Tripeln (w, T, t) modelliert, deren Werte w Einzelwerte, Folgen oder Matrizen oder ähnliches sein können. Die Zeitparameter T und t werden in Schritten angegeben. Sei W der Wertetyp der Elemente, dann ist der Elementtyp für die Tripel E = (W × INTEGER × INTEGER) und für ein d ∈ E gibt es die Selektion · , wie am Anfang des Kapitels eingeführt. Zu diesem Elementtyp E wird ein ausgezeichnetes Element definiert, das sich im Wert von allen anderen Elementen unterscheidet und dessen Dauer Null ist. Dieses Element wird mit *EOS* für *end of sequence* bezeichnet. Es ist in jeder Sequenz enthalten und sein Startzeitpunkt wird auf die Dauer der Sequenz gesetzt.

Der Datentyp der potentiell unendlichen Datenfolge eines kontinuierlichen Medii wird als *sequence* mit den Parametern *S_rate*, für die Schrittrate, und *T_rate*, für die Tickrate, sowie dem Start–Schritt *Start_Step* und der Gesamtdauer in Schritten beschrieben.

$$\textbf{\textit{sequence}} = (S\_rate, T\_rate, Start\_Step, Dauer, \langle d_j \rangle_n)$$
$$\textbf{\textit{sequence}} \in (\text{INTEGER} \times \text{INTEGER} \times \text{INTEGER} \times \text{INTEGER} \times E^n)$$

Für eine Sequenz sein die folgenden Operationen definiert:

Auf Sequenzen:

- Erzeugen einer neuen Sequenz **a**:
  create (S_rate, T_rate): ***INTEGER × INTEGER*** → ***sequence;***
  beim Erzeugen einer Sequenz werden die Parameter für die Schrittrate und die Tickrate mit übergeben, der Start–Schritt und die Dauer werden auf den Wert Null gesetzt. Die Sequenz besteht nur aus dem Element EOS.
- Löschen einer Sequenz **a**:
  destroy (a): ***sequence*** → ***Ø***;
- Auslesen der Parameter:
  get_S_rate (a): ***sequence*** → ***INTEGER;***
  get_T_rate (a): ***sequence*** → ***INTEGER;***
  get_start_step (a): ***sequence*** → ***INTEGER;***
  get_duration (a): ***sequence*** → ***INTEGER;***
  diese Operationen geben die gesetzten Werte zurück. Bei der Operation ***get_duration(a)*** könnte die Dauer auch aus den Einzeldauern berechnet werden. Wegen der erwarteten großen Datenmengen ist dies aber nicht vorteilhaft. Dafür müssen alle Operationen, die eine Auswirkung auf die Dauer haben, auch den entsprechenden Sequenzparameter und das

Element EOS selbst verändern. Da die Startzeitpunkte der Sequenzelemente relativ zum Start der Sequenz angegeben werden, muß für eine Sequenz ***a*** immer gelten, daß ihre Dauer *a.t* so bestimmt werden kann:

$$a.t = max \{ (d.T + d.t) \mid d \in a\}$$

Entsprechende Überlegungen gelten auch bei den ***set–***Operationen.

- Setzen der Parameter:
  set_S_rate (a, S_rate): ***sequence × INTEGER*** → ***sequence;***
  set_T_rate (a, T_rate): ***sequence × INTEGER*** → ***sequence;***
  set_start_step (a, T): ***sequence × INTEGER*** → ***sequence;***
  set_duration (a, t, Flag): ***sequence × INTEGER × BOOLEAN*** → ***sequence;***

  Die Operation ***set_duration (a, t, Flag)*** ist abhängig von den Elementen, sie kann in zwei Richtungen verwendet werden. Die Dauer kann einerseits aus den Elementen hergeleitet werden, andererseits kann die Veränderung der Dauer Auswirkungen auf die Elemente haben. Insbesondere, falls abwechselnd auf Elementen und Sequenzen gearbeitet wird, kann der Fall auftreten, daß die Parameter nicht das beschreiben, was der Benutzer meint. Das heißt, daß bei jedem Wechsel der Ebenen diese Operation aufgerufen werden kann. Es ist möglich die Berechnungsrichtung durch den Parameter Flag zu kennzeichnen. Ist das Flag *false*, so sollen die Werte der Sequenzparameter gelten, und die Elemente müssen angepaßt werden, andernfalls ist das Flag *true* und die Parameter berechnen sich aus den Elementen.

- Verketten / Konkatenieren zweier Sequenzen **a** und **b** zu einer:
  concat (a, b): ***sequence × sequence*** → ***sequence;***

  Ein Verketten ist genau dann möglich, wenn die Listenelemente vom gleichen Typ und die Parameter *S_rate* und *T_rate* gleich sind.

- Extraktion eines Teils von Index **p1** bis Index **p2** einer Sequenz **a**:
  extract(a, p1, p2): ***sequence × INTEGER × INTEGER*** → ***sequence;***

  Die Parameter *S_rate* und *T_rate* des Ergebnis sind die gleichen, wie bei ***a***, sein Startzeitpunkt ist das Minimum der Startzeitpunkte der Elemente von Index **p1** bis Index **p2** in ***a***. Die Startzeitpunkte der Elemente des Ergebnis werden relativ zum seinem Startzeitpunkt neu berechnet und die Dauer wird entsprechend ***get_duration(a)*** gesetzt.

Auf Sequenzelementen:

diese Operationen ergeben implizit eine Sequenz mit den gleichen Werten, aber einem veränderten internen Zeiger. Daher wurde aus Gründen der Übersichtlichkeit auf die Angabe des Ergebnis "neue" Sequenz verzichtet.

- einen Zeiger auf das erste Element einer Sequenz **a** setzen:
  first (a): ***sequence*** → ***Element;***
- den Zeiger auf das nächste Element der Sequenz **a** setzen:
  next (a): ***sequence*** → ***Element;***
- den Zeiger auf das vorherige Element der Sequenz **a** setzen:
  prev (a): ***sequence*** → ***Element;***
- den Zeiger auf das Element mit dem Index **pos** in der Sequenz **a** setzen:
  set_pos (a, pos): ***sequence × INTEGER*** → ***Element;***
- anfügen des Elements **d** an die Sequenz **a**:
  add (a,d): ***sequence × Element*** → ***sequence;***

- löschen des Elements **d** an der aktuellen Position in der Sequenz **a**:
  del_elem (a): ***sequence*** → ***sequence;***
  Das Element an der aktuellen Position des Zeigers der Sequenz wird aus der Sequenz entfernt. Der Zeiger zeigt nach Ausführen der Operation auf das nachfolgende Element in der Sequenz. Andere Elemente werden nicht verändert. Diese Operation kann die Normalisierung nach der zweiten Normalform zerstören. Die Dauer der Sequenz kann durch diese Funktion verändert werden, deshalb sollte die Dauer der Sequenz neu berechnet werden.
- ein Element **d** an der aktuellen Position in der Sequenz **a** verändern:
  set_elem (a, d): ***sequence × Element*** → ***sequence;***
  Das Verändern der Elemente einer Sequenz kann so allgemein beschrieben werden, wie es hier oben steht. Dabei können aber, wie bereits mehrfach erwähnt, eventuell bestehende Normalformen oder die Gültigkeit der Sequenzparameter verletzt werden. Daher ist es angebracht, diese Operation in einzelne Veränderungen der Elementparameter aufzubrechen. An dieser Stelle wird besonders deutlich, daß die hier entwickelte Sequenz mehr ist, als eine Neuinterpretation einer bekannten Datenstruktur.

An der aktuellen Position in der Sequenz **a** :
Je nachdem, welche Normalform verlangt wird, dürfen *set_el_start* oder *set_el_duration* nicht ausgeführt werden oder haben unterschiedliche Implementierungen. Zum Beispiel muß bei der Voraussetzung der ersten Normalform, durch *set_el_start* die Reihenfolge überprüft und das Element eventuell neu einsortiert werden.

- den Wert **w** eines Elements verändern:
  set_el_value (a, w): ***sequence × W*** → ***sequence;***
- den Zeitpunkt **T** eines Elements verändern:
  set_el_start (a, T): ***sequence × INTEGER*** → ***sequence;***
- die Dauer **t** eines Elements verändern:
  set_el_duration (a, t): ***sequence × INTEGER*** → ***sequence;***

4.2 Strukturierung der Datenmengen kontinuierlicher Medien

Die einzige bisher vorgestellte Struktur kontinuierlicher Medien ist die Folge von Einzelwerten, die in *sequence* als Datentyp definiert wurde. Herausragendes Merkmal der Sequenz *sequence* sind ihre strengen Zeitbedingungen. Diese Darstellung ist bei Anwendungen, die nur Perzeption und Präsentation verbinden, wie zum Beispiel Videotelephon oder Videokonferenz, oder bei Anwendungen vom Prinzip Perzeption–Speicherung–Präsentation noch ausreichend. Weitergehende Anwendungen, die insbesondere die Verarbeitung der Daten betreffen, benötigen aber weitere Strukturierungen.

Bevor auf die Strukturierung direkt eingegangen wird, muß klargestellt werden, was strukturiert wird. Allgemein ist von Ausschnitten aus kontinuierlichen Medien die Rede. Die Präzisierung ist, daß zeitliche Ausschnitte von Medien betrachtet werden, die von anderen Medien unabhängig sind. Das können einerseits Monomedien wie zum Beispiel Audio oder Video sein, es können aber auch fest miteinander verkoppelte abhängige Medien wie zum Beispiel die beiden Kanäle von Stereo–Audio oder Audio und Video im Fernsehfilm (TV) und deren jeweilige Codierungen sein. Kontinuierliche Medien, die in ihren Eigenschaften differieren, sind unterschiedliche Medien, unabhängig von der Semantik der Werte der Medien.

4.2.1 Granularitäten aus Benutzersicht

Im vorliegenden Funktionsmodell arbeitet der Benutzer mit Daten, die einen Ausschnitt aus einem kontinuierlichen Medium repräsentieren. Diese Daten werden sowohl untereinander verknüpft als auch in ihren Werten verändert. Bei der Bearbeitung ergeben sich häufig mehrere Datenfolgen, die in einer engen, nicht notwendigerweise zeitlichen Beziehung zueinander stehen. Beispiele hierfür sind Versionen eines Vorgangs, wie im Filmbereich mehrere verschiedene Aufnahmen einer Szene, oder nacheinander kommende Stücke, wie im Filmbereich verschiedene Szenen oder im Audiobereich Sätze einer Symphonie oder Lieder auf einer Schallplatte oder CD. Als Ausdruck der Zusammengehörigkeit der Datenfolgen soll eine Liste von Datenfolgen dienen. Damit läßt sich eine Schallplatte oder eine CD als Liste von Audio–Sequenzen beschreiben, wobei die einzelnen Lieder, oder allgemeiner Musikstücke, die Sequenzen sind. Diese Stücke haben zwar untereinander eine Reihenfolge, aber die Zeit, die vom Ende des einen Stücks bis zum Anfang des nächsten vergeht, unterliegt keiner allgemeinen Zeitschranke sondern den technischen Gegebenheiten der Abspielgeräte. In der Liste der Sequenzen gelten daher keine strengen Zeitbedingungen. Mit der Liste der Sequenzen und der Möglichkeit Sequenzen zu konkatenieren und zu extrahieren, werden die grundlegenden Bearbeitungsaufgaben auf kontinuierlichen Medien unterstützt.

Neben diesem Zusammenhang der Sequenzen untereinander spielt für den Anwender aber auch die Granularität der Sequenzelemente eine wichtige Rolle. Wie bereits beschrieben, sind diese Elemente heute meist das direkte Abbild der technischen Repräsentation der Medien, wie zum Beispiel PCM–Samples für Audio oder Raster–Einzelbilder für Video. Repräsentiert die Sequenz noch ein kontinuierliches Medium über eine gewisse Dauer, so stellt ein Einzelbild keine Bewegung oder kontinuierliche Veränderung und ein Sample nicht einmal mehr einen Ton dar. Der Sprung vom kontinuierlichen Medium zum statischen Einzelwert, eventuell allein ohne jede Interpretationsmöglichkeit, ist für Verarbeitungsanwendungen zu groß. Daher muß mindestens ein weiterer Granularitätsschritt eingeschoben werden, der dieses Problem löst.

audio music opus note sample	audio_list track sector sample
visual video image animation frame	video_list video_scene video_block video_frame

Abb. 25 mögliche Granularitätsstufen für Audio und Video

Der Benutzer benötigt also eine Folge von logisch zusammengehörenden Einheiten, die einzeln interpretierbar sind [SteiFri91] [Gibbs94]. Diese werden auch als LDU (= logical data unit) bezeichnet [SteiFri91]. LDUs sind die Einheiten, mit denen der Benutzer arbeitet. Eine LDU zeichnet sich dadurch aus, daß sie die Parameter Zeitpunkt und Dauer besitzt, also gemäß dem entwickelten Datenmodell interpretiert werden kann. Die Größe der LDU, also

die Breite des Ausschnitts aus dem Medium, das sie beschreibt, ist anwendungsabhängig, ja genauer sogar benutzerabhängig. So bilden zum Beispiel auch die oben vorgestellten Listen LDUs auf Audio–Werken oder Film–Werken, beziehungsweise LDUs auf einer groben Granularitätsstufe. Die unterschiedlichen Granularitätsstufen können dabei wie in Abbildung 25 aussehen [SteiFri91], [FrBöBr92].

Die benutzerdefinierten LDUs und die Anzahl ihrer Hierarchiestufen können sogar für ein einzelnes Medium differieren. Der Benutzer einer Anwendung entscheidet jeweils selbst, welche LDUs mit welcher Hierarchie er verwenden will. In der Anwendung reichen drei Modellierungsmittel aus, um die Hierarchie abzubilden. Das sind die Listen–, die Sequenz– und die Elementebene.

Die folgende Modellierung der Granularitätsebenen geht davon aus, daß Sequenzen von Sequenzen unnötig sind, schließt eine Modellierung dieser Art aber nicht aus. In der Arbeitsweise des Anwenders wird dann vorausgesetzt, daß Operationen auf den Daten innerhalb einer Ebene vorgenommen werden, und Wechsel der Ebenen besondere Operationen sind. Die Beschreibung der Ebenen ist bewußt informell gehalten, um für beliebige Realisierungsmethoden offen zu bleiben. Für das Modell ist eine objektorientierte Realisierung besonders geeignet; insbesondere lassen sich in einer objektorientierten Umgebung die verschiedenen LDUs einfach ableiten.

4.2.2 Die Listenebene kontinuierlicher Medien

Auf der Listenebene kontinuierlicher Medien wird eine Datenstruktur bereitgestellt, die speziell dazu dient, zusammengehörige Ausschnitte eines kontinuierlichen Medii zusammenzufassen, gemeinsam zu speichern und zu bearbeiten. Auf dieser Ebene stellen die Listenelemente Daten dar, die die Parameter Wert, Zeitpunkt, Dauer und eventuell weitere enthalten. Die verschiedenen Listenelemente können heterogen sein, das heißt verschiedene Strukturen haben, unterschiedliche Codierungen verwenden u. s. w. Die Liste unterliegt in ihrer Interpretation keinen zeitlichen Randbedingungen. Die Startzeitpunkte der Listenelemente beschreiben nur die Verzögerung der Präsentation. Die Präsentation einer gesamten Liste ist nur die Folge der Präsentationen ihrer Elemente, die Gesamtdauer kann dabei deutlichen Schwankungen unterliegen. Die Elemente einer Liste können aber auch durch Einzelzugriff gleichzeitig parallel präsentiert werden.

Die Listen lassen sich gemäß dem entwickelten Datenmodell für Folgen von Daten kontinuierlicher Medii beschreiben. Folgende Konventionen sollen gelten:

1. Die Zeitpunkte der Listenelemente einer Liste gelten relativ zu einem gemeinsamen Ursprung, der nicht als Datum festgehalten wird.
2. Die Listenelemente sind schwach chronologisch geordnet.

Operationen auf der Listenebene:

1. die bekannten Listenoperationen zum Einfügen, Löschen und Suchen, "doppeltverzeigert", vorwärts und rückwärts;
2. sortiertes Einfügen nach Startzeitpunkten
   zur Erhaltung der schwachen Chronologie;
3. Erstellung der schwachen Chronologie, falls diese verletzt wurde;
4. Berechnung einer Mindestgesamtdauer aus den Einzeldauern.

Die Liste kontinuierlicher Daten ist ein Modellierungsmittel in Multimediaanwendungen, das es ermöglicht, dem Anwender die gemeinsame Verwaltung und Bearbeitung zusammengehöriger Ausschnitte eines kontinuierlichen Medii bereitzustellen. Diese Ausschnitte können zum Beispiel die gesammelten Spielfilme eines Regisseurs sein und damit eine sehr große Datenmenge, oder die Spektren aller Töne eines Flügels und damit eine wesentlich geringere Datenmenge. Weitere Beziehungen zwischen Ausschnitten aus zeitabhängigen Medien können anwendungsbezogen definiert werden. Zu ihrer Verwaltung werden Datenbanken herangezogen. Beispiele dazu sind in [Gibbs94] zu finden. Zur Verarbeitung kontinuierlicher Medien sollten die Listenelemente Sequenzen sein.

4.2.3 Die Sequenzebene kontinuierlicher Medien

Die Sequenzebene stellt die Arbeitsebene kontinuierlicher Medien dar. Auf dieser Ebene wird kopiert, geschnitten und gemischt. Zur Interpretation oder Präsentation wird jeder Sequenz genau ein Schrittgeber zugeordnet. Die Operationen auf Sequenzen wurden bereits ausführlich beschrieben. Daher sollen hier vor allem das Zusammenspiel zwischen der Elementebene und der Sequenzebene sowie zwischen der Sequenzebene und der Listenebene erläutert werden.

Für die Präsentation einer Sequenz gelten sehr strenge Zeitbedingungen und eventuell auftretende zeitliche Schwankungen in der Präsentation müssen für den Benutzer unmerklich sein. Daher stellen Sequenzen auch besondere Bedingungen an ihre Elemente. Diese Elemente müssen bezüglich ihrer Struktur und ihrer Wertcodierungen gleichförmig sein. Zum Beispiel dürfen in einer Sequenz nicht Audiodaten im MIDI–Format mit Audiodaten im CD–Format vermischt vorkommen. Daher muß der Elementtyp der Sequenz erkennbar sein. So wird auch die Interpretation der Elemente bei der Präsentation festgelegt. Die Gegenüberstellung einer Bildergruppe (GOP: group of pictures) in MPEG und einer Folge von DVI kodierten Bildern ist ein Beispiel für die Notwendigkeit der Gleichförmigkeit der Sequenzelemente, denn beide stellen auch Kompressionseinheiten dar und erfordern unterschiedliche Bearbeitung. Innerhalb einer Sequenz sollen alle Elemente nach dem selben Verfahren komprimiert beziehungsweise dekomprimiert werden, so können zum Beispiel die strengen Zeitbedingungen vorgeplant werden. Es ist weiterhin vorteilhaft, wenn die Sequenzelemente einheitlich gleiche Datenmengen haben; diese Bedingung ist aber nicht zwingend.

Im Zusammenspiel zwischen Listen und Sequenzen als Listenelementen ergibt sich folgendes. Extrakte aus Sequenzen können in den Listen von Sequenzen abgelegt werden. Aus den einzelnen Sequenzen der Listen lassen sich durch Konkatenation längere Sequenzen bilden. Zum Beispiel läßt sich aus der Liste der Töne eines Klaviers eine Melodie als Sequenz durch Konkatenation zusammensetzen.

Der Sequenzebene läßt sich eine breite Palette von LDUs zuordnen. Diese beschreiben zusammenhängende, kontinuierliche Medienstücke. "Kleine" oder feingranulare LDUs sind typischerweise Bewegungen oder Tonfolgen, modellierbar als Folgen von aufeinanderfolgenden Elementen mit unterschiedlicher Dauer. Für diese LDUs gilt allerdings, daß sie im Verhältnis zur Gesamtdauer des Ausschnitts aus einem kontinuierlichen Medium immer noch extrem kurz sind, beziehungsweise ein Ausschnitt aus einem kontinuierlichen Medium sehr viele dieser LDU’s in strenger zeitlicher Ordnung enthält. Da diese LDU’s einerseits als Sequenzen von Bildern, Tönen oder Samples angesehen werden können,andererseits noch so

kurz sind, daß sie als Sequenzelemente anderer, abstrakterer LDU's dienen können, stellt sich die Frage, ob ein langer Ausschnitt aus einem kontinuierlichen Medium nicht als Sequenz von Sequenzen dargestellt werden sollte.

Diese Sicht der Sequenz von Sequenzen entspricht genau der Arbeitsweise der Anwender, ist aber in ihrem Aufbau extrem anwendungsabhängig. Darüber hinaus ist sie bei der Perzeption im allgemeinen nicht algorithmisch erzeugbar und nur unter großem Aufwand bei der Präsentation interpretierbar. Viele vom Anwender benötigte Funktionen lassen sich aber auch einfach auf der Ebene der Sequenzen realisieren, ohne Sequenzen von Sequenzen zu verwenden. Dies ist eine Abgrenzung nach unten, Sequenzelemente sollten nicht wieder Sequenzen sein.

4.2.4 Die Elementebene kontinuierlicher Medien

Als Elemente werden die kleinsten LDU's oder die kleinsten mit Zeitparametern ausgestatteten Einheiten modelliert, die eine interpretierbare Teilinformation des kontinuierlichen Medii darstellen. Die Elemente können außer den Zeitparametern weitere Parameter haben; ihre Werte können strukturiert sein. Die Interpretierbarkeit bedeutet, daß die Elemente bei einer Dauerpräsentation für den Anwender als Teil des Medii erkennbar sein sollten, zum Beispiel ein Dauerton oder ein Einzelbild in Dauerpräsentation als Standbild. Daher kann ein einzelner Sample kein Element sein, obwohl man ihm Zeitparameter zuordnen kann.

Bei Audio stellt ein Sample die feinste technische Auflösung dar. Bei Video kann dagegen ein Einzelbild in Blöcke von Bildpunkten weiter zerlegt werden, denen theoretisch jeweils wieder Interpretation und Zeitparameter zugeordnet werden können. Daraus ließe sich eine Arbeitsweise, wie sie bei der Animation verwendet wird, ableiten, wo Bildfolgen aus bewegten und unbewegten Bildelementen erstellt werden. Hier erhalten die Elemente aber eine zusätzliche Eigenschaft der Ortsabhängigkeit, während ihre Zeitparameter weiterhin vom Gesamtbild beeinflußt werden. Eine weitere Betrachtung führt allerdings zu tief in spezielle Anwendungen und Codierungen, so daß in dieser Arbeit nicht näher darauf eingegangen werden kann.

Generell stellt sich hier auch das Problem der Feinheit der Elemente. Diese ist zwar grundsätzlich davon abhängig, was in der Anwendung vom Benutzer noch interpretierbar ist. Aber darüber hinaus bilden auch die Zeitparameter eine Restriktion der Feinheit. Sowohl die Dauer eines Elements als auch die Differenzen der aufeinanderfolgenden Startzeitpunkte, also die Elementabstände, können nicht beliebig fein werden. Sie sind im Zeitbezugssystem jeweils durch mindestens einen Schritt festgelegt. Die Schrittgröße muß dabei berücksichtigen, daß zwischen zwei Schritten genügend Rechenzeit für alle Prozesse verstreichen kann, wie im Abschnitt über das Zeitbezugssystem bereits beschrieben. Somit gelten auch zeitlich "zu kurze" Strukturen als nicht mehr interpretierbar und damit nicht als Element.

Elemente können einzelne Töne oder einzelne Bilder sein. Auch die Kompressionseinheiten der Daten für kontinuierliche Medien bilden Elemente im Sinne kleinster Einheiten, da unterhalb der Kompressionseinheiten keine weitere allgemeine Ebene angegeben werden kann. Denn die Kompressions–Dekompressions–Verfahren benutzen unterschiedliche Strukturen und liefern unterschiedlich viele und verschieden strukturierte Teilelemente. Daher ist die für Elemente zu fordernde allgemeine Beschreibung von Operationen, auf den Kompressionseinheiten oder der Elementstruktur nicht möglich und direkt von der Anwendung abhängig.

4.2.5 Anwendung der Modellierungsebenen

Die Vorstellung von drei Modellierungsebenen bedeutet nicht, daß jede Granularitätshierarchie auch genau drei Stufen umfassen muß. Es soll aber zum Ausdruck gebracht werden, daß in den meisten Fällen diese drei Stufen ausreichen. Für mehr als dreistufige Hierarchien kann sowohl die Listenebene als auch die Sequenzebene wiederholt werden; wobei die Wiederholung der Listenebene impliziert, daß keine zeitabhängigen Beziehungen zwischen den Listenelementen bestehen, die Wiederholung der Sequenzen dagegen den kontinuierlichen Charakter der Sequenzelemente hervorhebt.

Daraus ergibt sich, daß bei der Wiederholung einer Ebene nur die zuletzt verwendete Ebene wiederholt werden kann. Ein Abwechseln der Listen– und Sequenzebenen darf wegen deren semantischen Unterschiede nicht auftreten. Im folgenden werden einige Modellierungsvarianten vorgestellt, die auch auf Beispiele für mehrstufige Hierarchien eingehen.

Mögliche Sequenzelemente sind eine Tonfolge oder die Video Aufzeichnung einer Bewegung. Diese entsprechen eher dem Begriff des Teils eines kontinuierlichen Medii oder einer LDU dafür, da sie selbst noch eine Veränderung enthalten. Diese Sequenzelemente können wie der Sektor einer Audio–CD eine feste Anzahl von Teilelementen (Werten) enthalten, aber auch wie eine Bildergruppe in MPEG variabel lang sein. Erst ihre infinitesimale Reduktion führt auf gerade noch interpretierbare Elemente, wie Einzelbilder und Einzeltöne, denen Zeitattribute zugeordnet werden können. Nach der Definition der Elemente als kleinste interpretierbare Teilinformation müssen also Töne und Bilder die Elemente bilden, falls sie einzeln zugreifbar sind. Die Varianten Bewegung oder Tonfolge können somit nur bei gemeinsamer Codierung als Element auftreten. Andernfalls sind sie strukturierte Sequenzelemente, die selbst wieder Sequenzen sind.

Für die Modellierung der Elemente Tonfolge oder Bewegung als Sequenz in einer Sequenz spricht, daß durch die Teilelemente direkt eine technisch tiefe Schicht mit ihren Zeitparametern modelliert wird, und dadurch sowohl die Benutzersicht als auch die Speicherstrukturen berücksichtigt werden. Die so entstehenden Hierarchien können vierstufig sein , wie:

Audioliste — Audiosequenz — Tonsequenz — Ton

oder

Videoliste — Videosequenz — Bewegungssequenz — Bild.

Nach der Definition des Elements als kleinster mit Zeitparametern ausgestatteter Medienteil ergibt sich ein Widerspruch dazu, Elemente als Sequenz eben solcher Elemente darzustellen, das heißt, es muß genau unterschieden werden, ob die Sequenzelemente Elemente im Sinn der Elementebene sind, oder beliebige Strukturen mit den geforderten Sequenzelementparametern. Gegen die Modellierung von Elementen als Sequenz spricht weiterhin, daß der Anwender nur selten alle vier Ebenen parallel benutzt, sondern vielmehr meist auf einer Ebene arbeitet und die nächsthöhere und die nächsttiefere dazu benutzt. Aus Effizienzgründen bietet es sich daher an, mehrere verschiedene Hierarchien nebeneinander zu benutzen. Diese könnten sein:

Audioliste — Audiosequenz — kodierte Tonfolge (Element)
Tonfolgenliste — Tonsequenz — Ton (Element)

oder

Videoliste — Videosequenz — kodierte Bewegung (Element)
Bewegungsliste — Bewegungssequenz — Bild (Element).

Durch diese parallelen Hierarchien wird aus der Bildung von Sequenzen von Sequenzen die Konkatenation von Sequenzen, was in der Handhabung leichter, im Ergebnis aber gleich ist. Wie dem Benutzer die einzelnen Ebenen präsentiert werden, ist von dieser programmiertechnischen Modellierung unabhängig.

4.3 Audio und Video mit Zeitparametern

Im folgenden werden für Audio und Video Beispielmodellierungen vorgestellt, die auf dem entwickelten Datenmodell basieren. Zunächst werden die LDUs vorgestellt, die der Benutzer verwendet. Danach wird gezeigt, wie diese im Datenmodell realisiert werden können.

4.3.1 Modellierungsbeispiele für Audio und Video

Der Benutzer verwendet in diesem Beispiel in seiner Anwendung Audio und Video in jeweils drei verschiedenen Granularitätenstufen. Die Codierungen seiner Daten sind für die Töne in Audio Spektren eines Klaviers, eine Spektrum für jeden Ton, und für Video einerseits MPEG kodierte GOPs andererseits Einzelbilder. Folgende Elemente sollen auf den verschiedenen Granularitätsstufen verfügbar sein.

- gröbste Granularität:
  - eine Bibliothek von Tönen eines Klaviers
  - eine Sammlung von Melodien
  - zwei Sammlungen SV1 und SV2 von Videoszenen,
    je eine für MPEG–Sequenzen und eine für Einzelbildfolgen;
- mittlere Granularität:
  - Tonfolgen
  - MPEG–Sequenz,
    so um Startzeit und Dauer der GOPs erweitert, daß sie als Sequenz im Sinne der Definition dieses Kapitels benutzt werden können.
  - Einzelbildfolgen;
- feinste Granularität:
  - Spektren
  - MPEG GOPs
  - Einzelbilder

4.3.2 Abbildung auf Listen, Sequenzen und Elemente

Die hier benutzten Granularitätsebenen lassen sich direkt den Modellierungsebenen zuordnen. Auf der gröbsten Granularitätsebene werden Listen verwendet, auf der mittleren Sequenzen und auf der feinsten Elemente. Damit ergibt sich folgende Modellierung.

```
ELEMENT: Spektrum, MPEG_GOP, Einzelbild;
SEQUENCE of Spektrum:   Tonfolge;
SEQUENCE of MPEG_GOP:   MPEG_Sequenz;
SEQUENCE of Einzelbild: Einzelbildfolge;
LIST of Tonfolge:       Bibliothek /*einelementige Sequenzen*/;
LIST of Tonfolge:       Melodien;
LIST of MPEG_Folge:     SV1;
LIST of Einzelbildfolge:SV2;
```

Alternativ ergibt sich bei Verwendung einer Klassenhierarchie (von links nach rechts gelesen absteigend):

```
ELEMENT: AUDIO_ELEMENT: Spektrum,
         VIDEO_ELEMENT: MPEG_GOP, Einzelbild;
SEQUENCE of AUDIO_ELEMENT:      Tonfolge;
SEQUENCE of VIDEO_ELEMENT:      motion
                        /* unterschiedliche Codierungen */
                        /* dürfen nicht gemischt werden */;
LIST of Tonfolge:       Bibliothek, Melodien;
LIST of motion:         SV1, SV2;
```

oder auch:

```
ELEMENT: AUDIO_ELEMENT: Spektrum,
         VIDEO_ELEMENT: MPEG_GOP, Einzelbild;
SEQUENCE of AUDIO_ELEMENT:      Tonfolge;
SEQUENCE of VIDEO_ELEMENT:      motion
                        /* unterschiedliche Codierungen */
                        /* dürfen nicht gemischt werden */;
LIST of SEQUENCE:       Bibliothek, Melodien, SV1, SV2
                        /* unterschiedliche Codierungen */
                        /* dürfen nicht gemischt werden */;
```

Auffallend ist zunächst, daß sowohl Einzelbilder als auch MPEG GOPs als Elemente modelliert werden. MPEG GOPs bilden auf Grund ihrer Kompression eine zunächst nicht weiter teilbare Einheit. Eine Bildergruppe in MPEG läßt sich nach der Dekomprimierung als Sequenz von Einzelbildern darstellen und ergibt somit die Struktur einer Sequenz von Sequenzen. Besser, weil übersichtlicher, ist allerdings, die dekomprimierte GOP als Sequenz eines anderen Elementtyps aufzufassen und die dekomprimierten GOPs in einer eigenen Liste zu sammeln. Diese weitere Liste kann hier direkt die Liste der Einzelbilderfolgen sein.

Für verschiedene Medien können auf verschiedenen Granularitätsebenen schon einige Strukturen vordefiniert werden. Für Sequenzen kann eine **`SEQUENCE of AUDIO_ELEMENT`** als **`track`** und eine **`SEQUENCE of VIDEO_ELEMENT`** als **`V_scene`** eingeführt werden. Als strukturierte Elemente einer Sequenz ist für Audio der **`sector`** und für Video der **`Videoblock`** vorgesehen. Die hier definierten strukturierten Elemente verhalten sich nach außen wie ein ARRAY, wobei für den **`sector`** entsprechend einem CD–Sector 1176 Samples vorgesehen sind, für den **`Videoblock`** dagegen keine feste Anzahl angenommen wird. Ein **`Videoblock`** kann zum Beispiel eine MPEG GOP sein; diese muß beim Zugriff auf ein Teilelement erst eine Dekomprimierung durchführen. Die vordefinierten Strukturen können als generische LDUs für die Anwender kontinuierlicher Medien angesehen werden.

4.4 Integration des Datenmodells in das Funktionenmodell

In diesem Abschnitt werden die Operationen des Datenmodells den Funktionen auf kontinuierlichen Medien zugeordnet. Es geht hier also um die datenflußorientierten Funktionen und ihre Beziehungen zum abstrakten Datenmodell der LDUs, den Listen, Sequenzen und Sequenzelementen sowie dem Ticker und den Schrittgebern. Dieser Abschnitt soll insbesondere die Idee der Verzahnung zwischen den Anwendungsbausteinen und den Datenoperationen unter Berücksichtigung der verschiedenen Granularitätsebenen und die Entkopplung der Zeit von den Daten vermitteln. Dabei ist der kontinuierliche Charakter der Sequenz von dem diskreten der Sequenzelemente und der Listen zu unterscheiden. Zusätzlich zu den Operationen auf dem Datenmodell müssen die verschiedenen Funktionen die Übertragung der Daten realisieren, wobei sie die durch das Zeitbezugssystem gesetzten Restriktionen einhalten müssen.

Die generelle Beziehung zwischen den Funktionen und den Operationen des Datenmodells besteht darin, daß die Funktionen verschiedene Operationen realisieren. Dabei ist einerseits zu berücksichtigen, daß die einzelnen Funktionen nur Untermengen der Operationen realisieren, andererseits unterscheiden sich diese Realisierungen in der Persistenz der Operationsergebnisse beziehungsweise in deren Zeitabhängigkeit.

4.4.1 Perzeption und Perzeptionsumgebung

Die Perzeption stellt Daten zur Verfügung, indem sie Signalparameter der Umwelt über einen Sensor mißt. Dabei entstehen zu diskreten Zeiten einzelne Datenwerte. Diese Datenwerte bilden die Elementebene der Perzeption. Gleichzeitig realisiert die Perzeption einen Schrittgeber, mit der Einschränkung, daß dieser nur vorwärts zählen oder explizit auf einen bestimmten Wert (z. B. null) gesetzt werden kann.

Die Perzeptionsfunktion kann zu jedem Schritt den aktuellen Datenwert übertragen, oder erst dann einen Wert übertragen, wenn der Datenwert sich ändert, oder über mehrere Schritte einen Mittelwert bilden und diesen nach einer gewissen Anzahl von Schritten übertragen. Es können beliebige weitere Strategien verwendet werden. In diesem Zusammenhang ist zu berücksichtigen, daß nur bei der Kombination *zu jedem Schritt ein Datum übertragen*, die Zeitpunkte und die Dauern der Elemente ex ante bestimmbar sind und die Daten mit geringster Verzögerung erhalten werden. Im übrigen haben die unterschiedlichen Verfahren zur Datenbestimmung auch Auswirkungen auf die Normalform der Datenfolgen. Bei kontinuierlich laufendem Schrittgeber erzeugt eine Perzeptionsfunktion, die zu jedem Schritt ein Datum überträgt, einen kontinuierlichen Datenstrom. Andere Kombinationen von Datenerzeugung und Übertragung führen zu Verzögerungen und eventuell stark schwankenden Datenraten.

Die Kombination zu jedem Schritt ein Datum läßt sich mit dem Parameter ***data_steps*** = k erweitern, um zu jedem k-ten Schritt ein Datum zu übertragen. Bei dieser Strategie sollte der Perzeptionsfunktion als weiterer Parameter die einem Datum zuzuordnende Gültigkeitsdauer ***duration*** angefügt werden.

Die Perzeptionsfunktion mit den Parametern ***start_tick***, ***rate***, ***data_steps*** und ***duration*** realisiert folgende Operationen:

- ***init (start_tick, rate, data_steps, duration):***
  INTEGER × INTEGER × INTEGER × INTEGER→ Ø;
  initialisieren des Schrittgebers mit Starttick und Schrittrate. Der Schrittgeber beginnt sofort gemäß dem Ticker zu zählen.
- ***set_rate (rate):*** ***INTEGER*** → ***Ø;***
  setzen der Schrittrate. Hierbei kann das Vorwärtszählen des Schrittgebers verlorengehen, so daß der nächste Schritt nicht genau den nachfolgende Wert des vorhergehenden Schritts erhält (nicht: n und n+1). Falls die neue Schrittrate null ist, also der Schrittgeber ruhen soll, ist nichts weiter zu berechnen. Andernfalls muß mit dem aktuellen Tickwert nach der Schrittgeberformel auf Seite 78 dazu ein neuer Starttick berechnet und gesetzt werden.
- ***get_rate ():*** ***Ø*** → ***INTEGER;***
  lesen der Schrittrate.
- ***set_start_tick (tick):*** ***INTEGER*** → ***Ø;***
  setzen des Startticks. Diese Operation beeinflußt direkt den Wert des nächsten Schritts nach der Formel auf Seite 78. Damit keine paradoxen Wertefolgen im Schrittgeber auftreten, sollte diese Operation nur zu Zeiten des Ruhens des Schrittgebers oder des Tickers verwendet werden.
- ***get_start_tick ():*** ***Ø*** → ***INTEGER***;
  lesen des Startticks.
- ***stop ():*** ***Ø*** → ***Ø;***
  beenden der Perzeption.

Weitere Operationen der Perzeptionsfunktion sind:

- ***send_sequence_parameters(tickrate):INTEGER→ Ø;***
  diese Funktion sendet die Werte der übergebenen Tickrate, der Schrittrate und des Start–Schritts sowie eine 0 als Datenwerte für den Start einer Sequenz. Diese Werte entsprechen den ersten vier Parametern einer Sequenz (vgl. Seite 80). Die übergebene Tickrate wird außerdem in der Berechnung der Schritte nach der Formel auf Seite 78 verwendet.

und falls die entsprechende Strategie verwendet wird:

- ***set_data_steps (steps):INTEGER*** → ***Ø;***
  festlegen, nach wievielen Schritten ein neuer Datenwert erzeugt und übertragen wird. Das entspricht dem Abstand der Startzeitpunkte der Daten.
- ***set_duration (duration):*** ***INTEGER*** → ***Ø;***
  festlegen, der Dauer eines Datenwerts.
- ***get_data_steps ():*** ***Ø*** → ***INTEGER;***
  feststellen der aktuellen Schrittweite.
- ***get_duration ():*** ***Ø*** → ***INTEGER;***
  feststellen der aktuellen Dauer.

Wie man an den Operationen erkennen kann, gibt es an der Perzeption die Ebenen Sequenz und Element. Erst die Abfolge der Elemente, die die Perzeption liefert, ergibt eine Sequenz. Solange die Bedingung des Vorwärtszählens erfüllt ist sogar in erster Normalform. Es bleibt dem Benutzer einer Perzeptionsfunktion überlassen, welche Folgen von Elementen er als Sequenz zusammenfassen, oder durch die Übertragung neuer Sequenzparameter in verschiedene Sequenzen aufteilen will.

Verschiedene Perzeptionsfunktionen unterscheiden sich vor allem im Typ ihrer Elemente. Sie können weitere Operationen anbieten, die die Gewinnung der Daten aus der Umwelt beeinflussen.

Die Operationen auf dem Ticker werden nicht von der Perzeptionsfunktion selbst, sondern von der Perzeptionsumgebung realisiert, da der Ticker allen in die Umgebung eingebetteten Perzeptionsfunktionen gemeinsam sein soll. Wie der Schrittgeber einer Perzeptionsfunktion muß auch der Ticker einer Perzeptionsumgebung stets vorwärtszählen oder explizit auf einen neuen Wert gesetzt werden. Damit Effekte wie Zeitraffer– oder Zeitlupenaufnahmen direkt bei der Perzeption erreicht werden können, muß die Perzeptionsumgebung auch die Ergänzungen zum Ticker realisieren.

4.4.2 Präsentation und Präsentationsumgebung

Die Präsentation erwartet Sequenzen in 1NF von einer Datenquelle in der Form der Übertragung der Sequenzparameter (vgl. ***send_sequence_parameters*** der Perzeptionsfunktion) und einer Folge von Elementen. Die Tick– und Schrittraten werden von den Sequenzparametern übernommen und als Grundrate oder Normalgeschwindigkeit eingestellt. Dabei wird eine Anpassung an die Tickrate der Präsentationsumgebung notwendig, falls deren Tickrate von der der Sequenz abweicht. Die Perzeption stellt die Sequenzelemente gemäß deren Startzeitpunkte und Dauern in Abhängigkeit ihres Schrittgebers dar. Dabei garantiert die erste Normalform, daß die Sequenz linear durchlaufen werden kann. Die Präsentation eines Elements darf noch nach dem Startzeitpunkt anfangen, aber sie darf sich nicht über das durch die Summe aus Startzeitpunkt und Dauer festgelegte Ende erstrecken.

Da Elemente weder Tick– noch Schrittraten als Parameter haben, werden sie entweder im Rahmen einer Sequenz von der Präsentation verarbeitet, oder gemäß der Schrittgeberparameter der Präsentation, die über die Schrittgeberoperationen veränderbar sind, interpretiert. Die Präsentation hat also zusätzlich zu den Schrittgeberparametern ***start_tick***, ***rate*** und ***step***, die aktuellen Sequenzparameter ***seq_s_rate, seq_t_rate, seq_start_step*** und ***seq_duration*** verfügbar. Die Sequenzparameter werden jeweils durch die Übertragung einer neuen Sequenz festgelegt und verändern dabei gleichzeitig die entsprechenden Schrittgeberparameter.

Somit realisiert die Präsentation folgende Operationen:

- lesen der Sequenzparameter:
  get_seq_s_rate: ***Ø*** → ***INTEGER***
  get_seq_t_rate: ***Ø*** → ***INTEGER***
  get_seq_start_step: ***Ø*** → ***INTEGER***
  get_seq_duration: ***Ø*** → ***INTEGER***
- initialisieren des Schrittgebers mit Starttick und Schrittrate:
  init (start_tick, rate): ***INTEGER × INTEGER*** → ***Ø;***

- setzen Schrittrate:
  set_rate (rate): ***INTEGER*** → ***Ø;***
- lesen Schrittrate
  get_rate (): ***Ø*** → ***INTEGER;***
- setzen des Startticks:
  set_start_tick (tick): ***INTEGER*** → ***Ø;***
- lesen des Startticks:
  get_start_tick (): ***Ø*** → ***INTEGER***;
- beenden:
  stop (): ***Ø*** → ***Ø;***

Bei der Präsentation sind auf der Elementebene noch einige Besonderheiten zu berücksichtigen. Eine Dauerpräsentation, wie zum Beispiel ein Standbild im Film, läßt sich durch die Operationen auf dem Zeitbezugssystem erreichen, indem man den Ticker oder den Schrittgeber ruhen läßt (rate=0).

Als Basis des Zeitbezugssystems wird der Ticker in der Präsentationsumgebung realisiert. Die Operationen des erweiterten Tickers sind Operationen der Präsentationsumgebung und beeinflussen alle eingebetteten Präsentationsfunktionen gleichermaßen. Bei einer Live–Präsentation werden die Umgebungen der Perzeption und der Präsentation gekoppelt. Diese Kopplung bedeutet, daß die Tickeroperationen in beiden Umgebungen wirksam werden, unabhängig davon, in welcher sie aufgerufen wurden. Daraus resultiert ein gemeinsames Zeitbezugssystem für die Live–Präsentation ohne, daß Ticks übertragen werden müssen.

4.4.3 Speicherung

Die Speicherung wurde im Modell der Funktionen kontinuierlicher Medien zunächst als eine Funktion modelliert und später datenflußorientiert in schreibende und lesende Speicher aufgeteilt. In diesem Abschnitt kann sie wieder weitgehend geschlossen betrachtet werden. Allgemein gilt, daß die Speicherfunktion genau die abstrakten Operationen realisiert, die im Datenmodell für die Ebenen Liste, Sequenz und Element vorgestellt wurden. Diese Realisierungen sind persistent und zeitunabhängig in dem Sinne, daß die selben Operationen zu verschiedenen Zeitpunkten stets die gleichen Ergebnisse liefern, sofern keine beeinflussenden Änderungsoperationen dazwischen ausgeführt wurden. Da mit dem Speicher keine zeitabhängige Umgebung definiert wurde, existiert auch kein Zeitbezugssystem, auf dem Operationen ausgeführt werden könnten. Speicherfunktionen lassen sich danach klassifizieren, welche Normalform sie unterstützen. Entsprechend unterscheiden sich die Implementierungen der einzelnen Funktionen, wie bereits beschrieben.

Der Speicher kennt Listen von Sequenzen, Sequenzen und Sequenzelemente. Im Gegensatz zu einem Datenbanksystem ist die Speicherfunktion für kontinuierliche Medien nur eine Zugriffsschnittstelle, die die Kontinuität auf Sequenzebene realisiert und im Zugriff die Zeitbedingungen einhält. Dabei können und dürfen Ungleichmäßigkeiten im Sinne von Vorgriffen auftreten. Operationen, wie die Auswahl bestimmter Listen oder Sequenzen, die Angabe von Inhaltsverzeichnissen oder Speicherorten, sind nicht die Aufgabe der Speicher–(Zugriffs–) Funktion, sondern ihrer Umgebung. Diese Umgebung beinhaltet ein oder besteht günstigerweise in einem (Multimedia–) Datenbank–System.

Die Veränderung der Dauer eines Elements ist eine Funktion des Speichers und wurde bisher nur "nach oben", zur Sequenz hin, untersucht. Sie kann sich aber auch "nach unten" auf den Aufbau des Elements hin auswirken, wenn dieses aus Teilelementen besteht, also beispielsweise wieder als Sequenz betrachtet werden kann. Insbesondere werden Veränderungen dann nötig, wenn die Elemente als Sequenzen bestimmten Normalformen genügen sollen, wie zum Beispiel ein Audio-CD-Sektor oder eine MPEG–GOP. Da diese Elemente bezüglich ihrer Teilelemente in 3NF sind, ändert sich durch eine Änderung der Anzahl der Teilelemente "nach oben" die Dauer des Elements; eine Änderung der Dauer es Elements muß dann "nach unten" auch eine Änderung der Anzahl der Teilelemente bewirken. Sollen, wie beim Audio–Sektor, stets eine feste Anzahl von Teilelementen mit einer festen Dauer ausgegeben werden, müssen zusätzlich Ausgleichsfunktionen zwischen aufeinanderfolgenden Elementen verwendet werden [Bast93]. Das Verhältnis zwischen alter und neuer Anzahl muß dem Verhältnis zwischen der definierten und der aus der alten Teilelementanzahl berechneten Dauer entsprechen. Im allgemeinen müssen dafür alle Teilelemente durch Interpolation neu berechnet werden. Verschiedene Interpolationen für Audio werden unter anderem in [Bast93] untersucht. Die Änderung der Anzahl der Teilelemente kann so lange hinausgeschoben werden, bis entweder eine andere Operationsrichtung ("nach oben") eintritt, oder die Daten zur Präsentation kommen. Dafür wird eine besondere Ausgleichsfunktion vorgesehen.

Wie die Berechnung genau vorgenommen wird, hängt davon ab, welche Effekte erzielt werden sollen. So kann schnelles Audio einfach in kürzeren Tönen gleicher Höhe bestehen, oder wie eine zu schnell laufende Schallplatte auch die Töne selbst verändern. Welcher Effekt erzielt werden soll, bestimmt der Anwender durch Verwendung einer entsprechenden Funktion.

Der zunächst notwendige Mehraufwand durch eine besondere Funktion zur Anpassung der Dauer und der Anzahl der Teilelemente birgt verschiedene Vorteile in sich. Zunächst einmal läßt sich die Anpassung automatisieren und ein eventueller Rundungsfehler bei den Zeitattributen auf maximal einen Schritt beschränken. Außerdem kann bei der Speicherung eine Datenreduktion ausgenutzt werden, wenn die Teilelemente nur geringe Unterschiede aufweisen, die durch die Interpolation genügend genau rekonstruiert werden können.

Die bisher vorgestellten Auswahloperationen der Elemente in einer Sequenz waren streng datentyporientiert und daher rein strukturell. Die Einbeziehung der Elementeigenschaften Startpunkt und Dauer legt eine Auswahl auch nach diesen Kriterien nahe. Abhängig vom Normalisierungsgrad der Sequenz sind zu deren Realisierung unterschiedliche Suchverfahren anzuwenden. Falls die Sequenz in 3 NF ist, kann aus der Startzeit direkt das Element berechnet werden. Die Suche nach einem vorgegebenen Elementwert kann aufgrund der zugelassenen Vielfalt der Elementtypen nicht als generelles Operationsangebot des Speichers eingeführt werden.

4.4.4 Verarbeitungsfunktionen

Verarbeitungsfunktionen können für alle Datenebenen realisiert werden. Da die abstrakten Operationen auf Listenebene und Sequenzebene durch die Speicherfunktion abgedeckt sind, besteht die eigentliche Aufgabe für Verarbeitungsfunktionen in der Veränderung der Sequenzelemente. Beispiele für Änderungen sind Verändern der Lautstärke bei Audio, Verän-

dern der Helligkeit bei Video oder auch Umkodierungen. Veränderungen der Dauer und/oder der Anfangszeiten sind Operationen auf der Sequenzebene (***set_el_start*** und ***set_el_duration***) und damit der Speicherfunktionen.

Die möglichen Operationen auf den Elementen und ihre Realisierungen werden vom repräsentierten Medium (Audio, Video ...) und von dessen Codierung bestimmt. Damit sind sie anwendungsspezifisch und können nicht allgemein näher gefaßt werden. Eine Verarbeitungsfunktion kann auch mehrere Operationen gleichzeitig oder nacheinander durchführen, sowie verschiedene Operationen alternativ anbieten. Zur Vereinfachung werden die Funktionen so behandelt, als würden sie nur eine Operation realisieren, da Kombinationen der Operationen über die Kombination der Verarbeitungsfunktionen im Funktionenmodell abgedeckt sind.

Neben den datenverändernden Operationen auf einem einzelnen Medium gehört zu den Verarbeitungsfunktionen auch das Mischen und Verteilen mehrerer Ebenen. Diese Funktionen sind allgemein bereits ausreichend behandelt, für die verschiedenen Codierungen muß jeweils eine eigene Mischfunktion implementiert werden, so daß an dieser Stelle nicht näher darauf eingegangen wird.

4.5 Beziehungen zwischen Sequenzen

Beziehungen zwischen Sequenzen handeln von der Synchronisation der Anwendung. Die Synchronisation kann auf vielen Ebenen untersucht werden. Da diese Arbeit sich mit Anwendungsmodellierung beschäftigt, werden Synchronisation auf Betriebssystemebene und auf Kommunikationsebene nicht weiter behandelt; es wird lediglich an einigen Stellen auf diese Erfordernisse hingewiesen. Bei der Übertragung und Auswertung der Ticks und der Generierung der Schritte spielen die weiteren Aspekte der Synchronisation allerdings eine wichtige Rolle.

4.5.1 Aspekte der Verwendung mehrerer Sequenzen

Das Wesentliche an multimedialen Systemen ist die Präsentation verschiedener zeitabhängiger und zeitunabhängiger Medien. Dabei treten diverse Probleme auf. Zum einen ist die Verarbeitung der großen Datenmengen der einzelnen Medien zeitkritisch, zum anderen sind Präsentationsanfang und –ende sowohl der zeitabhängigen als auch der zeitunabhängigen Medien aufeinander bezogen und voneinander abhängig. Die aus beiden Punkten resultierenden Zeitanforderungen werden durch zwei verschiedene Arten der Synchronisation beschrieben. Die **Intrasynchronisation** stellt die richtige zeitliche Abfolge der zeitabhängigen Daten eines Medii sicher. Die **Intersynchronisation** dagegen garantiert die richtigen zeitlichen Relationen zwischen verschiedenen Medien.

Die Intrasynchronisation wird bei der Präsentation durch das Zeitbezugssystem des Ticker–Schrittgeber–Modells gewährleistet. Die Güte der Synchronisation hängt von der Hardware– und Betriebssystemunterstützung auf der Rechnerplattform ab. Weiterhin sind bei verteilten Anwendungen die auftretenden Übertragungsschwankungen beim Transport der Daten zu berücksichtigen. Diese Problematik wird in verschiedenen Arbeiten behandelt [Herrtw90] [Ferrari91] [LittlKao92]. Wie die verbleibenden Verzögerungen in der Anwendung berücksichtigt werden können, ist in [Unge92] dargestellt.

Zur Entwicklung tatsächlicher *Multi*–Media–Anwendungen ist ein Konzept für die Intersynchronisation notwendig. Grundsätzlich gibt es dafür zwei Möglichkeiten. Die eine ist die Bildung abhängiger Medien, indem verschiedene Medien gemeinsam in einer Sequenz kodiert und interpretiert werden; wie zum Beispiel Audio und Video im Fernsehsignal. Die andere ist, die Medien unabhängig in verschiedenen Sequenzen zu kodieren und zusätzlich eine Beschreibung ihrer Abhängigkeiten zu erstellen. Die Vor– und Nachteile abhängiger und unabhängiger Medien wurden bereits behandelt. Die verbleibende Aufgabe der Intermediasynchronisation ist, die zeitlichen Relationen zwischen den unabhängigen Medien zu beschreiben und diese Beschreibung zu interpretieren.

4.5.2 Intersynchronisation

Für die Beschreibung der Intermediasynchronisation müssen drei Punkte unterschieden werden. Der erste ist das vom Benutzer definierte Verhältnis zwischen den zu synchronisierenden Medien, der zweite ist dessen interne Repräsentation. Die vom Benutzer definierten Beziehungen können mit einer beliebigen Methode erstellt werden, da die in [Unge92] aufgezeigten Äquivalenzen stets eine Umsetzung in die ebenfalls in [Unge92] entwickelte interne Repräsentation ermöglichen. Den dritten Punkt bilden die Verfahren zur Realisierung der definierten Synchronisation. Eine vergleichbare Einteilung wird in [PerLitt95] vorgestellt.

Im Ticker–Schrittgeber–Modell ist die Intersynchronisation durch die Start–Ticks der Schrittgeber bezüglich einer gemeinsamen Zeitachse und die Start–Schritte der Sequenzen beschrieben. Die Werte der Startticks lassen sich aus der internen Repräsentation der Intersynchronisation von [Unge92] direkt berechnen. Die Realisierung der Synchronisation ist durch die Anbindung der Präsentationsfunktionen an einen gemeinsamen Ticker gewährleistet, sie kann ohne zusätzliche Kommunikation zwischen den verschiedenen Präsentationsfunktionen erfolgen. Alle Synchronisationsbeschreibungen lassen sich in das Ticker–Schrittgeber–Modell umsetzen.

Neben diesen von den Startticks bestimmten zeitlichen Relationen untereinander kann eine bestimmte Präsentationsfolge auch durch die Ausführung von Schrittgeberoperationen zu festgelegten Zeitpunkten (Ticks) geschehen. Die Abfolge dieser Operationen nennt man *Skript.* Ein Skript kann entweder vom Benutzer direkt oder automatisch aus der internen Repräsentation der Intersynchronisation entwickelt werden.

Die Steuerung der Intersynchronisation ist somit zum einen bei der Anwendungsentwicklung durch die Anwendungsstruktur und die Definition der Uhrenoperationen eingebaut, zum anderen liegt sie in der Verwendung dieser Operationen an einer Steuerschnittstelle, die entweder interaktiv oder von einem Skript bedient wird.

4.5.3 Strategien zur Synchronisation

Die Grundstrategie der Synchronisation ist der Bezug des Präsentationstarts und der Dauer sowohl der Elemente als auch der Sequenzen auf einen gemeinsamen Ticker. Die Daten beschreiben selbst durch ihre Zeitparameter, in welchem Zeitraum sie präsentiert werden sollen. Somit ist keine weitere Abstimmung zwischen den einzelnen Präsentationsfunktionen erforderlich. Eine einzelne Präsentationsfunktion muß allerdings für jedes Element entscheiden, ob es dargestellt werden soll. Generell gilt die Regel, daß ein Datum nie vor seinem Startzeitpunkt präsentiert werden soll. Falls die Präsentation zum Startzeitpunkt noch nicht begonnen hat, darf sie solange noch nachträglich anfangen, wie seine Gültigkeitsdauer die

Präsentation verlangt. Im allgemeinen endet die Präsentation mit dem Ablauf der Dauer. Einzelne Präsentationsfunktionen können von dieser Regel zum Ende der Präsentation abweichen, falls das Folgedatum (noch) nicht dargestellt werden kann, oder definierte Lücken zu füllen sind, und der dargestellte Medientyp diese Abweichung ermöglicht (z. B. Video, Animation).

Zusätzliche Strategien zur Synchronisation werden erforderlich, falls mehrere Umgebungen innerhalb einer Anwendung zusammenarbeiten sollen. In diesem Fall müssen die einzelnen Ticker der verschiedenen Umgebungen miteinander abgestimmt werden. Der Grad der Abstimmung ist direkt von der Enge der Kopplung abhängig. Bei lose gekoppelten Umgebungen braucht nur einmal zu Beginn ein gemeinsamer Tickerstand gebildet werden. Diese Abstimmung kann nach beliebig langen Zeiten wiederholt werden. Bei enger Kopplung ist ein Austausch der Tickerstände eines Master–Tickers zu allen angebundenen Slave–Tickern in regelmäßigen Abständen nötig. Verfahren hierzu sind zum Beispiel in den verschiedenen Time–Code–Standards wie SMPTE Time Code (SMPTE = Society of Motion Picture and Television Engineers) [Burg93] und MIDI Time Code [MeBr89] vorgegeben. Eine dem MIDI Time Code entsprechende Verteilung der Ticks an verschiedene Umgebungen wird in [Hesme93] vorgestellt.

5 Umsetzung der Modelle auf Komponenten verteilter Systeme

In diesem Kapitel werden die Bausteine zur Integration der kontinuierlichen Medien als Komponenten verteilter Anwendungen beschrieben. Somit lassen sich die Bausteine in beliebige Anwendungen integrieren. Die Beziehungen zu den bisher erarbeiteten Ergebnissen sind folgende: Schnittstellen beschreiben die Funktionalität und bieten die zu realisierenden Operationen an. Komponenten entsprechen den Grund– oder den Verarbeitungsfunktionen, sie lassen sich auch als Geräte beschreiben, die in einer Anwendung eingesetzt werden. Diese Geräte können sowohl physikalische Geräte als auch reine Softwarerealisierungen sein. Die verwendete Spezifikationsmethode erlaubt eine generische Komponentenentwicklung, so daß nur Repräsentanten einer bestimmten Klasse von Komponenten spezifiziert werden. Komponenten für spezielle Geräte werden durch entsprechende Ableitung erzeugt. Nach dem selben Prinzip lassen sich ganze Anwendungen oder Anwendungsteile zur generischen Ableitung bereitstellen.

5.1 Komponenten in verteilten Anwendungen

In diesem Abschnitt wird kurz die Modellierung und Spezifikation verteilter Anwendungen vorgestellt, die in [Zimm92a] und [Zimm93] beschrieben wird. Verteilte Anwendungen bestehen aus einem Satz von interagierenden Komponenten, die einzeln für sich selbst, ohne Referenzen auf andere Komponenten definiert sind. Das Verhalten der Komponenten wird durch ihre Schnittstellen beschrieben.

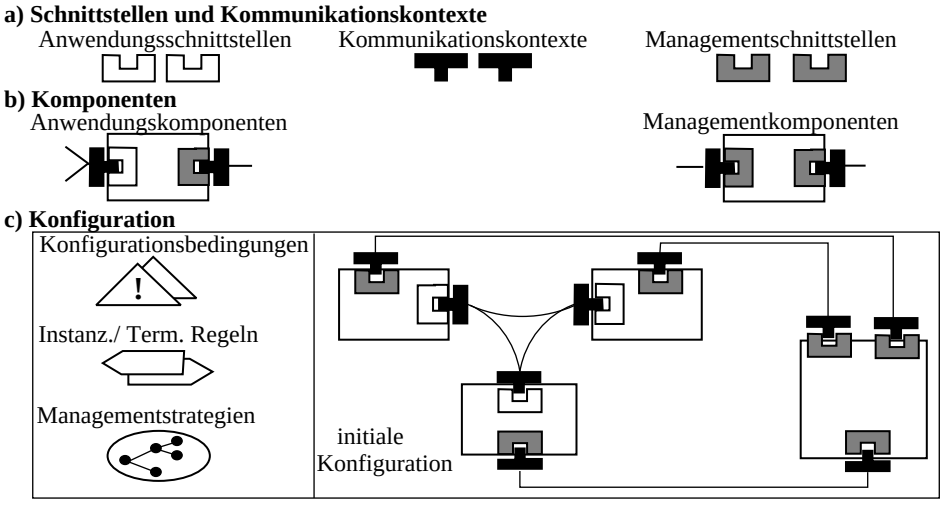


Abb. 26 Basisbausteine für die Konfiguration verteilter Anwendungen [Zimm93]

Die Beziehungen zwischen den Komponenten werden durch Bindungen zwischen den Schnittstellen hergestellt. Die Kommunikationsaspekte einer Schnittstelle werden im Kommunikationskontext berücksichtigt. Jede Schnittstelle muß einen Kommunikations-

kontext besitzen. Das Konzept der Kommunikationskontexte, seine Vorteile und Fähigkeiten, sind in [Feldh91] und [Feldh93] beschrieben. In Abbildung 26 sind diese Elemente und ihre Beziehungen dargestellt.

In [Zimm93] werden die Spezifikation der Schnittstellen, der Komponenten und der Anwendungskonfiguration voneinander getrennt und jeweils durch eine eigene Sprache unterstützt. Ein wichtiges Element der Spezifikationen von Schnittstellen und Komponenten sind die Reihenfolgebedingungen der Operationen. Diese werden mittels Pfadausdrücken festgelegt. So kann man festlegen, ob die Operationen nacheinander (p1 < p2), parallel (p1 || p2), sich gegenseitig ausschließend (p1; p2), null- oder mehrmals (p1°), mindestens einmal ($p1^{+}$) oder optional (p^{*}) durchgeführt werden oder im Rahmen einer Transaktion (T[p1]) ablaufen. Lange Pfadausdrücke werden lesbarer, wenn Teile durch Bezeichner ersetzt werden und die Bezeichner über die an ihrer Stelle einzusetzenden Teilausdrücke definiert werden [Alireza94].

set_attribute := volume; bass; treble; high; subsonic; muting
init < (set_attribute)° < release

Abb. 27 Beispiel eines Pfadausdrucks

Ein für die Spezifikationen sehr hilfreiches Mittel ist die auf allen Ebenen vorgesehene Verwendung von Schablonen. Durch Schablonen lassen sich die Spezifikationen gewissermaßen parametrisieren, indem für verschiedene Spezifikationen gleiche Teile aus der Schablone entnommen und nur an den in der Schablone gekennzeichneten Stellen erlaubte Ersetzungen vorgenommen werden.

5.1.1 Schnittstellenspezifikation

Die Schnittstellenbeschreibungssprache (Interface Definition Language, IDL) aus [Zimm93] basiert auf dem operationsorientierten Paradigma. Sie bildet die Grundlage zur Konstruktion verteilter Anwendungen. Der syntaktische Aufbau einer Schnittstellenbeschreibung gliedert sich in eine obligaten und einen optionalen Teil. Der Spezifikation der Operationen ist der obligate, das Kooperationsprotokoll der optionale Teil einer Schnittstellenbeschreibung. Eine Schnittstelle repräsentiert sich dem Benutzer nach außen als eine Menge von Operationen. So spezifiziert die IDL einen Schnittstellentyp kontextunabhängig von anderen Anwendungsbestandteilen, insbesondere unabhängig von Komponenten.

An jeder Schnittstelle können Operationen entweder vom Diensterbringer bereitgestellt oder vom Dienstbenutzer aufgerufen werden. Durch die Spezifikation ihrer Polarität können die Operationen beider Richtungen (**`CONSUMER INVOKES`** und **`SUPPLIER INVOKES`**) in einer gemeinsamen Schnittstellenspezifikation beschrieben werden. Eine Schnittstelle, die für den Diensterbringer und den Dienstbenutzer unterschiedliche Operationen spezifiziert, heißt asymetrische Schnittstelle. Benutzen mehr als zwei Komponenten die selbe Schnittstelle, kann es sinnvoll sein, diesen unterschiedliche Rollen zuzuordnen. Das Rollenschema beschreibt welche und wieviele Rollen es an einer Schnittstelle gibt. Zu einer Rolle wird beschrieben, welche Operationen von einer Komponente, die diese Rolle besitzt, verwendet werden dürfen.

Die Signatur einer Operation wird durch einen Operationsnamen und die zugehörigen Argumente in der abstrakten Typenbeschreibungssprache ASN.1 [ISO8824] festgelegt. Weitere Eigenschaften bezüglich der Ausführung der Operationen können durch Attribute festgelegt werden; zum Beispiel **CANCEL** für den möglichen vorzeitigen Abbruch von Funktionen oder **ASYNCHRONOUS** für die asynchrone Ausführung der Operation.

Die Struktur der Schnittstellenspezifikation wird in der folgenden Abbildung dargestellt. Die Instanzenbildung von Schnittstellen geschieht bei der Spezifikation von Komponenten, bei asynchronen Schnittstellen unter Berücksichtigung ihrer Polarität als Diensterbringer oder Dienstbenutzer [Zimm93].

```
<Name> INTERFACE
          CONSUMER INVOKES
              <Operationen>
          SUPPLIER INVOKES
              <Operationen>
          COOPERATION PROTOCOL
              <Pfadausdruck>
              <Rollenschema>
              <Attributierung der Operationen mit Rollen>
          <Signaturen>
```

Abb. 28 Struktur der Schnittstellenspezifikation

5.1.2 Kommunikationskontext

In diesem Abschnitt treten zum ersten Mal die Auswirkungen der Verteilung der Anwendungen deutlich hervor. Sind die Komponenten über verschiedene Rechner verteilt, so müssen die Interaktionen an den Schnittstellen über ein Kommunikationssystem ausgetauscht werden. Von der Kommunikation her gesehen befinden sich die Schnittstellen auf der Anwendungsschicht und stellen spezifisch eigene Kommunikationsanforderungen. Ein Kommunikationskontext dient dazu, die anwendungsorientierten Kommunikationsanforderungen der Schnittstellen zu spezifizieren [Feldh93]. Aus dieser Spezifikation kann dann automatisiert ein geeigneter Kommunikationsdienst konfiguriert werden.

Für asynchrone Schnittstellen teilt sich der Kommunikationskontext in ein korrespondierendes Kommunikationskontextpaar. Der eine Kontext beschreibt die Kommunikationsanforderungen der agierenden Seite, der andere die der reagierenden Seite. Die Wechselbeziehungen im Kommunikationskontextpaar werden dadurch ausgedrückt, daß die Definition des einen Kommunikationskontextes aus der des anderen hervorgeht, indem man nur die Eigenschaften, die das asymetrische Verhalten spezifizieren, mit den korrespondierenden Werten belegt und alle anderen Werte beibehält.

Der Kommunikationskontext legt folgende Eigenschaften fest [Feldh93]:

- Anforderungen an einen zu Grunde liegenden Transportdienst *TransportProperties;*
- Kooperation mit einem oder mehreren Partnern *CooperationStructure;*
- verbindungslose oder verbindungsorientierte Kommunikationsbeziehung *ConnectionManagement*;

- nachrichtenorientierter oder operationsorientierter Interaktionstyp zusammen mit weiteren Eigenschaften für die Interaktion und die Polarität der Kooperation *CooperationType*;
- ob und welche Art der Unterstützung verteilter Transaktionen existiert *TransactionType*.

Die Schlüsselworte, die für die Eigenschaften eingesetzt werden können brauchen hier nicht näher erläutert werden, da sie intuitiv verständlich sind. Genaue Beschreibungen sind in [Feldh93] zu finden.

5.1.3 Komponentenspezifikation

Die Spezifikation einer Komponente beinhaltet die Beschreibung der verwendeten Schnittstellen getrennt nach anwendungs– und management– bezogenen, deren Polarität als **CONSUMER** oder **SUPPLIER** bei asymetrischen Schnittstellen, sowie optional ein Kooperationsprotokoll auf Komponentenebene, das festlegt, wie Interaktionen über mehrere Schnittstellen hinweg abzuwickeln sind. Zur Beschreibung der Kommunikationsanforderungen an den Schnittstellen einer Komponente werden dieser ein oder mehrere Kommunikationskontexte zugeordnet. Zusätzlich kann eine Bindungsstrategie für eine Komponente angegeben werden, die festlegt, wann gebunden werden soll, welche Komponente oder Rolle dafür verantwortlich ist, und wieviele Bindungen an einer Schnittstelle erlaubt sind. Da die Komponentenspezifikation kontextunabhängig ist, also keine festen Bezüge zwischen Komponenten enthält, erfolgt die Spezifikation von Bindungen zwischen Komponenten erst bei der Anwendungskonfiguration. Die Struktur der Komponentenspezifikation wird in Abbildung 29 dargestellt.

Die Komponentenspezifikation erlaubt es mehrere Komponenten als Unterkomponenten in einer gemeinsamen Komponente zusammenzufassen. So können auch ganze Anwendungsteile separat spezifiziert werden. Unter der Verwendung von generischen Komponenten lassen sich damit generische Teilanwendungen spezifizieren.

```
<Name> COMPONENT

APPLICATION PROPERTIES

      INTERFACES
          CONSUMER AT
              <Schnittstellen_Name> : <Schnittstellentyp>;
                        :
              <Schnittstellen_Name> : <Schnittstellentyp>;
          SUPPLIER AT
              <Schnittstellen_Name> : <Schnittstellentyp>;
                        :
              <Schnittstellen_Name> : <Schnittstellentyp>;

      RESOURCES
          <Ressourcenliste>

      COOPERATION PROTOCOL
              <Kooperationsprotokoll>
```

```
MANAGEMENT PROPERTIES
      INTERFACES
          CONSUMER AT
              <Schnittstellen_Name> : <Schnittstellentyp>;
                        :
              <Schnittstellen_Name> : <Schnittstellentyp>;
          SUPPLIER AT
              <Schnittstellen_Name> : <Schnittstellentyp>;
                        :
              <Schnittstellen_Name> : <Schnittstellentyp>;
      COOPERATION PROTOCOL
              <Kooperationsprotokoll>
COMMUNICATION PROPERTIES
      <Kontext_Name> AT <Schnittstellen_Name> <Eigenschaften>
                :
      <Kontext_Name> AT <Schnittstellen_Name> <Eigenschaften>
```

Abb. 29 Struktur der Komponentenspezifikation

5.1.4 Anwendungskonfiguration

Die Konfigurationsbeschreibung besteht aus der Festlegung der initial vorhandenen Komponenten, der Spezifikation der Bindungen zwischen den Schnittstellen der Komponenten und optional der Formulierung von Konfigurationsbedingungen (**CONSTRAINTS**) für Existenz, Kardinalität und Plazierung [Zimm93] (vgl. Abb. 30). Die Konfigurationsbedingungen stellen die erlaubten Konfigurationen einer verteilten Anwendung dar; sie gliedern sich in folgende Gruppen:

- Existenzbedingungen beschreiben, welche Komponenten mit welchen Eigenschaften in der Anwendung vorhanden sein müssen;
- Kardinalitätsbedingungen schränken die Anzahl der Komponenten ein;
- Plazierungsbedingungen definieren Einschränkungen bezüglich der Zuordnung von Komponenten zu Rechnern, auf denen sie installiert werden.

```
<Name> DISTRIBUTED APPLICATION

COMPONENTS
       <Name> : <Komponententyp>;
                 :
       <Name> : <Komponententyp>;
BINDINGS
<Komponente>.<Schnittstelle> -- <Komponente>.<Schnittstelle>
                          :
<Komponente>.<Schnittstelle> -- <Komponente>.<Schnittstelle>
CONSTRAINTS
<Liste der Konfigurationsbedingungen>
PLACEMENT
       <Plazierungsbedingungen>
```

Abb. 30 Struktur der Anwendungskonfiguration

Komponenten können zu Komponentengruppen zusammengefaßt werden. Eine offene Komponentengruppe kann dynamisch ihre Größe ändern und Attribute besitzen, wie die initiale Anzahl der Gruppenmitglieder oder Regeln darüber, wann und von wem ein neues Mitglied der Komponentengruppe erzeugt werden darf.

5.2 Umsetzungsstrategie für kontinuierliche Medien

Ausgehend von den verschiedenen Grund– und Verarbeitungsfunktionen der kontinuierlichen Medien lassen sich Komponenten und ihre Schnittstellen im Sinne von [Zimm93] entwickeln, die funktionsorientierte Bausteine zur Integration kontinuierlicher Medien in verteilte Anwendungen darstellen und die generisch, objektorientiert erzeugt werden können. Die Entwicklung beruht auf der Idee, Grund– und Verarbeitungsfunktionen in Komponenten umzusetzen. Daraus ergibt sich, daß die Operationen auf den Medien die Operationen an den Schnittstellen der Komponenten festlegen. Eine Anwendung wird dann durch die Verbindung der Grundfunktionen analog zur physikalischen Verbindung von Geräten erstellt [Fritzsche96].

Dem konstruktiven Ansatz zur Anwendungsentwicklung von [Zimm93] folgend werden zunächst Schnittstellen definiert. Zur Modellierung von Datenfluß und Steuerungsstruktur werden drei Typen von Schnittstellen unterschieden. Ähnlich unterscheidet auch [Tenn94] "in-band" und "out-band" Kommunikation, ohne jedoch Operationen zu definieren. Steuerschnittstellen bieten die Operationen der Funktionen kontinuierlicher Medien an, Datenschnittstellen sorgen für die kontinuierliche Übertragung der Daten und Managementschnittstellen dienen der Überwachung und Steuerung der Komponenten über das Anwendungsmanagement. Managementoperationen werden zum Beispiel zur Erzeugung der initialen Konfiguration, zur Einführung neuer Komponenten in eine verteilte Anwendung, zum Entfernen einer Komponente aus dem System und zum Versetzen einer Komponente in einen neuen Zustand benötigt [Zimm93].

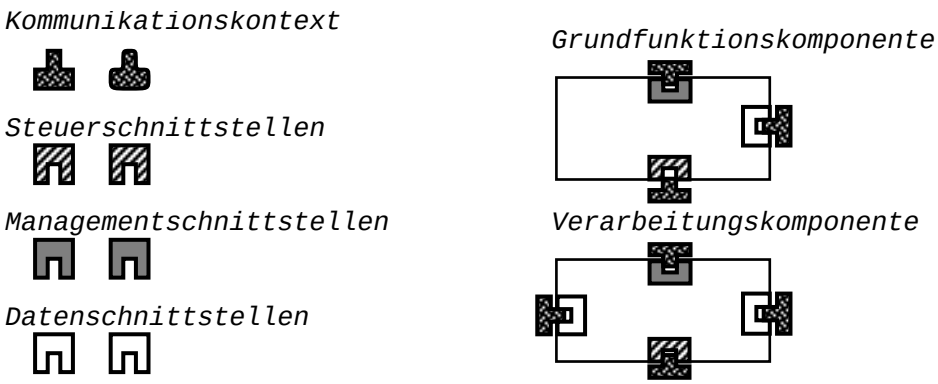


Abb. 31 Komponenten kontinuierlicher Medien und ihre Schnittstellen

Die Komponenten sind entweder Quellen oder Senken von Multimediadaten, falls sie Grundfunktionen darstellen, oder sowohl Quellen als auch Senken, falls sie Verarbeitungsfunktionen darstellen. Sie erfüllen als Grundfunktionskomponente genau eine Grundfunktion und besitzen eine Daten– eine Steuer– und eine Managementschnittstelle; als Verarbeitungs-

komponente erfüllen sie verschiedene Funktionen und besitzen mindestens zwei Daten– , eine Steuer– und eine Managementschnittstelle. Diese Beschreibung entspricht der Sichtweise auf Geräte im herkömmlichen analogen Medienbereich.

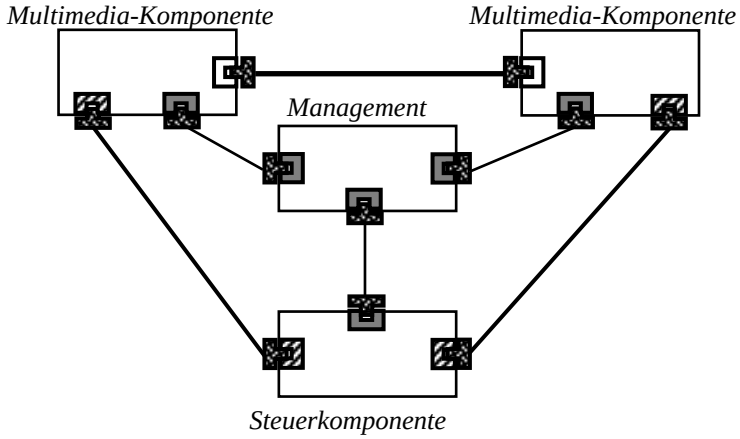


Abb. 32 Eine einfache Multimediaanwendung in Komponentendarstellung

Zusätzlich zu diesen Komponenten für Grund– und Verarbeitungsfunktionen werden Management- und Steuerkomponenten benötigt. Die Steuerkomponenten sind einerseits sogenannte interaktive Komponenten und enthalten die Benutzerschnittstelle der Multimediaanwendung, andererseits übernehmen sie die Koordination der Grund– und Verarbeitungsfunktionen durch Synchronisation der Operationen. Abhängig von der Anzahl der Benutzer oder der Anzahl der Steuerkomponenten ist die Steuerung der kontinuierlichen Medien zentral oder verteilt. Die Form der Zusammenarbeit mehrerer Benutzer und die dafür geltenden Regeln sowie die benötigten zusätzlichen Schnittstellen sollen hier nicht untersucht werden; sie sind Gegenstand der Gruppenarbeit, eines eigenen Forschungsgebiets.

Ein einfaches Beispiel für eine Multimediaanwendung ist in Abbildung 32 dargestellt. Aus Gründen der Übersichtlichkeit wird im folgenden auf die Darstellung der Managementkomponenten und der Managementschnittstellen verzichtet. Der in diesem Abschnitt vorgestellte konstruktive Ansatz wird schrittweise aufgebaut und verfeinert, indem zunächst die Schnittstellen, dann die Komponenten und zuletzt Anwendungsbeispiele vorgestellt werden. Dabei ist in jedem Schritt besonders der generische Charakter und die objektorientierte Verfeinerbarkeit und Ableitung von neuen Definitionen zu berücksichtigen.

5.3 Die Managementschnittstelle

Die Managementschnittstellen für die Komponenten kontinuierlicher Medien sowie für die Steuerkomponente bieten die Funktionalität für das Konfigurationsmanagement, das Bindungsmanagement sowie das Rollenmanagement. Insbesondere wird vom Management die Instanziierung und Bindung sowie die Terminierung der Komponenten übernommen, so daß diese Funktionalität in den weiteren Schnittstellen vorausgesetzt werden kann und nicht betrachtet wird. Managementfunktionalitäten werden in [Zimm93] ausführlich dargestellt, ebenso die Spezifikation der entsprechenden Schnittstellen. Weitere Betrachtungen zum Management von Multimediakomponenten sind in [Helbig94] zu finden.

An der Managementschnittstelle liegen ebenfalls Funktionen, die das Betriebssystem zur Verfügung stellt. Sie werden durch einfache Dateiverwaltung unterstützt. Zu den Funktionen zählt die Auswahl und das Belegen eines physikalischen Geräts bei Erzeugung einer Komponente in der Anwendung und das Laden der entsprechenden Funktionsimplementierungen und der Gerätetreiber.

Für Komponenten kontinuierlicher Medien kann einfach angenommen werden, daß die Komponente eine Initialisierung durchführen muß und danach sofort kontinuierlich arbeitet. Zur Initialisierung können der Komponente Parameter übergeben werden, aus welchen eine weitere Ressourcenbelegung, insbesondere für die Datenschnittstelle abgeleitet werden kann. Während der Initialisierung wird auch die Verbindung an der Datenschnittstelle aufgebaut. Die Komponente arbeitet kontinuierlich, bis sie eine Operation zur Aufgabe der Arbeit und der belegten Ressourcen (release) erhält. Die formale Definition der minimalen Managementschnittstelle für kontinuierliche Medien ist in Abbildung 33 wiedergegeben.

```
cont_media_management INTERFACE
        CONSUMER INVOKES
                init;
                release;
        COOPERATION PROTOCOL
                init < release;
```

Abb. 33 Spezifikation der allgemeinen Managementschnittstelle

Von dieser Schnittstelle lassen sich weitere Managementschnittstellen ableiten. Bei jeder abgeleiteten Schnittstelle werden die veränderten oder hinzugekommenen Operationen neu definiert. Das Kooperationsprotokoll muß dementsprechend neu spezifiziert werden, um die Reihenfolge der Operationsaufrufe festzulegen. Die Signatur der einzelnen Funktionen muß den jeweiligen Operationen angepaßt werden. So sind für die Initialisierung die verschiedenen Parameter der Funktionen auf kontinuierlichen Medien zu berücksichtigen.

5.4 Die Steuerschnittstelle

In diesem Abschnitt wird gezeigt, wie die Funktionen auf kontinuierlichen Medien in einer verteilten Anwendung zugänglich gemacht werden können. Die Steuerschnittstellen sind die Interaktionspunkte mit den Funktionen auf kontinuierlichen Medien. Sie lassen sich objektorientiert aus einer gemeinsamen Grundstruktur ableiten, indem sie nach ihren speziellen Eigenschaften verfeinert werden. Die im folgenden aufgezeigte Hierarchie stellt quasi nur ein Grundgerüst dar, das an allen Stellen weiter verfeinert werden kann. Zusätzlich können überall Querbeziehungen durch multiple Vererbung eingefügt werden. Dieses Hierarchie sollte aber nur dann verfeinert werden, wenn weitere medienspezifische Funktionen eines Geräts oder einer Software eingeführt werden. Die Wurzeln der Hierarchie bilden die allgemeinen Schnittstellen für Quellen und für Senken gemeinsam mit der Schnittstellen des Schrittgebers, sowie eine allgemeine Steuerschnittstelle für Verarbeitungsfunktionen. Somit existieren 4 Wurzeln, aus welchen durch multiple Vererbung die allgemeinen Schnittstellen für die Funktionen abgeleitet werden. Diese abgeleiteten Schnittstellen stehen dann zur weiteren Verfeinerung zur Verfügung.

5.4.1 Eine allgemeine Steuerschnittstelle

Die Steuerschnittstellen für Multimediageräte haben im allgemeinen Operationen wie *start, pause* und *stop*. Eine Steuerschnittstelle könnte wie in Abbildung 34 definiert werden. Sie wäre dann der Prototyp der Steuerschnittstellen aller Multimediageräte. Damit verliert man aber die bisher erarbeiteten Unterschiede in den Funktionen und den Bezug zum vorgestellten Datenmodell. Bei der folgenden Beschreibung der Funktionen werden die im vorigen Kapitel erarbeiteten Operationen verwendet, mittels derer die Operationen der Prototypschnittstelle realisiert werden. Diese Umsetzung und deren Koordination bei mehreren Komponenten ist eine Aufgabe der Steuerkomponente und wird im Abschnitt über diese näher behandelt.

```
prototype_control
INTERFACE
        CONSUMER INVOKES
        start;
        pause;
        stop;
        COOPERATION PROTOCOL
        (start <(pause < start*)° < stop)°;
```

Abb. 34 Spezifikation der Prototypsteuerschnittstelle für Grundfunktionen

Damit die Unterscheidung in Quellen und Senken auch an den Schnittstellen deutlich wird, werden die leeren Schnittstellen *any_source_control* und *any_sink_control* als Wurzeln der Ableitungsbäume eingeführt.

```
any_source_control
INTERFACE

any_sink_control
INTERFACE
```

Abb. 35 Ableitungswurzeln für die Schnittstellen von Quellen und Senken

5.4.2 Perzeption und Präsentation

Die Perzeptionsfunktion und die Präsentationsfunktion können im ersten Schritt gemeinsam behandelt werden. Wie aus dem vorigen Kapitel bekannt, realisieren sie die Operationen auf dem Schrittgeber. Zusätzlich zur Prototypschnittstelle für Grundfunktionen benötigen sie eine Operation zur Bestimmung der Zeit. Dazu wird hier die Operation *get_tick* verwendet, sie dient dazu, der Komponente den aktuellen Tickstand mitzuteilen. Diese Operation erfordert von der Kommunikationsverbindung an der Steuerschnittstelle einheitliche, bekannte Verzögerungen, also harte Zeitbedingungen. Die gemeinsame Steuerschnittstelle dieser Funktionen wird in Abbildung 36 definiert.

In den weiteren Operationen unterscheiden sich die Perzeption und die Präsentation. Daher wird unter Verwendung des Ableitungsmechanismus zunächst jeweils eine weitere Schnittstelle spezifiziert. Die Abbildungen 37 und 39 enthalten die Spezifikationen mit den zugehörigen Kooperationsprotokollen. Für die Perzeption kann eine weitere Verfeinerung angegeben werden, die sich auf eine der verschiedenen Erzeugungsstrategien für Elemente

bezieht. In allen Spezifikationen wurde der Übersicht halber auf die Angabe der Signaturen der Operationen verzichtet, da diese aus den Operationsdefinitionen im vorigen Kapitel abgeleitet werden können.

```
step_control
INTERFACE
        CONSUMER INVOKES
        set_rate;
        get_rate;
        set_start_tick;
        get_start_tick;
        get_tick;

        COOPERATION PROTOCOL
        set_parameters = (set_rate; set_start_tick)
        get_parameters = (get_rate; get_start_tick)
        get_tick° |(set_parameters; get_parameters)°
```

Abb. 36 Spezifikation der gemeinsamen Steuerschnittstelle für Perzeption und Präsentation

Das Kooperationsprotokoll an der Steuerschnittstelle *step_control* legt lediglich fest, daß alle Operationen mehrmals aufgerufen werden können, und daß *get_tick* parallel zu allen anderen Operationen ausgeführt werden kann.

```
perception_control
INTERFACE DERIVED FROM step_control, any_source_control
        CONSUMER INVOKES
        send_sequence_parameters;

        COOPERATION PROTOCOL
        set_parameters = (set_rate; set_start_tick)
        get_parameters = (get_rate; get_start_tick)
        get_tick° |
        ((send_sequence_parameters+;
         (set_parameters; get_parameters))°)
```

Abb. 37 Spezifikation einer allgemeinen Steuerschnittstelle für die Perzeption

Die Steuerschnittstelle der Perzeption unterscheidet sich von *step_control* nur dadurch, daß die Aufforderung zum Senden der aktuellen Sequenzparameter aufgerufen werden kann. Eine weitere Verfeinerung ist in der folgenden Spezifikation enthalten.

```
perception_control_multistep
INTERFACE DERIVED FROM perception_control
       CONSUMER INVOKES
       set_data_steps;   get_data_steps;
       set_duration;     get_duration;

       COOPERATION PROTOCOL
       set_parameters =
       (set_data_steps; set_duration; set_rate; set_start_tick)
       get_parameters =
       (get_data_steps; get_duration; get_rate; get_start_tick)
       get_tick° |
       ((send_sequence_parameters+;
        (set_parameters; get_parameters))°)
```

Abb. 38 Erweiterung der Steuerschnittstelle für Element–Mehrschritt–Strategie

An dieser Schnittstellenspezifikation fällt auf, daß die neuen Funktionen das eigentliche Kooperationsprotokoll nicht mehr verändern, sondern nur noch in den einzusetzenden Variablen auftreten. Weitere Verfeinerungen haben die selbe Eigenschaft.

Die einfachste Präsentationsschnittstelle bietet zusätzlich zur Schritteinstellung die Abfrage der Sequenzparameter der aktuell präsentierten Sequenz.

```
presentation_control
INTERFACE DERIVED FROM step_control, any_sink_control
       CONSUMER INVOKES
       get_seq_s_rate;
       get_seq_t_rate;
       get_seq_start_step;
       get_seq_duration;

       COOPERATION PROTOCOL
       set_parameters = (set_rate; set_start_tick)
       get_parameters = (get_rate; get_start_tick)
       get_seq_parameters =
       (get_seq_s_rate; get_seq_t_rate;
        get_seq_start_step; get_seq_duration)

       get_tick° |
       (set_parameters; get_parameters; get_seq_parameters)°
```

Abb. 39 Spezifikation der allgemeinen Schnittstelle für die Präsentation

Weitere Verfeinerung können darin bestehen, daß man abhängig vom bezogenen menschlichen Sinn spezielle Geräte zur Perzeption oder Präsentation untersucht und deren Einstellmöglichkeiten durch entsprechende Parameter beschreibt. Zum Beispiel kann eine Kamera durch einen Parameter Blendeneinstellung beschrieben werden, der außer einer Blendenzahl auch die Eigenschaften manuell oder automatisch hat. So ist es möglich, die Schnittstellen-

hierarchie weiter zu verfeinern, falls Geräte integriert werden sollen, die zusätzliche Funktionen bereitstellen. Die Geräte können dann wahlweise über ihre Elternschnittstellen, wie Standardgeräte angesprochen werden, oder über die verfeinerten Schnittstellen als Spezialgerät. Weitere Steuerschnittstellen für die Perzeption und die Präsentation sind in [Konst93] zu finden.

5.4.3 Speicher

Die Speicherfunktion wurde in eine lesende und eine schreibende aufgeteilt. Dementsprechend werden zwei Schnittstellen spezifiziert, die die lesenden und schreibenden Operationen trennen. Die Operationen ***concat*** und ***extract*** sind besonders hervorzuheben, da sie (mindestens) eine Sequenz als Folge von Elementen als Eingabe erwarten oder als Ausgabe produzieren. Die Operation ***concat(a, b)*** wird dem schreibenden Speicher in der Form zugerechnet, daß ***a*** eine Sequenz im Speicher bezeichnet und ***b*** als Sequenz an der Datenschnittstelle übertragen wird; nach der Ausführung von ***concat (a, b)*** kennzeichnet ***a*** die Stelle des Ergebnisses der Operation im Speicher. Die Operation ***extract(a, p1, p2)*** wird dem lesenden Speicher zugeordnet in der Form, daß ***a*** eine Sequenz im Speicher bezeichnet und das Ergebnis der Operation an der Datenschnittstelle ausgegeben wird.

Mittels der Funktionen ***create, concat, extract*** und ***destroy*** lassen sich einfache Speicherschnittstellen realisieren. Der Sonderfall von ***extract***, daß die gesamte Sequenz ausgelesen wird, soll durch eine eigene Funktion ***send_seq*** beschrieben werden. Damit stehen zwei einfache Speicherschnittstellen zur Verfügung, die für einen sehr großen Teil aller Anwendungen ausreichend sind, und speziell den kontinuierlichen Charakter der Sequenzen unterstreichen. In Abbildung 40 werden die beiden zugehörigen Spezifikationen angegeben.

```
simple_write_control
INTERFACE DERIVED FROM any_sink_control
       CONSUMER INVOKES
       create;
       concat;
       destroy;

simple_read_control
INTERFACE DERIVED FROM any_source_control
       CONSUMER INVOKES
       extract;
       send_seq;
```

Abb. 40 einfache Schreib– und einfache Lese–Steuerschnittstelle

Aus diesen beiden Schnittstellen kann ein einfacher Schreib/Lese–Speicher für kontinuierliche Medien hergeleitet werden. Die folgende Abbildung enthält die entsprechende Schnittstellenspezifikation.

```
simple_read_write_control
INTERFACE DERIVED FROM simple_write_control;simple_read_control
       COOPERATION PROTOCOL
       (create;concat)+ < (extract; send_seq)+
```

Abb. 41 einfache Schreib/Lese–Steuerschnittstelle

Sowohl die Steuerschnittstelle des schreibenden als auch die des lesenden Speichers können gemäß des Datenmodells aus dem vorangehenden Kapitel erweitert werden. Dabei stellt die Erweiterung des lesenden Speichers weniger Anforderungen an ihre Realisierung, als die des schreibenden an seine, da letzterer bei manchen Operationen eine große Anzahl von Änderungen durchführen muß. Die Parameter der neuen Operationen werden alle an den Steuerschnittstellen übergeben. Die folgenden Spezifikationen geben die Sequenzebene des Speichers wieder. Es wird nochmals darauf hingewiesen, daß nur die Operationen auf der Sequenz und ihren Elementen, nicht aber Such– oder Auswahl–Operationen von Sequenzen betrachtet werden. Weitere Verfeinerungen der Steuerschnittstellen der Speicherfunktionen werden in [Konst93] entwickelt.

```
seq_write_control
INTERFACE DERIVED FROM simple_write_control
        CONSUMER INVOKES
        set_S_rate;
        set_T_rate;
        set_start_step;
        set_duration;
        add;
        del_elem;

        set_el_value;
        set_el_start;
        set_el_duration;
        COOPERATION PROTOCOL

seq_read_control
INTERFACE DERIVED FROM simple_read_control
        CONSUMER INVOKES
        get_S_rate;
        get_T_rate;
        get_start_step;
        get_duration;

        first;
        next;
        prev;
        set_pos;
        COOPERATION PROTOCOL
```

Abb. 42 Schreib– und Lese–Steuerschnittstelle für Sequenzen

5.4.4 Verarbeitungsfunktionen

Die Verarbeitungsfunktionen werden auf alle Elemente einer Sequenz angewandt. Dabei wird eine Sequenz an der einen Datenschnittstelle erwartet, ihre Elemente werden nach dem Pipeline-Prinzip nacheinander verändert und die neue Sequenz wird elementweise an einer anderen Datenschnittstelle ausgegeben. Beinhaltet die Verarbeitungsfunktion ein Mischen oder Verteilen, so werden entsprechend mehrere Datenschnittstellen als Eingang beziehungsweise als Ausgang benötigt, oder es müssen Mehrfachbindungen an die Schnittstelle erstellt werden.

Die Steuerung dieser Funktionen ist daher abhängig von den speziellen Aufgaben der Funktion und beinhaltet im Wesentlichen das Setzen von Attributen, die die Berechnung der neuen Elementwerte bestimmen. Eine allgemeine Steuerschnittstelle für Verarbeitungsfunktionen wird wie in Abbildung 43 spezifiziert.

```
control_f
INTERFACE
        CONSUMER INVOKES
        set_parameter;
```

Abb. 43 Beispiel einer Steuerschnittstelle für Verarbeitungsfunktionen

Eine Verarbeitungsfunktion kann aber auch eine eigenständige, interaktive Komponente sein und benötigt dann keine Steuerschnittstelle. In diesem Fall werden ihre Parameter über die interaktive Benutzerkommunikation ermittelt. Ebenfalls keine explizite Steuerschnittstelle benötigen Verarbeitungsfunktionen, die nur eine Umkodierung der Daten vornehmen. Eine ausführliche Beschreibung von Steuerschnittstellen für Verarbeitungsfunktionen und die Spezifikation der zugehörigen Komponenten findet sich in [Alireza94].

5.5 Die Datenschnittstelle

An der Datenschnittstelle werden Sequenzen kontinuierlich übertragen. Sie wird also ausschließlich für den Transport der Multimediadaten verwendet. Dafür ist eine nachrichtenorientierte Schnittstelle geeignet. Eine solche Schnittstelle bietet lediglich Operationen zum Verschicken (*send*) und zum Empfangen (*receive*) an [Zimm93] [Alireza94]. Es ist hier sogar ausreichend, wenn die Datenschnittstelle dem Konsumenten lediglich den Operationsaufruf *receive* anbietet. Diese Operation verlangt aber, daß vor ihrem Aufruf die Sequenz bereits vollständig vorliegen muß. Von den beiden vorhandenen Quellen für Sequenzen, welche sind die Perzeption und der lesende Speicher, kann das aber nur der Speicher gewährleisten. Es ist daher naheliegend, die Operation *receive* aufzuspalten in *receive_seq_parameters* und *receive_seq_element*; und die kontinuierliche Übertragung der Sequenz durch den kontinuierlichen Aufruf von *receive_seq_element* zu realisieren.

Die kontinuierliche Übertragung der Sequenzelemente stellt spezielle Ansprüche an das Kommunikationssystem und den Datentransport. Diese Dienstgüteanforderungen sind keineswegs von vornherein fest, sondern sie hängen vom Elementtyp der Sequenz, der minimalen Dauer eines Elements sowie den vom Benutzer an die Anwendung gestellten Qualitätsanforderungen ab. Daher sollten die Dienstgüten der Datenschnittstelle auf Anwendungsebene verhandelt werden. Dieser Forderung genügt zur Zeit kein existierendes Kommunikationssystem. Sie kann auch nicht allgemein für beliebige Schnittstellen und Kommunikationsdienste realisiert werden. Für einen Kommunikationsdienst für kontinuierliche Medien ist es aber möglich, dieses durchzuführen. Dazu wird im einfachsten Fall die Operation *setup* eingeführt, die als Parameter die geforderten und als Ergebnis die erreichten Dienstgütewerte hat. Die Operation *setup* muß bei der Initialisierung einer Quellenkomponente von dieser bei der Senkenkomponente aufgerufen werden. Wie bei nicht ausreichenden Dienstgütewerten verfahren wird, bleibt der Anwendung überlassen. Die allgemeine Datenschnittstelle wird somit wie in Abbildung 44 spezifiziert.

```
continuous_data
INTERFACE
        CONSUMER INVOKES
        setup;
        receive_seq_parameters;
        receive_seq_element;

        COOPERATION PROTOCOL
        (setup+ < receive_seq_parameters+ < receive_seq_elemet°)°
```

Abb. 44 Spezifikation der allgemeinen Datenschnittstelle

Die Datenschnittstelle läßt sich nach verschiedenen Gesichtspunkten verfeinern. Zum einen sind es die verschiedenen Typen der Sequenzelemente, die eine Unterscheidung der Schnittstellen zunächst nach den Medien, wie Audio oder Video, und dann nach ihrer Kodierung, wie CD–Sektor, μ–law, MPEG, JPEG, DVI, nahelegen. Zum anderen können an den Datenschnittstellen verschiedene Normalformen der Sequenzen gefordert werden. Diese Verfeinerungen unterscheiden sich von der allgemeinen Datenschnittstelle zunächst nur durch eine andere Signatur beziehungsweise eine andere Implementierung der Operationen. Daher würde eine explizite Spezifikation der verfeinerten Schnittstellen hier zu weit führen.

Eine andere Erweiterung der Datenschnittstelle soll hier noch kurz angesprochen werden. Insbesondere bei Speicherschnittstellen kann es sinnvoll sein, die Sequenzelemente nicht mit einer festen Rate, sondern abhängig von der Pufferfüllung des Empfängers zu senden. Dabei kann sowohl die Anzahl der Elemente, als auch die Schrittzahl des Schrittgebers als Steuergröße verwendet werden. Bei der Verwendung der Elementanzahl ergibt sich eine Variante des Sliding–Window-Protokolls. Analog dazu kann durch die Schrittzahl der größte Startzeitpunkt berechnet werden, bis zu dem im Voraus gesendet werden darf.

5.6 Kommunikationsanforderungen

Für Multimediasysteme werden in der Literatur verschiedene Kommunikationsdienste beschrieben. Durch die entwickelten Bausteine zur Integration kontinuierlicher Medien in verteilte Anwendungen und die vorgestellte Aufgabenteilung von Steuer– und Datenschnittstelle, ergeben sich klarere, zum Teil auch einfachere Anforderungen. Ausgehend von den Anforderungen der Anwendung ist zu unterscheiden, welche Aufgaben die Anwendung erfüllen muß und welche das Kommunikationssystem übernehmen kann. Grundsätzlich entscheidet die Anwendung darüber, welche Leistungen zu erbringen sind, eventuell nach dreiseitiger Übereinkunft. Das Kommunikationssystem entscheidet darüber, wie die Leistungen zu erbringen sind.

Kommunikationsanforderungen der Managementschnittstellen sind in [Zimm93] ausreichend beschrieben und werden hier nicht näher behandelt. Die Kommunikationsanforderungen an der Steuerschnittstelle unterscheiden sich von denen an der Datenschnittstelle. Daher werden die beiden Schnittstellen getrennt betrachtet. Gemeinsam ist ihnen, daß ein großer Teil der Anforderungen allgemeingültig für alle Steuer– beziehungsweise alle Datenschnittstellen ist. Zentrale Punkte für die Auffächerung der Steuer– und der Daten– Schnittstellen sind die Transport–Dienstgüteparameter, Durchsatz, maximal zulässige Übertragungsverzögerung, zulässige Varianz der Verzögerung und akzeptierbare Fehlerwahrscheinlichkeit.

Diese Anforderungen sind von den Datentypen und Codierungen der an den Schnittstellen ausgetauschten Information abhängig. Die Kommunikationsanforderungen werden daher durch Schablonen für Kommunikationskontexte beschrieben, die nur noch in den Transporteigenschaften vervollständigt werden müssen. Die entsprechenden Stellen sind in den Schablonen durch spitze Klammern und kursive Schrift gekennzeichnet. Die wenigen Parameter der Schablonen können für konkrete Schnittstellen jeweils leicht bestimmt werden, so daß zu einer konkreten Bibliothek von Schnittstellen eine konkrete Bibliothek von passenden Kommunikationskontexten erstellt werden kann. Es kann zu einer Schnittstelle eine Reihe von passenden Kommunikationskontexten geben, wie auch ein Kontext zu verschiedenen Schnittstellen passen kann.

Da die Kooperation der Komponenten kontinuierlicher Medien asymmetrisch ist, werden für die Partner an den Schnittstellen Kommunikationskontextpaare definiert; jeweils ein Kontext für den Consumer und den Supplier an der Schnittstelle. An der Steuerschnittstelle ist stets die Funktion des kontinuierlichen Medii der Anbieter, die Steuerkomponente der Benutzer. An der Datenschnittstelle ist die Quelle der Benutzer und die Senke der Anbieter.

5.6.1 Anforderungen der Steuerschnittstelle

Für die Steuerschnittstelle wurden bereits verschiedene Schnittstellenvarianten vorgestellt. Einige davon können die gleichen verwenden, so daß nicht für alle Schnittstellen ein eigenes Kommunikationskontextpaar definiert werden muß. Allerdings unterscheiden sich die Kommunikationsanforderungen der Schnittstellen so weit, daß es nicht sinnvoll ist, nur ein Kommunikationskontextpaar zu verwenden. Zunächst werden die gemeinsamen Anforderungen beschrieben und am Ende dieses Abschnitts werden dann die unterschiedlichen Transportanforderungen behandelt.

Die Steuerschnittstelle ist operationsorientiert durch die von den einzelnen Funktionen realisierten Operationen auf den Medien. Sie ist eine klassische asynchrone Klient–Bediener–Schnittstelle. Daher wird als Interaktionsanforderungen ein Klient–Bediener–Modell mit Operationen definiert. Die Operationen werden asynchron initiiert und liefern ein Operationsergebnis oder eine Fehlermeldung. Die Operationen sollen im Fehlerfall implizit wiederholt werden und es soll die Möglichkeit bestehen, nicht abgeschlossene Operationen abzubrechen. Die Operationen sind voneinander unabhängig und benötigen keine Verschlüsselung. Im Kommunikationskontext werden unter `INRTERACTION` genau diese Anforderungen durch die entsprechenden Schlüsselworte spezifiziert.

Die Kommunikation an der Steuerschnittstelle ist verbindungsorientiert; die Verbindung wird vom Benutzer der Funktion her auf– und auch wieder abgebaut. Im Falle eines Abbruchs soll die Verbindung automatisch wieder aufgebaut werden. Eine Steuerschnittstelle soll hier nur einem Benutzer zugeordnet werden. Falls eine Kooperation mehrerer Benutzer an einer Steuerschnittstelle tatsächlich gewünscht wird, muß diese erweitert werden und einen entsprechenden Kommunikationskontext erhalten.

Eine wesentliche Unterscheidung für verschiedene Steuerschnittstellen ist die zwischen enger und loser Kopplung. Bei loser Kopplung reichen im allgemeinen unbestimmte Transportanforderungen aus, so daß ein Kommunikationskontext wie in Abbildung 45 spezifiziert werden kann.

```
   ContMedFunctionControlContext
 COMMUNICATIONCONTEXT
  TRANSPORT(ARBITRARY)
  SINGLE-PARTY
  CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
        ESTABLISH-RESPONDER
        RELEASE-RESPONDER
        REESTABLISHING))
  INTERACTION (SERVER
    OPERATIONS(
            ASYNCHRONOUS
            REPLY
            RETRANSMITTING
            CANCEL
            NO_LINKED_OPERATION
            NO_ENCRYPTION
            NO_TOKENSUPPORT))
::=ContMedFunctionControlContext
```

```
   UserControlContext
 COMMUNICATIONCONTEXT
  TRANSPORT (ARBITRARY)
  SINGLE-PARTY
  CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
        ESTABLISH-INITIATOR
        RELEASE-INITIATOR
        REESTABLISHING))
  INTERACTION (CLIENT
        OPERATIONS(
            ASYNCHRONOUS
            REPLY
            RETRANSMITTING
            CANCEL
            NO_LINKED_OPERATION
            NO_ENCRYPTION
            NO_TOKENSUPPORT))
::=UserControlContext
```

Abb. 45 Kommunikationskontextpaar für Steuerschnittstellen in loser Kopplung

Um eine erfolgreiche Zusammenarbeit verschiedener Komponenten sicherzustellen, ist es allerdings besser die Transportanforderungen gemäß der folgenden Überlegungen zu spezifizieren. Insbesondere für die enge Kopplung von Steuerschnittstellen müssen konkrete Anforderungen an den Transportdienst gestellt werden. Dabei lassen sich aber nur durch Tickrate und Schrittrate sowie Codierungen parametrisierte Berechnungsgrundlagen und Grenzwerte finden. Es gelten die folgenden Überlegungen:

- Durchsatz:
  Der Durchsatz an der Steuerschnittstelle bestimmt vor allem, wie kurz hintereinander Operationen bei der Funktion kontinuierlicher Medien aufgerufen werden können. Die Maximalforderung für die enge Kopplung ergibt sich aus der möglichen Parallelität der *get_tick* Operation zu den anderen Operationen. Es muß also mindestens einen Tickwert pro Tickzeit plus einen Betrag für die übrigen Operationen zur Verfügung stehen. Vom Benutzer ist dabei festzulegen, in welcher Zeit diese Operationen mit ihren Parametern übertragen werden. Trotzdem bleibt die Datenrate gering, da die Werte nur wenige Bytes benötigen.
  Für Speicherschnittstellen ist der wesentliche Faktor für den Durchsatz das Datenvolumen der Sequenzelemente, denn dort treten diese als Parameter auf. Sequenzelemente als Parameter haben beispielsweise die Operationen *add* und *set_elem*; aber auch die Operation *set_el_value* benötigt eine ähnliche Datenmenge. Darüber hinaus ist ein Zuschlag für die Codierung der Operationen selbst sowie für weitere Parameter zu berücksichtigen.
  Die folgenden Beispiele zeigen typische Berechnungen:

  - eng gekoppelte Perzeption oder Präsentation:
    600 HZ Ticker mit 4 Byte Integer ⇒ 19200 bit/s
    dazu
    werden für die Operationen, da nur Ganzzahlparameter vorkommen, 20 Byte als ausreichend angesehen, die in einer Zwanzigstelsekunde übertragen sein müssen, damit „kurze" Abstände, in etwa der halben menschlichen Reaktionszeit, zwi-

schen zwei Operationen eingehalten werden können. Daraus ergibt sich eine Anforderung von 3200 bit/s.
Insgesamt sollte der Durchsatz also 22400 bit/s (23 kbit/s) sein.

- lesender oder schreibender Speicher für einfache, unkomprimierte Videobilder in besten Farben:
  Es gibt keinen Ticker oder Schrittgeber, aber Sequenzelemente als Parameter:
  1 Farbbild mit 384×280 Punkten und je 3 Byte RGB-Information
  in einer Sekunde ⇒ 2580480 bit/s.
  Es ergib sich eine Anforderung von 2,46 Mbit/s.
- lesender oder schreibender Speicher für Audio mit Telephonqualität:
  bei einer Übertragung in etwa der Zeit, in der Audio zu hören wäre, ergibt sich eine Anforderung von 64 Kbit/s.

- maximal zulässige Übertragungsverzögerung:
  Die Verzögerung bestimmt, wie schnell eine Operation beim Partner ausgelöst werden kann. Von der Anwendung der Medien her ist bekannt, daß eine Verzögerung um einen Schritt ein guter Wert ist. Damit ist der Wert wieder direkt anwendungsabhängig, kann aber durch 1/30 Sekunde für Video und 1/75 Sekunde für CD-Sektoren abgeschätzt werden.
  Eine Besonderheit dieser Anforderung ist, daß weniger ihr absoluter Wert, als vielmehr die Unterschiede der tatsächlichen Übertragungsverzögerung der verschiedenen Steuerschnittstellen einer Anwendung entscheidend sind. Welche genauen Anforderungen zu stellen sind, kann erst durch eine breite Anwendung des entwickelten Modells festgestellt werden. Da die maximale Übertragungsverzögerung an der Steuerschnittstelle aber mit der Reaktionszeit verglichen werden kann, sollte ein Wert von 1/10 Sekunde, der der menschlichen Reaktionszeit entspricht, noch ausreichend sein.

- zulässige Varianz der maximal zulässigen Übertragungsverzögerung:
  Die Varianz der Verzögerung ist vor allem bei den Steuerschnittstellen der Perzeption und der Präsentation ein wichtiger Faktor, besonders, wenn diese in enger Kopplung stehen. Während bei loser Kopplung eine Varianz von einem Schritt durchaus vertretbar ist, soll bei enger Kopplung jeder Tick mit genau der gleichen Verzögerung übertragen werden. Diese Forderung heißt isochrone Übertragung. Eine echte Isochronität der Ticks wird allerdings nie erreichbar sein. Daher lautet die Frage, wie nahe muß man der Isochronität kommen?
  Genaue Antworten lassen sich auch hier erst aus einer längeren Erfahrung mit Anwendungen des entwickelten Modells ableiten. Als grobe Abschätzung soll hier der vermutlich große Wert von einer halben Tickzeit gelten. Bei einem 600 HZ Ticker sind das 1/1200 Sekunden ≈ 0,8 ms.

- Die akzeptierbare Fehlerwahrscheinlichkeit:
  An der Steuerschnittstelle kann weder für eine Operation noch für einen Parameter ein Fehler akzeptiert werden. Die einzige akzeptierbare Fehlerwahrscheinlichkeit ist daher 0. Der Transportdienst muß zuverlässig sein.

Das in Abbildung 46 spezifizierte Kontextpaar definiert die oben beschriebenen genaueren Anforderungen, die einer engen Kopplung genügen.

```
TightMedFunctionControlContext Template
COMMUNICATIONCONTEXT
    TRANSPORT(
        DOUPLEX,
        THROUGHPUT: <23000> Bit/sec,
        TRANSMITDELAY: <100> msec,
        DELAYJITTER: <0.8> msec
        TRANSFERFAILUREPROB: <0> %
        )
   SINGLE-PARTY
  CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
        ESTABLISH-RESPONDER
        RELEASE-RESPONDER
        REESTABLISHING)
        )
    INTERACTION (SERVER
    OPERATIONS(
              ASYNCHRONOUS
              REPLY
              RETRANSMITTING
              CANCEL
              NO_LINKED_OPERATION
              NO_ENCRYPTION
              NO_TOKENSUPPORT)
              )
::=TightMedFunctionControlContext
```

```
TightUserControlContextTemplate
COMMUNICATIONCONTEXT
    TRANSPORT (
        DOUPLEX,
        THROUGHPUT: <23000> Bit/sec,
        TRANSMITDELAY: <100> msec,
        DELAYJITTER: <0.8> msec
        TRANSFERFAILUREPROB: <0> %
        )
   SINGLE-PARTY
  CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
         ESTABLISH-INITIATOR
         RELEASE-INITIATOR
        REESTABLISHING)
         )
    INTERACTION (CLIENT
         OPERATIONS(
              ASYNCHRONOUS
              REPLY
              RETRANSMITTING
              CANCEL
              NO_LINKED_OPERATION
              NO_ENCRYPTION
              NO_TOKENSUPPORT)
              )
::=TightUserControlContext
```

Abb. 46 Kommunikationskontextpaar für Steuerschnittstellen in enger Kopplung

```
StoreFunctionControlContextTemplate
COMMUNICATIONCONTEXT
    TRANSPORT(
        DOUPLEX,
        THROUGHPUT: <64000> Bit/sec,
        TRANSMITDELAY: <100> msec,
        DELAYJITTER: <10> msec
        TRANSFERFAILUREPROB: <0> %
        )
   SINGLE-PARTY
   CONNECTION-MANAGEMENT(
     CONNECTIONORIENTED(
        ESTABLISH-RESPONDER
        RELEASE-RESPONDER
        REESTABLISHING)
        )
   INTERACTION (SERVER
     OPERATIONS(
              ASYNCHRONOUS
              REPLY
              RETRANSMITTING
              CANCEL
              NO_LINKED_OPERATION
              NO_ENCRYPTION
              NO_TOKENSUPPORT)
              )
::=StoreMedFunctionControlContext
```

```
StoreUserControlContextTemplate
COMMUNICATIONCONTEXT
    TRANSPORT (
        DOUPLEX,
        THROUGHPUT: <64000> Bit/sec,
        TRANSMITDELAY: <100> msec,
        DELAYJITTER: <10> msec
        TRANSFERFAILUREPROB: <0> %
        )
 SINGLE-PARTY
 CONNECTION-MANAGEMENT(
   CONNECTIONORIENTED(
         ESTABLISH-INITIATOR
         RELEASE-INITIATOR
        REESTABLISHING)
         )
 INTERACTION (CLIENT
         OPERATIONS(
              ASYNCHRONOUS
              REPLY
              RETRANSMITTING
              CANCEL
              NO_LINKED_OPERATION
              NO_ENCRYPTION
              NO_TOKENSUPPORT)
              )
::=StoreUserControlContext
```

Abb. 47 Kommunikationskontextpaar für Speicher–Steuerschnittstellen

Das Kommunikationskontextpaar für Steuerschnittstellen der Speicherfunktionen wird in Abbildung 47 spezifiziert. Dabei geht man davon aus, daß eine Varianz der Verzögerung von 10% zulässig ist. Verfeinerungen durch genauere Anforderungen können bei der Entwicklung verteilter Anwendungen jederzeit hinzugefügt werden.

5.6.2 Anforderungen der Datenschnittstelle

Die Datenschnittstelle ist lediglich eine Schnittstelle zum Transport der Sequenzen. Der Transport kontinuierlicher Medien ist ein reiches Forschungsgebiet, auf dem eine Fülle von verschiedenen Vorschlägen und zum Teil prototypischen Implementierungen existieren, die sich auf die verschiedensten Schichten des ISO OSI-Referenz–Modells beziehen. Durch das neu entwickelte Datenmodell und die festgelegte Datenschnittstelle werden die Anforderungen jedoch vereinfacht und es läßt sich eine Schablone für die Kommunikationskontexte entwickeln, die nur noch in speziellen Transportanforderungen auszufüllen ist. Hier spielen die Elementdauern eine entscheidende Rolle. Da ein Element minimal einen Schritt dauert, muß es in maximal dieser Zeit übertragen werden, da sich sonst ein Stau bei der Quelle beziehungsweise ein Loch bei der Senke ergeben kann. Allerdings ist die Datenmenge eines Elements weder generell gleich (Audio vs. Video) noch für alle Elemente einer Sequenz einheitlich (MPEG-GOPs). Ein wichtiger Punkt ist aber der, daß bei diesen Schnittstellen im allgemeinen keine Isochronität verlangt wird. Eine tiefgreifende Analyse der Dienstgüteparameter und eventuelle Erweiterungen werden in [CaCoGa95] vorgestellt. Einen Überblick über bestehende Architekturen zur Dienstgütesicherung von Anwendung zu Anwendung (end-to-end QoS) bietet [CaAuHa95]. Übersicht über Netzwerk– und Transportanforderungen bieten auch [Stüttgen95] [VoKeBo95].

Allgemein gelten die folgenden Überlegungen:

- Durchsatz:
  Das Datenaufkommen ist sehr ungleichmäßig und kann in verschieden großen Schüben auftreten. Diese Schübe sind von der Datenmenge eines Elements und seiner Dauer abhängig. Die Mindestanforderung an den Durchsatz ist, daß jedes Datenelement innerhalb seiner Dauer übertragen wird. Trotzdem kann ein Über– beziehungsweise Unter–Laufen einer Quelle oder einer Senke nicht ausgeschlossen werden. Die maximale Anforderung ist, daß das Element mit der größten Datenmenge innerhalb der kürzesten in der Sequenz auftretenden Dauer eines Elements übertragen werden kann. Eine große Varianz der Dauern kann allerdings zu einer schlechten Ausnutzung des reservierten Durchsatzes führen. Genauere Analysen und mögliche sowohl anwendungs– als auch benutzerabhängige Verfahren zur Optimierung der Anforderungen sprengen den Umfang dieser Arbeit.
  - Ein einfaches Beispiel ist die unkomprimierte Video-Übertragung mit den Werten aus dem Beispiel der Steuerschnittstelle:
    minimale Dauer eines Elements: 1/30 Sekunde;
    Datenvolumen eines Elements: 2580480 bit;
    Es ergibt sich eine Anforderung von 30 × 2580480 bit pro Sekunde, also ca. 74 Mbit/sec.
- maximal zulässige Übertragungsverzögerung und
  deren zulässige Varianz :
  Die maximal zulässige Übertragungsverzögerung an der Datenschnittstelle ist von

der der Steuerschnittstelle abhängig und sollte nicht mehr als einen Schritt größer als jene sein; entsprechendes gilt für die zulässige Varianz der Übertragungsverzögerung. Der Grund für diese Anforderungen liegt in der Reduzierung von Sende– und Empfangspuffern für die möglicherweise sehr großen Sequenzelemente.

  - Im obigen Beispiel ergeben sich damit (1/10 + 1/30) = 0,13 Sekunden maximale Verzögerung und 1/30 Sekunde Varianz.

- Die akzeptierbare Fehlerwahrscheinlichkeit:
  Über die akzeptierbare Fehlerwahrscheinlichkeit für kontinuierliche Daten ist bekannt, daß unkomprimierte Daten auffallend hohe Fehlerraten vertragen, während komprimierte Daten schon bei geringen Fehlern wertlos werden; wieder entscheidet letztenendes der aktuelle Benutzer einer verteilten Anwendung darüber, welche Fehlerraten er noch akzeptieren will.

  - Für das unkomprimierte Videobild mit hoher Farbauflösung im obigen Beispiel sei eine Fehlerrate von 20% akzeptabel.

Auch an der Datenschnittstelle ist also die Dienstgüteanforderung parametrisiert und nur für die konkrete Ausprägung der Schnittstelle angebbar.

Ein weiteres Problem stellt die Beziehung zwischen den Zeitparametern der Sequenzelemente und den Ticks dar. Ein Sequenzelement sollte der Präsentation verfügbar sein, bevor der Tick zu seinem Startzeitpunkt ankommt. Mindestens muß aber die Präsentation begonnen haben, bevor der das Ende des Gültigkeitszeitraums des Elements bestimmende Tick ankommt. Diese Problem stellt sich besonders kraß bei der Verbindung von Perzeption und Präsentation.

Für die in Abbildung 48 spezifizierten Kommunikationskontexte der Datenschnittstellen sind nicht nur die an der Schnittstelle auszutauschenden Daten, sondern auch die Qualitätsanforderungen des Benutzers der verteilten Anwendung maßgeblich. Für diesen Kontext währe daher ein Dienst erforderlich, der es erlaubt, Transportanforderungen zur Laufzeit mit der Anwendung auf beiden Seiten zu verhandeln und eventuell nachzuverhandeln. Wie bereits bei der Schnittstellendefinition diskutiert, ist diese Forderung kritisch. Alternativen zu dieser Forderung betreffen die Komponentendefinition oder die Anwendungskonfiguration und werden dort besprochen.

Der Kommunikationskontext der Datenquelle entspricht dem *DataAccessOutMultiParty-Contex* mit der Kooperationsstruktur der Gruppenkommunikation für eine statische Gruppe von Komponenten mit einem ausgezeichneten Sender (*MULTI-PARTY (STATIC CENTRAL (SENDER)...*) an der Datenquelle und eventuell mehreren Empfängern (*MULTI-PARTY (STATIC CENTRAL (RECEIVER) ...)*) als Datensenken [Alireza94].

Die Schablone kann zunächst für jeden entwickelten Elementtyp mit Standardwerten ausgefüllt werden. Eine ausgefüllte Schablone ist in Abbildung 49 angegeben. Sollen für eine Komponente zusätzliche Fähigkeiten eingeführt werden, die den Datenaustausch betreffen, wie Zeitraffer oder Zeitlupe (ändern der Tickrate und/oder Schrittrate), so sind die Anforderungen entsprechend zu verändern.

```
ContinuousDataSourceContextTemplate
COMMUNICATIONCONTEXT
  TRANSPORT (
    SIMPLEX,
    THROUGHPUT: <x> Bit/sec,
    TRANSMITDELAY: <130> msec,
    DELAYJITTER: <33> msec
    TRANSFERFAILUREPROB: <20> %,
    )
MULTI-PARTY (
  STATIC CENTRAL (SENDER)
    GROUP_ONLY
    NO_FLOORCONTROL
    NONE
  )
  CONNECTION-MANAGEMENT(
  CONNECTIONORIENTED(
    ESTABLISH-INITIATOR
    RELEASE-INITIATOR
    REESTABLISHING)
    )
INTERACTION (CLIENT
  MESSAGES(
      UNCONFIRMED
      NO_RETRANSMITTING
      NO_CHECK_POINTS
      NO_ENCRYPTION
      NO_TOKEN_SUPPORT)
      )
::=ContinuousDataSourceContext
```

```
ContinuousDataSinkContextTemplate
COMMUNICATIONCONTEXT
  TRANSPORT (
    SIMPLEX,
    THROUGHPUT: <x> Bit/sec,
    TRANSMITDELAY: <130> msec,
    DELAYJITTER: <33> msec
    TRANSFERFAILUREPROB: <20> %,
    )
MULTI-PARTY (
    STATIC CENTRAL (RECEIVER)
      GROUP_ONLY
      NO_FLOORCONTROL
      NONE
    )
  CONNECTION-MANAGEMENT(
  CONNECTIONORIENTED(
      ESTABLISH-RESPONDER
      RELEASE-RESPONDER
      REESTABLISHING)
  )
INTERACTION (SERVER
    MESSAGES(
      UNCONFIRMED
      NO_RETRANSMITTING
      NO_CHECK_POINTS
      NO_ENCRYPTION
      NO_TOKEN_SUPPORT)
    )
::= ContinuousDataSinkContext
```

Abb. 48 Schablone für die Kommunikationskontexte der Datenschnittstelle [Alireza94]

```
ContinuousDataSourceContext
COMMUNICATIONCONTEXT
  TRANSPORT (
    SIMPLEX,
    THROUGHPUT: 74 MBit/sec,
    TRANSMITDELAY: 130 msec,
    DELAYJITTER: 33 msec
    TRANSFERFAILUREPROB: 20 %,
    )
MULTI-PARTY (
    STATIC CENTRAL (SENDER)
    GROUP_ONLY
    NO_FLOORCONTROL
    NONE
    )
CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
      ESTABLISH-INITIATOR
      RELEASE-INITIATOR
      REESTABLISHING)
    )
INTERACTION (CLIENT
    MESSAGES(
      UNCONFIRMED
      NO_RETRANSMITTING
      NO_CHECK_POINTS
      NO_ENCRYPTION
      NO_TOKEN_SUPPORT)
      )
::=ContinuousDataSourceContext
```

```
ContinuousDataSinkContext
COMMUNICATIONCONTEXT
  TRANSPORT (
    SIMPLEX,
    THROUGHPUT: 74 Mbit/sec,
    TRANSMITDELAY: 130 msec,
    DELAYJITTER: 33 msec
    TRANSFERFAILUREPROB: 20 %,
    )
MULTI-PARTY (
    STATIC CENTRAL (RECEIVER)
    GROUP_ONLY
    NO_FLOORCONTROL
    NONE
    )
CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
      ESTABLISH-RESPONDER
      RELEASE-RESPONDER
      REESTABLISHING)
  )
INTERACTION (SERVER
    MESSAGES(
      UNCONFIRMED
      NO_RETRANSMITTING
      NO_CHECK_POINTS
      NO_ENCRYPTION
      NO_TOKEN_SUPPORT)
      )
::= ContinuousDataSinkContext
```

Abb. 49 ein Datenschnittstellenkommunikationskontextpaar

5.7 Beschreibung der Komponenten

Aus den Funktionen auf kontinuierlichen Medien ergeben sich die Komponenten für Perzeption, Präsentation, lesenden und schreibenden Speicher, die sogenannten Grundfunktionskomponenten, sowie die Komponenten für die Verarbeitungsfunktionen. Dazu kommen Managementkomponenten und Steuerkomponenten zum Betrieb der verteilten Anwendung. Zu jeder dieser Komponentenarten lassen sich Schablonen angeben, die den Aufbau einer Komponente und die in der Komponente auftretenden Wechselwirkungen zwischen den Schnittstellen beschreiben. Die Schablonen unterscheiden sich von tatsächlichen Komponenten dadurch, daß die Schnittstellen nur durch typbestimmende Platzhalter besetzt sind, für die eine entsprechende verfeinerte Schnittstelle einzusetzen ist. Durch das Prinzip der Schablonen wird die Struktur der Komponenten klar, ohne durch eine Menge von Feinheiten das Wesentliche zu verdecken. Verfeinerungen zwischen den Komponenten beziehen sich auf ihre Schnittstellen und somit entweder auf die *set_attribute* Funktionen oder auf den an der Datenschnittstelle verwendeten Elementtyp, der zunächst in Abhängigkeit der menschlichen Sinne und dann bezüglich der Datenkodierung definiert wurde.

Die Abhängigkeiten der Schnittstellen einer Komponente untereinander enthalten immer wieder eine Implikation von Operationen der Form, daß eine an einer *"supplier-at"*–Schnittstelle aufgerufene Operation einen Aufruf einer Operation an einer "*consumer-at"*–Schnittstelle bewirkt. Besteht immer dieser zwingende Zusammenhang, so wird das in einem erweiterten Kooperationsprotokoll durch eine Implikation mit dem Zeichen => ausgedrückt.

5.7.1 Grundfunktionskomponenten

Die Grundfunktionen wurden datenflußabhängig in Quellen und Senken unterteilt. Bei der Definition der Komponenten wird diese Unterscheidung in der Polarität der Datenschnittstelle deutlich. Daher werden Komponentenschablonen für Quellenkomponenten und Senkenkomponenten entwickelt, die diese Unterschiede und die Abhänigkeiten zwischen den Schnittstellen verdeutlichen. Zur Entwicklung eigenständiger Komponenten sind dann jeweils die entsprechenden Schnittstellen für die Perzeption, Präsentation und den lesenden oder schreibenden Speicher einzusetzen. Dies kann zunächst wieder in Form von Schablonen geschehen, womit man verfeinerte Schablonen zu den Quellen und Senken erhält. Man kann aber auch direkt eigenständige Komponenten bilden. Die Spezifikationen aller Grundfunktionskomponenten würden hier zuviel Raum einnehmen; daher sind die einfachen Komponentenspezifikationen im Anhang zusammengefaßt.

Das Kooperationsprotokoll der Quellen fordert, daß eine Komponente durch die Managementfunktionen initialisiert beziehungsweise terminiert wird. Dazwischen werden die Operationen an der Steuer- und an der Datenschnittstelle verwendet. Da sich die Operationen an den Steuerschnittstellen der einzelnen Funktionen unterscheiden, und um nicht alle Operationen einzeln aufzählen zu müssen, werden sie hier durch "?" abgekürzt.

Die Quellenkomponente ist klar an der Polarität der Datenschnittstelle zu erkennen. An der Steuerschnittstelle kann jede Quellensteuerung eingesetzt werden. Die Quellen lassen sich damit zu Perzeption oder lesendem Speicher verfeinern. Es muß lediglich `<any_source_control>` durch die entsprechende Schnittstellenspezifikation ersetzt und das Kooperationsprotokoll spezifiziert werden.

Lose gekoppelte Quellenkomponenten lasen sich aus folgender Schablone ableiten:

```
<source_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
       data     : <continuous_data>;
    SUPPLIER AT
       control  : <any_source_control>;
  RESOURCES
       <source device>, <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
       data.setup <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_element)°

MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       (manage.init | data.setup) <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_element)°
       < manage.release

COMMUNICATION PROPERTIES
       <ContinuousDataSourceContextTemplate> at data;
       <ContMedFunctionControlContextTemplate> at control;
       ManagementContext at manage;
```

In der Definition der Quellenkomponente werden Schablonen für Kommunikationskontexte eingesetzt. Handelt es sich bei der Quellenkomponente allerdings um eine Komponente, die in enger Kopplung arbeiten soll, so muß ein anderer Kommunikationskontext, hier wieder eine Schablone, für die Steuerschnittstelle verwendet werden. Dieser ist in der folgenden Schablone eingesetzt.

```
<tightly_coupled_source_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
       data     : <continuous_data>;
    SUPPLIER AT
       control  : <any_source_control>;
  RESOURCES
       <source device>, <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
       data.setup <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_element)°
```

```
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       (manage.init | data.setup) <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_element)°
       < manage.release

COMMUNICATION PROPERTIES
       <ContinuousDataSourceContextTemplate> at data;
       <TightMedFunctionControlContextTemplate> at control;
       ManagementContext at manage;
```

Eng gekoppelte Quellenkomponenten können Perzeptionskomponenten sein. Als weitere Festlegungen im Kooperationsprotokoll der Perzeption kommen die folgenden Implikationen zwischen den Schnittstellen hinzu:

```
manage.init => data.setup
control.send_sequence_parameters => data.receive_seq_parameters
```

Das Kooperationsprotokoll des lose gekoppelten, lesenden Speichers kann um die folgenden Implikationen erweitert werden:

```
manage.init => data.setup
control.extract =>
     (data.receive_seq_parameters < data.receive_seq_element°)
control.send_seq =>
     (data.receive_seq_parameters < data.receive_seq_element°)
```

Auch eine Senkenkomponente wird eindeutig durch die Polarität der Datenschnittstelle gekennzeichnet. Für die Schnittstellenspezifikation *any_sink_control* kann jede davon abgeleitete Schnittstelle eingesetzt werden. Das Kooperationsprotokoll besagt, daß die Operationen an der Steuer– oder der Datenschnittstelle nur nach einer Initialisierung und vor der Terminierung verwendet werden dürfen. Weitere Einschränkungen sind an der Senke nicht nötig.

Lose gekoppelte Senkenkomponenten lassen sich aus folgender Schablone ableiten:

```
<sink_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
      SUPPLIER AT
       data     : <continuous_data>
       control  : <any_sink_control>
  RESOURCES
       <sink device>, <memory>, <schedule requirements>;
```

```
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_element)°
       < manage.release

COMMUNICATION PROPERTIES
       <ContinuousDataSinkContextTemplate> at data
       <ContMedFunctionControlContextTemplate> at control
       ManagementContext at manage;
```

Analog zu den Quellen mit enger Kopplung kann eine Schablone für Senken in enger Kopplung angegeben werden.

```
<tightly_coupled_sink_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       SUPPLIER AT
       data     : <continuous_data>
       control  : <any_sink_control>
  RESOURCES
       <sink device>, <memory>, <schedule requirements>;

MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_element)°
       < manage.release

COMMUNICATION PROPERTIES
       <ContinuousDataSinkContextTemplate> at data
       <TightMedFunctionControlContextTemplate> at control
       ManagementContext at manage;
```

5.7.2 Verarbeitungskomponenten

Die Komponenten, die die Verarbeitungsfunktionen realisieren, werden auch Werkzeuge genannt. Werkzeuge können unterschiedlich aufgebaut sein, es gibt folgende Varianten:

- ohne Steuerschnittstelle
- mit Steuerschnittstelle
- genau zwei Datenschnittstellen für die Funktionen f_1 bis f_4
- mehrere Datenschnittstellen für die Funktion f_5

Die Varianten mit oder ohne Steuerschnittstelle lassen sich beliebig mit den Varianten mit zwei Datenschnittstellen und mit mehreren Datenschnittstellen kombinieren. Damit ergeben sich je zwei Möglichkeiten für die Funktionen f_1 bis f_4 und die Funktion f_5 . Eine umfassende Behandlung der Werkzeuge ist in [Alireza94] zu finden.

5.7.3 Steuerkomponenten

Steuerkomponenten regeln das Wechselspiel zwischen Quellen, Senken und Werkzeugen einer Multimediaanwendung. Sie können dabei als interaktive Komponenten direkt von einem Benutzer bedient oder über eine weitere Schnittstelle von anderen Komponenten verwendet werden. Die Komponentenspezifikation kann nur ganz allgemein als Schablone angegeben werden, da Anzahl und Typ der Schnittstellen anwendungsabhängig sind. Zunächst wird diese Schablone spezifiziert. Einige einfache, konkrete Steuerkomponenten werden im Beispiel der Anwendungskonfiguration vorgestellt.

Die Steuerkomponente realisiert auch die Einbettung der zeitabhängigen Funktionen in ihre Umgebung. Sie sorgt dafür, daß Perzeptions– und Präsentationskomponenten ihre Ticks erhalten. Die Unterscheidung in enge und lose Kopplung an den Kontexten der Schnittstellen liefert die Information dafür, ob regelmäßig Ticks übertragen werden müssen.

In der Regel wird nur eine Steuerkomponente in einer Anwendung vorkommen. Das ändert sich, falls mit den kontinuierlichen Medien kooperativ gearbeitet wird, oder falls Anwendungsteile hierarchisch zu Komponenten zusammengefaßt werden. Auf die durch hierarchische Zusammenfassung strukturierten Komponenten wird in der Anwendungskonfiguration eingegangen.

Die Arbeitsweise der Steuerkomponente läßt sich im Rahmen der in diesem Kapitel verwendeten Technik beschreiben, wenn die Komponente eine Bediener–Schnittstelle anbietet. Das heißt, die dem Benutzer interaktiv angebotenen Operationen werden als von der Komponente angebotene Schnittstelle modelliert. In der Komponentenspezifikation ist diese Schnittstelle angegeben, sie wird für interaktive Komponenten aber auskommentiert. Die einfachste Bediener–Schnittstelle bietet nur die Operationen des Tickers an, also die Steuerung der Umgebung. Diese Schnittstelle läßt sich auch als Komponentenschnittstelle beschreiben; sie benötigt kein Kooperationsprotokoll.

```
tick_control
INTERFACE
        CONSUMER INVOKES
        set_rate; set_internal_rate;
        get_rate; get_internal_rate;
        set_tick;
        get_tick;
```

Der Benutzer benötigt keine zusätzlichen Operationen um eine einfache Anwendung zu steuern. Mit dieser Schnittstelle kann daher eine Schablone für die Steuerkomponente erstellt werden.

```
<control_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       /* SUPPLIER AT tick_control */
       CONSUMER AT
       sink_control_1    : <any_sink_control>
                :
       sink_control_n    : <any_sink_control>

       source_control_1  : <any_source_control>;
                :
       source_control_m  : <any_source_control>;
  RESOURCES
       <memory>, <schedule requirements>;

MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (sink_control_<?>.?;source_control_<?>.?;tick_control)°
        < manage.release;

COMMUNICATION PROPERTIES
       <ContMedFunctionControlContextTemplate>
                        at <control_x>,.. ,<control_z>
       <TightMedFunctionControlContextTemplate>
                        at <control_k>,.. ,<control_l>
       ManagementContext at manage;
```

An dieser Stelle wird gezeigt, wie die Steuerkomponente die Prototypschnittstelle realisiert. Eine Operation der Prototypschnittstelle erfordert dabei Operationsfolgen an anderen Schnittstellen, die zum Teil transaktionsartig zusammengefaßt werden:

- *start:*
  ein Start der Anwendung geschieht, falls die interne Tickrate von 0 auf einen von 0 verschiedenen Wert wechselt. Dann erzeugt die Steuerkomponente Ticks und verteilt diese.
- *pause:*
  bei Pause wechselt die interne Tickrate von einem von 0 verschiedenen Wert auf den Wert 0, der aktuelle Tickwert bleibt erhalten und wird zweimal hintereinander an alle zeitabhängigen Funktionen übertragen. Somit erhalten auch lose gekoppelte Funktionen den Hinweis auf das Anhalten des Tickers.
- *stop:*
  Die (interne) Tickrate wechselt auf den Wert 0, der neue Tickwert wird auf 0 gesetzt und zweimal hintereinander an alle zeitabhängigen Komponenten übertragen. Vor dem nächsten Start müssen alle Perzeptionskomponenten die Aufforderung zum Senden der Sequenzparameter erhalten.

Es bleibt dem Konstrukteur der verteilten Anwendung überlassen, ob er die Steuerschnittstellen der einzelnen Funktionen, die Tickersteuerung oder die Prototypschnittstelle an der Benutzeroberfläche zugänglich macht. Alternativ kann als Bedienschnittstelle der Steuerkomponente auch eine gemeinsame Ableitung aus der Prototypschnittstelle mit der obigen Interpretation und der Tickersteuerung verwendet werden. Weitere Operationen des Benutzers und ihre Realisierungsmöglichkeiten werden in [WiRoWa95] untersucht. Dabei ergeben sich einige Parallelen zu der hier skizzierten Vorgehensweise der Realisierung der Benutzerschnittstelle, die die Effektivität der Strategie bestätigen.

Diese Alternative läßt sich zu einer flexiblen Steuerung der Komponenten erweitern, wenn zusätzlich alle enthaltenen Schnittstellen der Funktionen kontinuierlicher Medien an die Bedienschnittstelle vererbt werden. Diese erweiterte Alternative ermöglicht es, einzelne Komponenten getrennt zu starten, zu pausieren oder zu stoppen, indem man statt des Tickers deren Schrittgeber beeinflußt. Eine Steuerkomponente mit diesem Funktionsumfang bietet dann eine Schnittstelle der folgenden Art an:

```
user_control
INTERFACE
      DERIVED FROM tick_control, prototype_control,
      <any_source_control>, <any_sink_control>
```

5.8 Anwendungskonfiguration

In der Anwendungskonfiguration wird beschrieben, welche Komponenten für die Funktionen auf kontinuierlichen Medien und welche Steuerkomponenten in der verteilten Anwendung verwendet werden. Dazu kommt deren Plazierung auf den Knoten und die Verbindungsstruktur zwischen den Komponenten. Da allgemeine Anwendungskonfigurationen bereits in Kapitel 3 besprochen wurden und in [SteiMey92] ausführlich beschrieben sind, werden hier keine weiteren Beispiele angeführt. Schablonen für die Konfiguration einfacher Aufzeichnungssysteme und einfacher Wiedergabesysteme, wie sie in Kapitel 3 definiert wurden, werden als allgemeine Anwendungsbeispiele vorgestellt. Da keine aktuelle Zuordnung zu Knoten vorgenommen werden kann, und keine Komponenteninstanzen eingesetzt werden, ist die Schablonenschreibweise das geeignete Darstellungsmittel. Aus den Anwendungsbeispielen ergeben sich geschlossene Anwendungen. Offene Anwendungen erhält man, falls in der Konfiguration nicht gebundene Schnittstellen auftreten, an denen weitere Komponenten oder wieder offene Anwendungen angefügt werden können.

5.8.1 Beispielanwendungen

5.8.1.1 Ein einfaches Aufzeichnungssystem

Ein Aufzeichnungssystem ist durch die Verbindung von Perzeption und schreibendem Speicher gekennzeichnet. Es werden also eine Perzeptionskomponente, eine Schreib–Speicher–Komponente und eine entsprechende Steuerkomponente benötigt. Die Steuerkomponente sei eng an die Perzeptionskomponente gekoppelt und soll auf dem selben Knoten liegen, sie wird als interaktive Komponente angesehen und ermöglicht dem Benutzer den Zugriff auf die Schnittstellen der gebundenen Komponenten.

Zunächst wird die entsprechende Steuerkomponente als Schablone spezifiziert. Anschließend kann die Anwendungsschablone erstellt werden. Aus diesen Schablonen lassen sich Aufzeichnungssysteme für beliebige Medien herleiten.

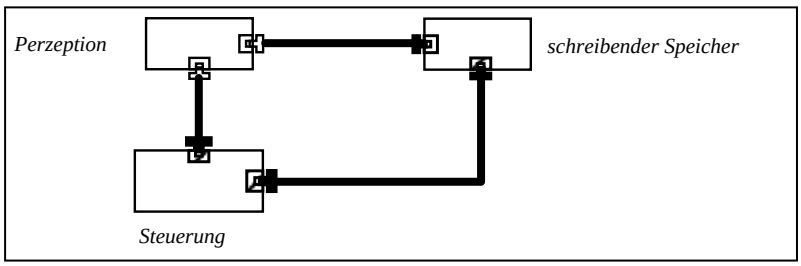


Abb. 50 graphische Darstellung der Komponenten eines Aufzeichnungssystems

```
<record_control_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       /* SUPPLIER AT user_control */
       CONSUMER AT
       sink_control      : <simple_write_control>
       source_control    : <perception_control>;
  RESOURCES
       <memory>, <schedule requirements>;

MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage            : <cont_media_management>;

  COOPERATION PROTOCOL
       manage.init <
       (sink_control.?;source_control.?)° <
       manage.release

COMMUNICATION PROPERTIES
       <ContMedFunctionControlContextTemplate> at sink_control
       <TightMedFunctionControlContextTemplate> at source_control
       ManagementContext at manage;


<recording_system> DISTRIBUTED APPLICATION
       COMPONENTS
       perception : <tightly_coupled_perception_component>;
       store :  <simple_write_component>;
       control: <record_control_component_template>;
       BINDINGS
       perception.data -- store.data;
       perception.control -- control.source_control;
       store.control -- control.sink_control;
```

```
      CONSTRAINTS
      PLACEMENT
        NODES={ node1, node2}
         perception AT node1;
         control AT node1;
         store AT node2
```

5.8.1.2 Ein einfaches Wiedergabesystem

Ein einfaches Wiedergabesystem ist das Gegenstück zum Aufzeichnungssystem. Die zugehörigen Komponenten sind die Präsentation, der lesende Speicher und eine Steuerkomponente. Letztere ist eng an die Präsentation gebunden und sollte bevorzugt auf dem selben Rechner liegen. Wie beim Aufzeichnungssystem werden Schablonen für die Steuerkomponente und die Anwendung spezifiziert. Die geringfügigen aber entscheidenden Unterschiede zum Aufzeichnungssystem werden dabei durch Unterstreichung hervorgehoben.

```
<play_control_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       /* SUPPLIER AT user_control */
       CONSUMER AT
       sink_control      : <presentation_control>;
       source_control    : <simple_read_control>
  RESOURCES
       <memory>, <schedule requirements>;

MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage            : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (sink_control.?;source_control.?) <
       < manage.release

COMMUNICATION PROPERTIES
       ContMedFunctionControlContextTemplate at source_control
       TightMedFunctionControlContextTemplate at sink_control
       ManagementContext at manage;


<replay_system>_DISTRIBUTED APPLICATION
       COMPONENTS
       presentation : <tightly_coupled_presentation_component>;
       store :  <simple_read_component>;
       control: <play_control_component_template>;
       BINDINGS
       presentation.data -- store.data;
       presentation.control -- control.source_control;
       store.control -- control.sink_control;
```

```
CONSTRAINTS
PLACEMENT
  NODES={ node1, node2}
   presentation AT node1;
   control AT node1;
   store AT node2
```

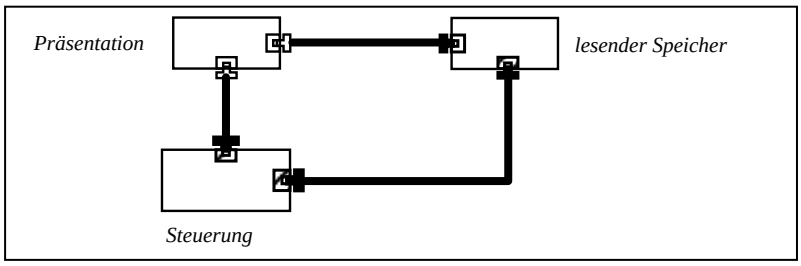


Abb. 51 graphische Darstellung der Komponenten eines Wiedergabesystems

5.8.1.3 Bearbeitungssysteme

Bearbeitungssysteme sind Anwendungen, die aus mindestens einem lesenden und einem schreibenden Speicher bestehen und mindestens eine Verarbeitungsfunktion verwenden. Beispiele für Bearbeitungssysteme sind einfache Anwendungen der Verarbeitungsfunktionen, wie umkodieren der Werte, oder Erhöhung der Helligkeit bei Video oder verändern der Lautstärke bei Audio. Weitere Beispielanwendungen ergeben sich aus dem Datenmodell, wie die Erstellung von Normalformen der Sequenzen. Dabei kann für die erste Normalform auf verschiedene Sortierverfahren zurückgegriffen werden. Berücksichtigt man die zum Teil erhebliche Menge der Daten und ihre sequentielle Organisation, die sich auch im Speichern der Medien auf Bändern ausdrückt, liegt ein Bändermischverfahren nahe. Andererseits bietet der wahlfreie Zugriff auf die Sequenzen auch die Möglichkeit jedes Sortierverfahren, das auf Vertauschen beruht, zu verwenden. Je nach dem gewählten Verfahren und der verwendeten Sortierfunktion wird eine mächtige Verarbeitungsfunktion oder eine größere Anzahl von Speicherkomponenten benötigt. Ihre Beschreibung kann daher recht umfangreich werden, ohne neue Aspekte darzustellen. Daher wird hier auf eine entsprechende Anwendungskonfiguration verzichtet. Beispiele für Bearbeitungssysteme sind ausführlich in [Alireza94] behandelt.

5.8.2 multifunktionale und multimediale Komponenten

Verändert man die Definition der Steuerkomponenten im Beispiel des Aufzeichnungssystems oder des Wiedergabesystems derart, daß eine Bedienerschnittstelle eingefügt wird, so erhält man eine offene Anwendung. Die offene Anwendung ist genau über die Bedienerschnittstelle zugänglich. Diese Anwendung kann auch als hierarchische Komponente beschrieben werden, wobei die einzige Schnittstelle der hierarchischen Komponente genau der Bedienerschnittstelle entspricht. Diese hierarchische Komponente bezeichnet man auch als multifunktionale Komponente, da sie mehrere verschiedene Funktionen in sich vereinigt.

```
<record_composed_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       SUPPLIER AT coordination.user_control

  RESOURCES
       <memory>, <schedule requirements>;
  SUBCOMPONENTS
       coordination : <record_control_component_template>
       perception   : <tightly_coupled_perception_component>;
       store        : <simple_write_component>;
       BINDINGS
       perception.data -- store.data;
       perception.control -- coordination.source_control;
       store.control -- coordination.sink_control;


MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage           : <cont_media_management>;

  COOPERATION PROTOCOL
       manage.init <
       (coordination.user_control.?;
        coordination.user_control.?)°
       < manage.release

COMMUNICATION PROPERTIES
       <anyContext> at coordination.user_control
       ManagementContext at manage;
```

Abb. 52 Spezifikation einer zusammengesetzten Aufnahme–Komponente

Man kann die hierarchische, multifunktionale Komponente auch dahingehend verändern, daß die Datenschnittstellen nicht per Definition verbunden, sondern ebenfalls nach außen verfügbar sind.

Analog erhält man multimediale Komponenten, wenn offene Anwendungen zu einer Komponente zusammengefaßt werden, die die gleiche Funktion aber auf verschiedenen Medien ausführen. Es lassen sich somit zum Beispiel Komponenten definieren, die gleichzeitig Audio und Video aufnehmen und an getrennten Schnittstellen ausgeben, oder unter Verwendung einer Verarbeitungsfunktion gemeinsam kodieren und an einer Schnittstelle ausgeben. Beispiele für multifunktionale und multimediale Komponenten sind in [Konst93] und in [Alireza94] zu finden.

5.8.3 Auswahl von Komponenten

Die entwickelten Kommunikationskontexte, Schnittstellen und vor allem die Komponententypen können zur Erleichterung der Wiederverwendung in einer Bibliothek gesammelt werden. Mittels graphischer Konfigurationswerkzeuge können unter Verwendung der Bibliothek Anwendungen interaktiv spezifiziert werden. Dabei entscheidet die Lei-

stungsfähigkeit der Bibliothek über den Unterscheidungsgrad und den Suchaufwand für Komponenten. So lassen sich durch die Verwendung von Anwendungsschablonen sehr schnell und gezielt die gewünschten Funktionalitäten erreichen. In Abbildung 53 ist das Prinzip der Komponentenauswahl dargestellt. Ein entsprechendes Konfigurationswerkzeug wird in [Dömel95] vorgestellt.

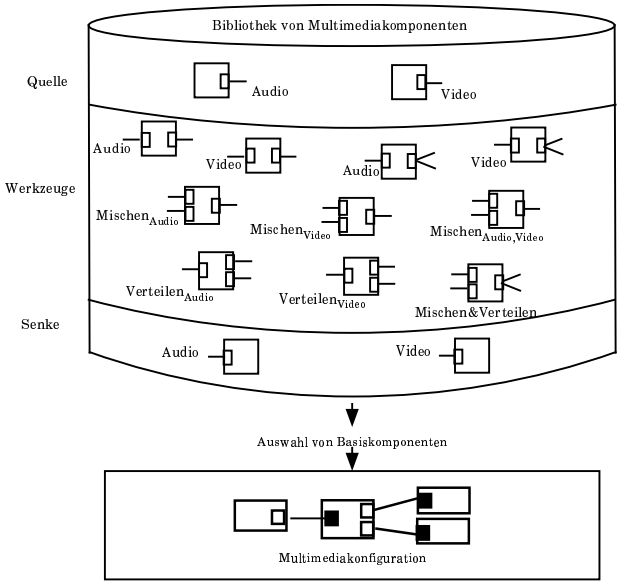


Abb. 53 Konfiguration von Komponenten

5.8.4 Umkonfiguration der Anwendung

Die Funktionen des Managementsystems in [Zimm93] erlauben eine Umkonfiguration einer Anwendung zur Laufzeit. Insbesondere ist dabei an langlebige Anwendungen gedacht, wo neue Komponenten hinzukommen können, aktuelle Instanzen von Komponenten durch neue Versionen dieser Komponenten ersetzt werden, oder Komponenten ausscheiden, da sie nicht mehr gebraucht werden [Zimm93]. Dazu muß das Management im laufenden Betrieb Komponenten beenden und neue starten können. Das Erzeugen einer Komponente umfaßt die Aufgaben Software laden, Gerät belegen, Prozesse starten; das Beenden einer Komponente umfaßt die laufende Bearbeitungen abschließen, Prozesse terminieren, Geräte freigeben.

Durch die besonderen Anforderungen an die Knoten und die Übertragung, die die Komponenten der kontinuierlichen Medien stellen, kann es vorkommen, daß das Management bereits beim Starten einer Anwendung auf unerfüllbare Anforderungen trifft. Das gilt vor allem für die Dienstgütenanforderungen der Kommunikation.

Generell treffen in einer Komponente drei Dienstgüten für die Datenschnittstelle aufeinander. Das sind die vom Kommunikationskontext geforderte, die vom Gerät maximal bereitgestellte und die vom Kommunikationssystem maximal gelieferte. Die vom Gerät maximal bereitgestellte Dienstgüte muß mit der vom Kommunikationssystem geforderten entweder di-

rekt harmonieren oder innerhalb der Komponente angeglichen werden. Ist das nicht der Fall, kann die Komponente nicht dieses Gerät belegen. Der Kommunikationskontext garantiert, daß alle gebundenen Komponenten eine ausreichende Dienstgüte bereitstellen können.

Falls das Kommunikationssystem nicht die entsprechende Leistung bereitstellen kann, muß entweder die Komponente die Anforderungen senken, indem sie zum Beispiel die Datenkodierung ändert, oder es muß eine Ersatzkonfiguration von Komponenten mit geringeren Anforderungen gefunden werden. Dazu ist im allgemeinen der Benutzer interaktiv tätig, um entweder Komponenten direkt zu ersetzen, oder Verarbeitungskomponenten einzusetzen, die in der Lage sind die Kommunikationsanforderungen, zum Beispiel durch umkodieren der Daten, zu senken.

6 Resümee und Ausblick

In diesem Abschnitt wird eine Gesamtrückschau auf die Arbeit gegeben und dazu beschrieben, welche Konzepte bereits realisiert sind. Anschließend werden mögliche Anwendungen und Erweiterungen vorgestellt.

6.1 Entwicklung der Konzepte und Modelle

Das Ziel der vorliegenden Arbeit war die Entwicklung einer komfortablen Beschreibung verteilter Anwendungen, die kontinuierliche Medien integrieren. Die Klarheit des Ansatzes ergibt sich aus der Beschränkung auf die anwenderrelevanten Funktionalitäten. Weitere Gebiete, die systembezogen sind, wurden nur soweit wie nötig behandelt. Die Aufgaben anderer Bereiche, wie des Betriebssystems und des Managementsystems sowie der Kommunikationsdienste, konnten nur gestreift werden, indem die anwendungsabhängigen Anforderungen spezifiziert wurden. Durch deren Extraktion und die Zuordnung der Anforderungen an die einzelnen Bereiche, ergibt sich eine klarere Sicht auf Betriebssystem, Management und Kommunikationsdienste und deren notwendige Weiterentwicklung.

Das entwickelte Funktionenmodell beschreibt zusammenhängend alle mit kontinuierlichen Medien verbundenen Arbeiten. In der vorliegenden Arbeit wurde gezeigt, wie aus den Funktionen auf kontinuierlichen Medien durch die Spezifikation geeigneter Schnittstellen Bausteine zur Integration der Medien in verteilte Anwendungen erstellt werden. Die Beschreibung der Bausteine erfolgt durch diese Schnittstellen; es sind Steuer–, Daten– und Managementschnittstellen. Die Herauslösung der gesonderten Beschreibung der Multimedia−Datenflußstruktur schafft einerseits die Grundlage für eine Teilklassifikation der Anwendungen nach Medien-Gesichtspunkten. Andererseits kann die Erstellung einer Anwendung aus einer bestimmten Anwendungsklasse, wie zum Beispiel ein einfaches Wiedergabesystem, durch die gesonderte Beschreibung der Multimedia-Datenflußstruktur schneller in der Bausteinstruktur realisiert werden. Das Funktionenmodell wird auch in [Fritzsche96] beschrieben.

Das in dieser Arbeit konzipierte Bausteinmodell gewährleistet eine integrierte Beschreibung von Geräten, Werkzeugen und Anwendungen kontinuierlicher Medien. Die verwendete Beschreibungstechnik erlaubt dabei nicht nur eine übersichtliche Darstellung sondern bietet auch hierarchische Strukturierungen an. Das Zusammenspiel der Bausteine erfordert zusätzliche Komponenten zur Steuerung und Abstimmung der einzelnen Funktionen, die in dieser Arbeit neu eingeführt werden. Es lassen sich sowohl zentralistische als auch verteilte Steuerungen realisieren. Mit einer entsprechenden Schnittstelle versehen kann eine Steuerkomponente eine ganze Gruppe von Bausteinen dem Benutzer als Einheit zur Verfügung stellen. Somit lassen sich auch verschiedene Medien und/oder mehrere Funktionen gemeinsam mit einer Steuerkomponente zu einem Baustein zusammenfassen. Diese zusammengesetzten Bausteine bieten nun echte Multifunktionalität und Multimedialität. Durch die Komponenten– und Anwendungsmodellierung nach [Zimm93] wird darüber hinaus eine flexible, auch dynamisch änderbare Anwendungsstruktur vom Anwendungs–Management ermöglicht. Das Bausteinmodell wird auch in [Fritzsche96] behandelt.

Bisherigen Ansätzen für Multimedia−Komponenten fehlt die allgemeine Interoperabilität der Komponenten. Diese kann nur durch eine umfassende, formale Spezifikation der Komponenten−Schnittstellen, insbesondere aber von Steuerschnittstellen, erfolgen. Zur Spezifikation der Schnittstellen ist die Integration der kontinuierlichen oder zeitabhängigen Medien als abstrakte Datentypen unabdingbar. Auf diese Art werden aus den Komponenten Bausteine. Im vorliegenden Ansatz wurden erstmalig Steuerschnittstellen für Multimedia−Komponenten spezifiziert und als Hierarchie dargestellt. Der neue Ansatz erlaubt es daher, multimediale Systeme nach einem Baukastensystem zu erstellen, indem Bausteine durch Bindung untereinander zu einer Anwendung zusammengesetzt werden. Nach der Verbindungsstruktur der multimedialen Anwendung können verschiedene Anwendungstypen unterschieden werden.

Die Definition der Komponentenschnittstellen bezieht sich auf ein abstraktes Datenmodell für kontinuierliche Medien. Das Datenmodell ist eine eigenständige Weiterentwicklung der Ansätze von [Herrtw91] und [Gibbs94] und kann auch zur Realisierung der Komponenten verwendet werden. Multimediadaten wurden zunächst auf zwei Ebenen als *Sequenz* und *Sequenzelemente* modelliert. Daraus lassen sich bereits einige Funktionen auf den Daten ableiten, die von den Bausteinen realisiert werden müssen. Kennzeichnend für die Sequenzelemente ist, daß sie die Zeitparameter Zeitpunkt und Dauer besitzen und damit eine explizite Integration der Zeit in das Datenmodell realisieren. Aus diesen Parametern der Elemente können auch für die Sequenz die Parameter Zeitpunkt und Dauer abgeleitet werden. Somit könnte eine Sequenz selbst wieder Element einer Sequenz werden. Da diese Sequenzen von Sequenzen aber zum Teil schwer zu handhaben sind und zum Aufbau von sehr komplexen Verschachtelungen verleiten, wird in dieser Arbeit eine andere Erweiterung der Datenhierarchie, eine Liste, vorgestellt. Diese Erweiterung führt nur eine weitere Hierarchieebene oder Granularitätsstufe ein, ist aber durch die vorgegebenen Funktionen gleichmächtig wie die Verschachtelung der Sequenzen, im Operationsablauf aber leichter nachzuvollziehen. Die Liste repräsentiert die gröbste Granularitätsstufe. Diese ist mit der Titelfolge einer Schallplatte oder einer CD vergleichbar. Die einzelnen Teile haben zueinander nur eine lose Ordnung. In der ersten Verfeinerung der Granularität wird in jedem einzelnen Listenelement eine strenge zeitliche Ordnung gefordert; ein Listenelement ist eine Sequenz. In der zweiten Stufe der Verfeinerung, der Unterteilung der Sequenzen, treten die bereits bekannten Sequenzelemente auf.

Die Daten werden im Ticker–Schrittgeber–Modell interpretiert. Dieses Modell erhält zwei Zeitebenen, den Ticker als Bezugssystem der Funktionen untereinander und den Schrittgeber als Steuerung der einzelnen Funktionen. Ein zweistufiges Uhrenmodell mir festgesetzten Operationen und Uhrenbeziehungen wird in dieser Arbeit neu eingeführt. Die Beziehung zwischen Schrittgeber und Ticker ist, daß ein Schritt nach einer bestimmten Anzahl von Ticks erfolgt.

Der Startwert des Tickers kann frei gewählt werden, ebenso der Startwert des Schrittgebers. Für den Schrittgeber bestimmt sein Start–Tick, wann er beginnt fortzuschreiten. Ein Schrittgeber ist mit genau einer Sequenz verbunden, deren Start–Schritt beschreibt, bei welchem Schrittwert das erste Sequenzelement gültig wird. Die Start–Zeitpunkte der Elemente und ihre Dauern werden in Schritten gemessen. Das Datenmodell für Multimedia wurde in [Fritzsche95] veröffentlicht.

Als Grundlage für die Entwicklung der Bausteine zur Integration kontinuierlicher Medien in verteilte Anwendungen wurden die Funktionen auf den Medien herangezogen. Diese sind in ihren einfachsten Formen die Grundfunktionen Perzeption, Präsentation und Speicherung der Medien, wobei die Speicherung in die Funktionen Schreiben in den Speicher und Lesen aus dem Speicher geteilt wird. Die durch die Perzeption festgelegten, oder künstlich erzeugten Mediendaten können zwischen den einzelnen Funktionen übertragen werden. Eine Bearbeitung der Daten ist beim Austausch zwischen den Funktionen möglich. Die Veränderung der Daten und ihr Bezug zu den Grundfunktionen wird durch die Verarbeitungsfunktionen der Typen f_1 bis f_5 beschrieben. Die Funktionen werden durch Operationen gesteuert, die aus dem Datenmodell abgeleitet werden. Insbesondere wird so auch die explizite Veränderung der Zeitparameter möglich. Somit bietet das Datenmodell eine geeignete Grundlage für jede Art der Verarbeitung kontinuierlicher Medien.

Das entwickelte Modell unterstützt die Anwendungserstellung durch objektorientierte Ansätze auf den Ebenen der Konzeption, der Anwendungsspezifikation und der Komponentenentwicklung. Konzeptionell bietet das Funktionenmodell die schnelle und übersichtliche Darstellung der Anwendung. Die aus dem Funktionenmodell ableitbare Anwendungsspezifikation unterstützt die weitere Entwicklung durch Anwendungs– und Komponentenschablonen, sowie durch die vorgefertigte und erweiterbare Hierarchie der Schnittstellen und durch die Bibliotheken für Standardbausteine. Die Verwendung dieser Elemente der Anwendungsspezifikation läßt sich teilweise automatisieren. Das Ergebnis der Anwendungsspezifikation ist eine Menge von Komponenten, die alle vollständig spezifiziert sind. Diese Komponenten sind die funktionsorientierten Bausteine zur Integration kontinuierlicher Medien in verteilte Anwendungen.

6.2 Implementierungen

Im ersten Schritt wurde das vorgestellte Datenmodell mit seinen Operationen in einer objektorientierten Programmiersprache (C^{++} [Lipp91]) implementiert [Braun92]. Darauf aufbauend wurden verschiedene Anwendungsfunktionen und Normalisierungsoperationen entwickelt und für den Bereich Audio realisiert [Bast93]. Die von den Funktionen auf kontinuierlichen Medien abgeleiteten Bausteine werden, wie in der vorliegenden Arbeit ausführlich dargestellt, als Komponenten verteilter Anwendungen realisiert. Aus den verschiedenen Realisierungsebenen sollen hier zwei Beispiele hervorgehoben werden. Zunächst wird auf die Komponentenrealisierung eingegangen; danach folgt die Realisierung von Tickern und enger Kopplung. Diese beiden Punkte stellen zentrale Aufgaben des Ansatzes dar.

6.2.1 Realisierung von Komponenten

Die Realisierung der Komponenten gliedert sich in zwei Abschnitte. Der erste Abschnitt ist die Zerlegung einer Komponente in Standardobjekte nach [Zimm93]. Die Standardobjekte entstammen Kommunikationsklassen, Stub- und Dispatcherklassen, Anwendungsklassen und Kooperationsprotokollklassen. Die Objekte der Anwendungsklassen realisieren die Anwendungsfunktionalität der Komponente. Das Ausprogrammieren dieser Objekte stellt den zweiten Abschnitt der Komponentenrealisierung dar. Dazu liefert das entwickelte Datenmodell die Programmierunterstützung.

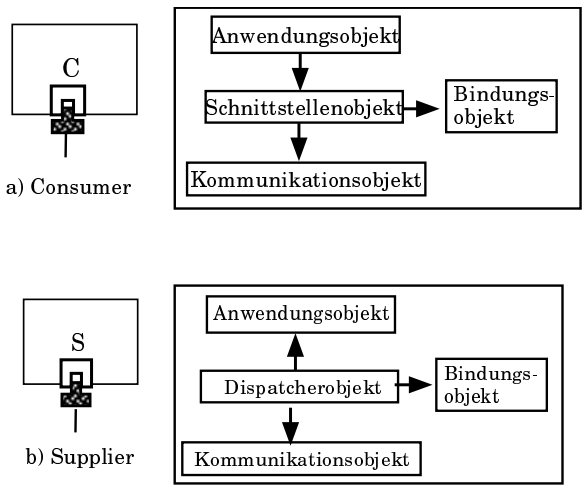


Abb. 54 Elementare Objektarchitektur [Zimm93]

Zur Abbildung der Spezifikationskonstrukte der Komponenten auf Implementierungskonstrukte wird in [Zimm93] eine Methode vorgestellt, die die unterschiedlichen Konstrukte für Schnittstellen, Kommunikationskontexte und Komponenten auf Klassen und Objekte abbildet. So entsteht eine Klassenhierarchie von C^{++} Klassen [Lipp91] für kommunikations-, anwendungs- und managementorientierte Objekte. Weiterhin wird in [Zimm93] ein Verfahren vorgestellt, durch das in Abhängigkeit von den Eigenschaften einer Komponente parallel ablaufende Datenflüsse in ein System von leichtgewichtigen Prozessen (Threads) transformiert werden können. Als Resultat gewinnt man eine modulare Softwarearchitektur der Komponente, die sich aus interagierenden Objekten und zugehörigen Threads zusammensetzt. In [Zimm93] werden folgende Objektklassen unterschieden:

- Kommunikationsklassen
- Stub- und Dispatcherklassen
- Anwendungsklassen
- Kooperationsprotokollklassen

Eine elementare Objektarchitektur aus diesen Klassen ist in Abbildung 54 dargestellt. Es gibt jeweils eine Realisierung für eine Supplier–Komponente und eine Consumer–Komponente. Die Anwendungsobjekte können bezüglich ihrer Funktionalität in initiierende und akzeptierende Objekte eingeteilt werden.

Im Falle unidirektionaler Schnittstellen sind die Anwendungsobjekte auf der Konsumentenseite (z.B. Benutzerkomponente) für die Initiierung von Methoden an Schnittstellenobjekten verantwortlich. Beispielsweise ist ein Anwendungsobjekt innerhalb der Benutzerkomponente für die Initiierung der Steueroperationen verantwortlich. Im Falle von interaktiven Komponenten [Zimm93] erfolgt dazu ein Benutzerdialog mit einem interaktiven Benutzer. Also realisiert innerhalb der Benutzerkomponente das Anwendungsobjekt einen solchen Benutzerdialog. Anwendungsobjekte auf der Konsumentenseite stellen somit typischerweise keine eigenen Methoden bereit, sondern bestehen lediglich aus einem Konstruktor. Auf der akzeptierenden Seite, den Anbieter (Supplier), realisiert ein Anwendungsobjekt die Operationen an einer Schnittstelle. Dazu wird eine Methode *accept* benötigt, falls ein verbindungs-

orientierter Kommunikationskontext zugrunde liegt. Diese Methode dient der Behandlung eingehender Verbindungswünsche. In [Alireza94] werden verschiedene Komponentenrealisierungen ausführlich vorgestellt.

6.2.2 Realisierung von Tickern, Schrittgebern und enger Kopplung

Die Realisierung der Ticker und Schrittgeber stellt die Einbettung der zeitbezogenen Komponenten in ihre (Betriebssystem-) Umgebung dar. Ähnlich, wie eine Komponente über den Socketmechanismus Zugang zum Kommunikationssystem erhält, erhält eine zeitbezogene Komponente über den Ticker–Schrittgeber–Mechanismus Zugang zum Zeitbezugssystem. Denn die Schrittgeber beziehen sich auf Ticker, Ticker aber auf die Systemzeit. Da auch die Systemzeit als Takt zur Verfügung gestellt wird, können Ticker und Schrittgeber wegen ihrer ähnlichen Funktionalitäten aus einer gemeinsamen Zeitgeberklasse abgeleitet werden. Im Anhang C ist die Deklaration dieser gemeinsamen Klasse angegeben.

In einer Anwendung beziehen sich die Schrittgeber verschiedener Komponenten auf einen gemeinsamen Ticker. Dieser Ticker liegt in der Systemumgebung der den Komponenten gemeinsamen interaktiven Benutzerkomponente. Die interaktive Benutzerkomponente verteilt die Ticks über die Steuerschnittstellen an die Komponenten und realisiert so die enge Kopplung der Komponenten. Bei einer Tickrate von 600 Hz ist es nur innerhalb eines Systems sinnvoll jeden Tick als Ereignis zu verteilen. Anstatt nun zu jedem Tick ein Ereignis zu verteilen werden bei der Tickverteilung Tickwerte mit fester Rate verteilt, wobei diese Rate in die Größenordnung der Schritte fällt. Um die Übertragungsraten gemäß den Anforderungen an der Steuerschnittstelle klein zu halten, wird zu jedem Schritt nur ein Teil (1 Byte) des Tickwertes übertragen. Begonnen wird mit der Übertragung des höchstwertigen Bytes, so daß im letzten Schritt einer Tickerübertragung mit dem letzten Byte der genaue aktuelle Tickwert übertragen wird. Ähnliche Verfahren werden bereits bei anderen Synchronisationsverfahren verwendet. Eine genaue Beschreibung sowie die Kodierung für die verschachtelte Übertragung von Tickwerten und Schnittstellen-Aufrufen wird in [Hesme93] vorgestellt.

6.3 weitere Entwicklung

Zur Realisierung verteilter multimedialer Anwendungen, muß man die einzelnen verteilten Komponenten bestimmen und ihre Funktion beschreiben. Die Komponenten tauschen untereinander Steuerungsinformationen und Multimediadaten aus. Diese Daten und das beim Austausch verwendete Protokoll sollten allgemein standardisiert sein, um den Zusammenschluß heterogener Systeme zu ermöglichen.

In der vorliegenden Arbeit wurde gezeigt, wie sowohl die Daten als auch das Zusammenspiel der Komponenten festgelegt werden können. Obwohl alle Geräteklassen und Gerätefunktionen sowie verschiedene Werkzeuge entwickelt wurden, und das vorgestellte Modell die gesamte Entwicklung verteilter multimedialer Anwendungen unterstützt, ist dieses große Gebiet noch lange nicht erschöpfend behandelt. Eine Erweiterung der Managementschnittstellen und die Realisierung von komplexen Werkzeugen sind die vordringlichsten Aufgaben. Damit entsteht ein mächtiges Entwicklungswerkzeug für Multimediaanwendungen.

Eine weitere Aufgabe ist die genauere Untersuchung der Nebenbedingungen, die zur Unterscheidung der Funktionen der Typen f_1 bis f_5 führten. Aus diesen Untersuchungen sowie aus den Ergebnissen der Ticker– und Schrittgeber–Realisierung lassen sich dann genauer spezifizierte Anforderungen an die Betriebs– oder Kommunikations–Systeme ableiten.

Literatur

Alireza94 Ameneh Alireza
Werkzeuge in verteilten Multimedia–Systemen,
Beschreibung der Verarbeitungsfunktionen zwischen Multimediakomponenten
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität, Frankfurt am Main 1994

Allen83 James F. Allen
Maintaining Knowledge about Temporal intervals
Communications of the ACM,No. 11, November 1983

AlSchTh94 M. Altenhofen, J. Schaper, S. Thomas
The BERKOM Multimedia Teleservices
Second International Workshop on Advanced Teleservices and High-Speed Communication Architectures (IWACA) 94, Heidelberg, Germany, 26.-28. September 1994
Ralf Steinmetz (Ed.), Lecture Notes in Computer Science 868, Springer

AndCh91 David P. Anderson, Pamela Chan
Toolkit Support for Multiuser Audio/Video Applications
Second International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer

AndHerr91 David P. Anderson, Ralf G. Herrtwich
Internet Communication with End–to–End Performance Guarrantees
in „Telekommunikation und multimediale Anwendungen der Informatik", J. Encarnacao (Hrsg.), Proceed. of the GI'91, Darmstadt Oktober 1991, Informatik Fachberichte 293, Springer (1991)

AraClay90 Mauricio E. Arango, R. Clayton
An Infrastructure to Support Multimedia Applications
Internal Report, Bellcore 1990

ArBaGo92 Mauricio E. Arango, Peter C. Bates, Gita Gopal, et al.
TOURING MACHINE: A Software Infrastructure to Support Multimedia Communications
Multimedia'92, 4th IEEE COMSoc International Workshop on Multimedia Communications, Monterey California 1992

BaHeRo95 Ingo Bart, Tobias Helbig, Kurt Rothermel
Implementierung multimedialer Systemdienste in CNEMA
Kommunikation in Verteilten Systemen (KIVS) 95,
GI/ITG-Fachtagung, Chemnitz-Zwickau, 22.-24. Februar 1995

Baker92 Rusti Baker, Alan Downing, Kate Finn, Earl Rennison,Doo Hyun, David Kim, Young Hwan Lim
Multimedia Processing Model for a Distributed Multimedia I/O System
Third Intern. Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, Nov 1992, Lecture Notes in Computer Science 712, Springer Verlag

BarDer93 Ingo Barth, Gabriel Dermler, Tobias Helbig, Kurt Rothermel, Frank Sembach, Thomas Wahl
Cinema: Eine konfigurierbare, integrierte Multimedia–Architektur
FOKUS, Praxis Information und Kommunikation, Band 5, Verteilte Multimedia–Systeme
Hrsg: Wolfgang Effelsberg und Kurt Rothermel, Verlag S·G·Saur 1993

Bast93 Claudia Bastian
Integration der Zeit in das Audio/Video Datemodell
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität, Frankfurt am Main, 1993

BatBac94 John Bates, Jean Bacon
A Developement Platform for Multimedia Applications in a distributed ATM Network Environment
ICMCS'94, 14.-19. May, Boston, Massachusets 1994

BateSega90 Peter C. Bates, Mark Segal, Mauricio E. Arango
Touring Machine: A Video Telecommunications Software Testbed
First Intern. Workshop on Network and Operating System Support for Digital Audio and Video, Berkeley 1990

BERKOM91 BERKOM Reference Model: Lower Layers
Version 2.0 — February 1991

BERKOM92 BERKOM Reference Modell: Application–Oriented Layers
Version 4.0.1 — December 1992

BERMMC93 Michael Altenhofen, Jürgen Dittrich, Rainer Hammerschmidt, Thomas Käppner, Carsten Kruschel, Ansgar Kückes, Thomas Steinig
The BERKOM Multimedia Collaboration Service
Proccedings ACM Multimedia 1993, Anaheim, California, August 1993

BlaCou92 Gordon S. Blair, Goffrey Coulson
An Integrated platform and Computational Model for Open Distributed Multimedia Applications
Third Intern. Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, Nov 1992, Lecture Notes in Computer Science 712, Springer Verlag

BlaCoDa94 Gordon S. Blair, Goffrey Coulson, Nigel Davies
System Support For Multimedia Applications: An Assessment of the State of the Art
Lancaster University UK, Report Number MPG–93–29
in "Information and Software Technology" Special Issue on Multimedia, 1994

Blako93 Gerold Blakowski
Entwicklungs– und Laufzeitunterstützung für verteilte multimediale Anwendungen
Reihe Informatik, Verlag Shaker, 1993

BlHüLa91 Gerolg Blakowski, Jens Hübel, Ulrike Langrehr
Tools for Specifying and Executing Synchronized Multimedia Presentations
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

Blum93 C. Blum, M. Festini, A. Fischer, G. R. Hofmann, B. Hox, P. Kaufmann, L. A. Kotsch, A. Liebing, H. Lindner, H. Mauersberg, E. Moeller, W. Müller, B. Neidecker–Lutz, M. Niemöller, J. Rückert, A. Scheller, G. Schürmann, S. Thomas, I. Varsek
The BERKOM, Multimedia–Mail Teleservice, DeTeBerkom: Internal Report, March 1993

Börger92 Martin Börger
Modellierung von Anwendungen und Geräten in multimedialen Systemen
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität Frankfurt am Main, 1992

Brand90 S. Brand
Media Lab: Die Erfindung der Zukunft
Rowohlt Taschenbuch Verlag 1990

Braun92 Hartmut Braun
Abstraktion von Multimedia–Daten
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität Frankfurt am Main, 1992

Burg93 Jeff Burger
The desktop multimedia bible
Addison Weseley 1993

ButHeLu91 B. Butcher, L. Henckel, T. Luckenbach
Die Kommunikationsplattform BERKOM für multimediale Anwendungen
Informatik Spektrum, Vol.14, No.5, Oktober 1991

CaAuHa95 Andrew Campbell, Cristina Aurrecoechea, Linda Hauwn
Architectural Perspectives on QoS Management in Distributed Multimedia Systems
PROMS 95, Second Wokshop on Protocols for Multimedia Systems,
Mozart on Multimedia Highways, Salzburg, Austria, 9.-12. October 1995

CACM89 Communications of the ACM
Special Section in Interactive Technologie
Vol. 32, No. 7 , July 1989

CaCoGa95 Andrew Campbell, Geoffrey Coulson, Francisco García, David Hutchison
Integrated Quality of Service for Multimedia Communications
ESPRIT Reserarch Report, Project 5341, OSI 95, Volume 1
The OSI 95 Transport Service with Multimedia Support

CaCoHu94 Andrew Campbell, Geoffrey Coulson, David Hutchinson
A Quality of Service Architecture
ACM Computer Communications Review, April 1994

CarDeWi89 M. J. Carey, D. J. DeWitt, J. E. Richardson, E.J. Shekita
Storage Managements for Objects in EXODUS
in *Object-Oriented Concepts, databases, and Applications*
W. Kim, F.H. Lochovsky Eds., Addison Wesley 1989

Conkli87 J. Conklin
Hypertext: A survey and Introduktion
IEEE Computer, September 1987

CouBaRo93 Geoffry Coulson, Gordon S. Blair, Philippe Robin, Doug Shepherd
Extending the Chorus Micro–Kernel to Support Continuous Media Applications
fourth Inter. Workshop on Network and Operating System Support for Digital Audio and Video,
Lancaster House, Lancaster UK, Nov 1993

CoulBla93 Geoffrey Coulson, Gordon S. Blair, Nigel Davies, N. Williams
Extensions to ANSA for Multimedia Computing.
Computer Networks and ISDN Systems, 25, 1993

Coulson93 Geoffry Coulson
Multimedia Application Support in Open Distributed Systems
Computing Department Lancaster University,
Thesis submitted for the degree of Doctor of Philosophy, April 1993

Dann92 Roger B. Dannemberg, Tom Neuendorffer, Joseph M. Newcomer, Dean Rubine
Tactus: Toolkit–Level Support for Synchronized Interactive Multimedia
Third Intern. Workshop on Network and Operating System Support for Digital Audio and Video,
San Diego, Nov 1992, Lecture Notes in Computer Science 712, Springer Verlag

Dömel95 Peter Dömel
Eine Entwurfs– und Laufzeitumgebung für konfigurierbare verteilte Anwendungen
Kommunikation in Verteilten Systemen (KIVS) 95,
GI/ITG-Fachtagung, Chemnitz-Zwickau, 22.-24. Februar 1995

DürNes90 M. Dürr, R. Neske
Hypertext und Datenbanken: Gegensatz oder Symbiose?
in [GloStr90] pp 149-161

Eventor94 Seongbae Eun et alii
Eventor: an authoring system for interactive multimedia applications
Multimedia Systems, Vo.2, No.3, 1994, pp. 129-140, ACM/Springer

Ferrari91 Domenico Ferrari, The Tenet Group
Design and Applications of a Delay Jitter Control Scheme for Packet–Switching Internetworks
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

Feldh91 Magdalena Feldhoffer
Communicationsupport for Distributed Applications
International Workshop on ODP, Berlin, Oct. 1991

Feldh93 Magdalena Feldhoffer
Spezifikation Konfiguration und Implementation anwendungsorientierter Kommunikationsdienste
Dissertation, Fachbereich Informatik, J. W. Goethe Universität
Frankfurt am Main, 1993

Fisher92 Tom Fisher
Real Time Scheduling Support in ULTRIX 4.2 for Multimedia Communication
Third Inter. Workshop on Network and Operating System Support for Digital Audio and Video,
San Diego, Nov 1992, Lecture Notes in Computer Science 712, Springer Verlag

FreyIn91 Stefan Frey, Daniel P. Ingold
An Application Framework for Multimedia Communication
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

FrBöBr92 J. Christian Fritzsche, Martin Börger, Hartmut Braun
An Abstract View on Multimedia Devices in Distributed Systems,
Computer Networks, Architecture and Applications
IFIP TC6 Working conference, NETWORKS ’92, Trivandrum, India, 28-29 October 1992,
S. V. Raghavan, G. v. Bochmann, G. Pujolle (Editors)
IFIP Trans. C: Communication Systems Volume C-13, North-Holland

Fritzsche95 J. Christian Fritzsche
Continuous Media described by Time Dependent Data
ACM/Springer Multimedia Systems Journal special issue on Multimedia Databases,
No. 5/6, Vol. 3, November 1995

Fritzsche96 J. Christian Fritzsche
Multimedia Building Blocks for Distributed Applications
MMSD-96, Intern. Workshop on Multimedia Software Development, Berlin 25/26 März 1996

FunPon94 Chi–Leung Fung, Man–Chi Pong
MOCS: an Object-Oriented Programming Model for Multimedia Object Communication and Synchronisation
14th International Conference on Distributed Computing Systems, Pozan, Poland, June, 1994

Gibbs91 Simon Gibbs, Christian Breiteneder, Laurent Dami, Vicki de Mey, Dennis Tsichrityzis
A Programming Environment for Multimedia Applications
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

Gibbs92 Simon Gibbs
Application Construction Component Design in an Object–Oriented Multimedia Framework
Third Inter. Workshop on Network and Operating System Support for Digital Audio and Video,
San Diego, Nov 1992, Lecture Notes in Computer Science 712, Springer Verlag

Gibbs94 Simon Gibbs, Christian Breiteneder, Dennis Tsichritzis
Data Modeling of Time–Based Media
to appear in sigmod 94, available via ftp from cui.unige.ch

GloStr90 P.Gloor, N. Streitz
Hypertext and Hypermedia
Informatik Fachberichte 249, Springer Verlag 1990

Gold91 Charles F. Goldfarb
HyTime: A Standard for Structural Hypermedia Exchange
IEEE Computer, Vol.24, No.8, August 1991

Gold91a Charles F. Goldfarb
The SGML Handbook
Claredadon Press, Oxford 1991

GuAL93 Thomas Gutekunst, Thomas Schmidt, Günter Schulze, Jean Schweitzer, Michael Weber
A Distributed Multimedia Joint Viewing and Tele–Operation Service for
Heterogenous Workstation Environments
FOKUS, Praxis Information und Kommunikation, Band 5, Verteilte Multimedia–Systeme
Hrsg: Wolfgang Effelsberg und Kurt Rothermel, Verlag S·G·Saur 1993

Harney91 Kevin Harney, Mike Keith, Gary Lavelle, Lawrence D. Rayn, Daniel J. Stark
The i750 Video processor: ATotal Multimedia Solution
Communications of the ACM, Vol. 34, No.4, April 1991

HehSaSt90 D. B.Hehmann, M. G. Salmony, H. J. Stüttgen
Transport Services for Multimedia Applications on Broadband Networks
Computer Communications, Vol13. No.4, May 1990

Helbig94 Tobias Helbig
Development and Control of Distributed Multimedia Applications
4th Open Workshop on High–Speed Networks,Brest (France), Sept. 1994

Herrtw90 Ralf Guido Herrtwich
Timed Data Treams in Continuous–Media Systems
Int. Computer Science Institute, Berkeley, California, Tr-90-017, 1990
deutsch erschienen als:
Zeitkritische Datenströme in verteilten Multimedia–Systemen
Proc. GI/ITG-Fachtagung Kommunikation in verteilten Systemen Feb. ’91
Informatik Fachberichte 267, Springer Verlag 1991

Herrtw91 Ralf Guido Herrtwich
Time Capsules: An Abstraction for Access to Continuous Media Data
The Journal of Real–Time Systems, Kluwer Academic Publishers, 1991

Hesme93 Frank Hesmert
Implementation multimedialer Präsentationskomponenten im Ticker/Schrittgeber–Modell
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität Frankfurt am Main, Juni 1993

HoCoLa89 M. Hofman, R. Cordes, H. Langendörfer
Hypertext/Hypermedia
Informatikspektrum, Bd.12 Nr. 4, August 1989

Hodges89 Matthew E. Hodges, Russel M. Sasnett, Mark S. Ackermann
Athena Muse: A Construction Set for Multi–Media Applications
IEEE Software, Jan. 1989

Hoep91a Petra Hoepner
Synchronisation und Präsentation von Multimedia–Objekten –Modell und Beispiele
in „Telekommunikation und multimediale Anwendungen der Informatik"
J. Encarnacao (Hrsg.), Proceed. of the GI'91, Darmstadt Oktober 1991
Informatik Fachberichte 293, Springer Verlag (1991)

Hoep91b Petra Hoepner
Presentation Scheduling of Multimedia Objects and Its Impact on Network and Operating System Support
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

Hoffert92 E. Hoffert et al.
QuickTime: An Extensible Standard for digital Multimedia
IEEE Computer Conference (CompCon'92), Februar 1992

HyTime91 ISO/IEC JTC 1/SC 18
Revised Text of CD 10744, Information Technology -
Hypermedia/Time–based Structuring Language (HyTime), Sept. 1991

H.261 International Telecommunication Union,
the International Telegraph and Telephone Conultative Committee
Line Transmission on non–Telephone Signals: Video Codec for Audiovisual Services at p × 64 kbits/s
CCITT Recommendation H.261, Geneva, 1990

ISO8613 ISO; Information Processing
Office Document Architecture and Interchange Format
ISO Genf 1989

ISO8824 ISO; Information Processing Systems — Open System Interconnection
Specification of Abstract Syntax Notation One (ASN.1), 1987

IMA92 Interactive Multimedia Association
Arhitecture Reference Model
Revision 3.1 November 6, 1992

IMA94 IMA Recommended Practice
Multimedia System Services
Second Draft, 23. September 1994

IMAL Integrated Media Architecture Laboratory, L. F. Ludwig, D. F. Dum
Laboratory for Emulation and Study of Integrated and Coordinated
Media Communication, Frontiers in Computer Technologie,
Proceedings of the ACM SIGCOMM '87, Workshop Aug. 11-13, 1987

JaSrNe95 Paul W. Jardetzky, Cormal J. Sreenan, Roger M. Needham
Storage and synchronizatoion for distributed continuous media
Multimedia Systems, Vo.3, No.4, 1995, pp. 151-161, ACM/Springer

JPEG92 ISO IEC JTC 1; Information Technology —
Digital Compression and Coding of Continuous Tone still Images
International Standard ISO/IEC IS 10918, 1992

KaHeSch95 Thomas Käppner, F. Henkel, A. Schröder, M. Müller
Eine verteilte Entwicklungs- und Laufzeitumgebung für multimediale Anwendungen
Kommunikation in Verteilten Systemen (KIVS) 95,
GI/ITG-Fachtagung, Chemnitz-Zwickau, 22.-24. Februar 1995

KaMeMe94 Rolf Kachenhoff, Detlef Merten, Klaus Meyer–Wegener
Moss as a Multimedia-Object Server
High-Speed Communication Architectures (IWACA) 94
Heidelberg, Germany, 26.-28. September 1994
Ralf Steinmetz (Ed.), Lecture Notes in Computer Science 868, Springer

Kaul95 Manfred Kaul
Distributed Multimedia Applications and Information on Demand Services
PROMS 95, Second Wokshop on Protocols for Multimedia Systems,
Mozart on Multimedia Highways, Salzburg, Austria, 9.-12. October 1995

KLLMW93 Thomas Kirsche, Richard Lenz, Horst Lührsen, Klaus Meyer–Wegener, Hartmut Wedekind
CoDraft —Eine Verteilte Architektur zur Unterstützung von Gruppenarbeit durch Multimediale Objekte
FOKUS, Praxis Information und Kommunikation, Band 5, Verteilte Multimedia–Systeme
Hrsg: Wolfgang Effelsberg und Kurt Rothermel, Verlag S·G·Saur 1993

Konst93 Therese Konstantinov
Komponenten verteilter Multimedia–Anwendungen
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität Frankfurt am Main, 1993

LeGall91 Didier LeGall
MPEG: A Vidao Compression Standard for Multimedia Applications
Communicatioons of the ACM, Vol. 34, No.4 April 1991

LehLind89 T. J. Lehman, B. G. Lindsay
The Starburst Long Field Manager
Interantional Conference on Very Large Databases (VLDB), 1989

LeydTeu91 Peter Leydekkers, Bertjan Teunissen
Synchronisation of Multimedia Data Streams in Open Distributed Environments
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

Lipp91 S. Lippman
C++ Einführung und Leitfaden
Addison Wesley, 1991

Little90 Thomas D. C. Little, Arif Ghafoor
Synchronization and Storage Models for Multimedia Objects
IEEE Jour. o. Selected Areas in Communications, Vol.8, No. 3, April 1990

LittlKao92 Thomas D. C. Little, F. Kao
An Intermedia Skew Control System for Multimedia Data Presentation
Third Intern. Workshop on Network and Operating System Support for Digital Audio and Video,
San Diego, Nov 1992, Lecture Notes in Computer Science 712, Springer Verlag

Littm91 D. Littman, T. Morgan
Quicktime: It’s about Time
Mac World, Aug. 1991

Mattern91 Friedemann Mattern
Über die relativistische Struktur logischer Zeit in verteilten Systemen
Universität des Saarlandes Saarbücken, Fachbereich Informatik, Interner Bericht A05/91

MeBr89 Chris Meyer, Evan Brooks
MIDI Time Code Specification
ftp von /pub von ucds.edu 1989

MEDUSA94 Stuart Wray, Tim Glavert, Andy Hopper
The Medusa Application Environment
International Conference on Multimedia Computing and Systems
ICMCS 94, 14.-19. May, Boston, Massachusets, 1994

Meißner91 Klaus Meißner
Architectural Aspects of Multimedia CD-I Integration in UNIX/X–Windows Workstaions
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

MeyGib93 Vicki de Mey, Simon Gibbs
A Multimedia Component Kit
Proccedings ACM Multimedia 1993, Anaheim, California, August 1993

MHEG90 Multimedia and Hypermedia information coding Expert Group
ISO/IEC JTC1/SC2/WG12.
MHEG Working Document "S", Version 1, June 1990

Milner83 R. Milner
Calculi for Synchrony and Asynchrony
Theoretical Computer Science 25: 267-310, 1993

MIME92 Eduard Vienlmetti
This is an experimental multi-media mail message
Article 183 of comp.mail.multi-media, 16. Feb. 92

MiViFe95 C. Miguel, L. Vidaller, A. Fernández
Extended LOTOS
ESPRIT Research Reports, The OSI 95 Transport Service with Multimedia Support, Project 5341, OSI95, Vol. 1

MPEG93 ISO IEC JTC1; Information Technology
Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 MBits/s
International Standard ISO/IEC IS 11172, 1993

NiHaNo93 J. Nieh, J. G. Hanko, D. Northcutt G. A. Wall
SVR4 UNIX Scheduler Unacceptable for Multimedia Applications
Fourth Inter. Workshop on Network and Operating System Support for Digital Audio and Video,
Lancaster House, Lancaster UK, Nov 1993

Niels90 J. Nielsen
Hypertext and Hypermedia
Academic Press 1990

NRSV90 X. Nicollin, J. L. Richier, J. Sifakis, J. Voilon
ATP: An algebra for timed processes
IFIP Working Conference on Programming Concepts and Methods,
M. Broy, C.B. Jones eds., The Netherlands 1990, IFIP Elsevier Science

OikaTok93 S. Oikawa, H. Tokuda
User–Level Real–Time Threads
Fourth Inter. Workshop on Network and Operating System Support for Digital Audio and Video,
Lancaster House, Lancaster UK, Nov 1993

Ortiz91 Guillermo A. Ortiz
Quicktime 1.0: "You Oughta Be In Pictures"
develop — The Apple Technical Journal, 1991

PerLitt95 M. JH. Pérez-Luque, T. D. C. Little
A Temporal Reference Framework for Multimedia Synchronization
IEEE Workshop on Multimedia Synchronization
in conjunction with the
Proc. International Conference in Multimedia Computing and Systems
ICMCS 95, Washington D.C. May 1995

PraRag94 B. Prabhakaran, S. V. Raghavan
Synchronisation-Models for Multimedia Presentation with User Participation
Multimedia Systems, Vo.2, No.2, 1994, pp. 53-62, ACM/Springer

PhSo88 N. V. Philips, Sony Corporation
CD–I Full Functional Specification
Green Book, N. V. Philips and Sony Corporation Documentation

Popescu91 Radu Popescu–Zeletin, V. Tschammer, Michael Tschichholz
Y's Distributed Application Plattform
Computer Communications, 1991

Rengar92 T. K. Rengarajan
Rdb/VMS Support for Multi–Media Databases
IEEE CS Office Automation Symposium, Gaithersberg, April 1992

Ripley89 G. D. Ripley
DVI — A Digital Multimedia Technology
Communications of the ACM, Vol.32,No.7, July 1989

RoBaHe94 Kurt Rothermel, Ingo Barth, Tobias Helbig
CINEMA – An Architekture for Configurable Distributed Multimedia Applications
Fakultätsbericht 3/1994, April 1994, Fakultät für Informatik, Universität Stuttgart

RotHelb94
RotHelb96 Kurt Rothermel, Tobias Helbig
Clock Hierarchies: An Abstraction for Grouping an Controling Media Streams
Fakultätsbericht 2/1994, April 1994, Fakultät für Informatik, Universität Stuttgart
überarbeitet in:
IEEE Journal on Selected Areas in Communications -
Synchronization Issues in Multimedia communications, 1996

RustBCD91 Lillian Ruston, Gordon S. Blair, Goffrey Coulson, Nigel Davies
Integrating Computing and Telecommunications: A Tale of Two Architectures
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

SemRo93 Frank Sembach, Kurt Rothermel
TEATIME: Gemeinsamer Arbeitsbereich für kooperativ bearbeitete multimediale Objekte
FOKUS, Praxis Information und Kommunikation, Band 5, Verteilte Multimedia–Systeme
Hrsg: Wolfgang Effelsberg und Kurt Rothermel, Verlag S·G·Saur 1993

Skritek88 Skritek, Paul
Handbuch der Audio–Schaltungstechnik
Franzis Verlag 1988

SloKr89 Morris Sloman, Jeff Kramer
Verteilte Systeme und Rechnernetzt
Hanser / Prentice-Hall 1989

Stalli88 W. Stallings
Data and Computer Communication
Macmillan Publishing Company, 1988

SteHaHo92 Jean–Bernard Stefani, Laurent Hazard, Francois Horn
A Computatioinal Model for Ditributed Multimedia Applications Based on
a Synchronous Programming Language
Computer Communications, Vol.15, No.2, March 1992

SteiFri91 Ralf Steinmetz, J. Christian Fritzsche
Abstractions for Continuous–Media Programming
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

SteiHe91 Ralf Steinmetz, Ralf Guido Herrtwich
Integrierte verteilte Multimedia Systeme
Informatikspektrum(1991) Vol. 13, Nr.5 Oktober 1991, Springer Verlag

SteiMey92 Ralf Steinmetz, Thomas Meyer
Modelling Distributed Multimedia Applications
IEEE International Workshop on Advanced Communications and
Applications for High Speed Networks, München, März 1992

Stein90 Ralf Steinmetz: Das aktuelle Schlagwort: Multimedia-Systeme
Informatik-Spektrum (1990) 13: 280-282, Springer Verlag

Stein90a Ralf Steinmetz
Synchronization Properties in Multimedia Systems
IEEE Jour. o. Selected Areas in Communications, Vol.8,Nol. 3, April 1990

Stein93 Ralf Steinmetz,
Compression Technics in Multimedia Systems: A Survey
IBM ENC, Internal Report 43.9307, 1993

Stein93a Ralf Steinmetz,
Multimedia–Technologie, Einführung und Grundlagen
Springer Verlag 1993

Stein94 Ralf Steinmetz
Human Perception of Media Synchronization
based on changed and enhanced IBM Technical Report No. 439310, 1993
Ralf Steinmetz, Clemens Engler
Human Perception of Media Synchronization
submitted to IEEE JSAC special issue on Multimedia Synchronization

SteiSir91 Daniel Steinberg, Josh Sirota
Components: A Multimedia Programming Model
Second Inter. Workshop on Network and Operating System Support for Digital Audio and Video
Heidelberg, Nov 1991, Lecture Notes in Computer Science 614, Springer Verlag

Stüttgen95 Stüttgen
Network Evolution and Multimedia Communication
IEEE Multimedia, Vol. 2, No. 3, Fall 95, pp 42-59

SülCor90 K. Süllow, R. Cordes
Einbeziehung von Hypermediatechniken in die multimediale Kommunikation
in [GloStr90] pp 139-143

Tenn94 David L. Tennenhouse et alii
A Softwareoriented Approach to the Design of
Media Processing Environments
International Conference on Multimedia Computing and Systems
ICMCS 94, 14.-19. May, Boston, Massachusets, 1994

Tenn95 David L. Tennenhouse et alii
The View Station: A software-intensive approach to media processing and distribution
Multimedia Systems, Vo.3, No.3, 1995, pp.104-115, ACM/Springer

Topo90 Claudio Topolcic
Experimental Internet Stream Protocol, Version 2 (ST-II)
RFC 1190, Oct. 1990

Tsich91 Dennis Tsichritzis, Simon Gibbs, Laurant Dami
Active Media Object Composition, Dennis Tsichritzis (Hrsg.), Genève, Juni 1991
Université de Genève, Centre Universitaire d'Informatique

Unge92 Gisela Ungeheuer
Aspekte der Synchronisation in multimedialen Systemen,
Diplomarbeit, Fachbereich Informatik, J. W. Goethe Universität Frankfurt am Main, 1992

VoKeBo95 Andreas Vogel, Brigitte Kerhervé, Gregor von Bachmann, Jan Gecsei
Distributed Multimedia and QoS: A Survey
IEEE MultiMedia, Vol. 2, No. 2, Summer 1995, pp 10-19

WahlRo94 Thomas Wahl, Kurt Rothermel
Representing Time in Multimedia Systems
Proc. International Conference in Multimedia Computing and Systems
ICMCS 94, CS Press, Los Alamitos, California 1994, pp. 538-543

WahlRo95 Thomas Wahl, Kurt Rothermel
Representing Time in Multimedia Systems
IEEE Workshop on Multimedia Synchronization
in conjunction with the
Proc. International Conference in Multimedia Computing and Systems
ICMCS 95, Washington D.C. May 1995

WiBla94 Neil Williams, Gordon S. Blair
Distributed multimedia applications: A review
Computer Communications, Vol.17, No.2 February 1994

WiRoWa95 Stefan Wirag, Kurt Rothermel, Thomas Wahl
Modelling Interaction in HyTime
Kommunikation in Verteilten Systemen (KIVS) 95,
GI/ITG-Fachtagung, Chemnitz-Zwickau, 22.-24. Februar 1995

WoeKi87 D. Woelk, W.Kim
Multimedia Information Management in an Object–Oriented Database System
13th VLDB Conference, Brighton 1987

WoQaGh94 Miae Woo, Naveed U. Qazi, Arif Ghafoor
A Synchronisation Framework for Communication of Pre–orchestrated Muiltimedia Information
IEEE Network, January/February 1994

Zimm92 Martin Zimmermann
An Integrated Approach for the Construction of Distributed Applications
IFIP — Open Distributed Processing, J. de Meer, V. Heymer and R. Roth (Editors)
Elsevier Science Publishers B.V. (North–Holland) 1992, IFIP

Zimm92a Martin Zimmermann
Specification and Implementation of Cooperation Paradigms for Distributed Applicatoins
Open Forum, Utrecht, Netherlands, 25-27 Nov. 1992

Zimm93 Martin Zimmermann
Konstruktion und Management verteilter Anwendungen
Dissertation, Fachbereich Informatik, J. W. Goethe Universität Frankfurt am Main 1993

Anhang A
Schnittstellen und Kommunikationskontexte

An dieser Stelle sind alle Schnittstellen– Kommunikationskontextdefinitionen aus der vorliegenden Arbeit wiederholt, sowie Beispiele für spezielle Kommunikationsanforderungen eingefügt.

A.1 management

```
cont_media_management INTERFACE
        CONSUMER INVOKES
                init;
                release;
        COOPERATION PROTOCOL
                init < release;
```

A.2 data

```
 continuous_data
INTERFACE
        CONSUMER INVOKES
        setup;
        receive_seq_parameters;
        receive_seq_elemet;

        COOPERATION PROTOCOL
        (setup+ < receive_seq_parameters+ < receive_seq_elemet°)°
```

A.3 control

```
any_source_control
INTERFACE

any_sink_control
INTERFACE

step_control
INTERFACE
        CONSUMER INVOKES
        set_rate;
        get_rate;
        set_start_tick;
        get_start_tick;

        get_tick;
```

```
        COOPERATION PROTOCOL
        set_parameters = (set_rate; set_start_tick)
        get_parameters = (get_rate; get_start_tick)

        get_tick° |(set_parameters; get_parameters)°
perception_control
INTERFACE DERIVED FROM step_control, any_source_control
        CONSUMER INVOKES
        send_sequence_parameters;

        COOPERATION PROTOCOL
        set_parameters = (set_rate; set_start_tick)
        get_parameters = (get_rate; get_start_tick)
        get_tick° |
        ((send_sequence_parameters+;
         (set_parameters; get_parameters))°)
perception_control_multistep
INTERFACE DERIVED FROM perception_control
        CONSUMER INVOKES
        set_data_steps;   get_data_steps;
        set_duration;     get_duration;

        COOPERATION PROTOCOL
        set_parameters =
        (set_data_steps; set_duration; set_rate; set_start_tick)
        get_parameters =
        (get_data_steps; get_duration; get_rate; get_start_tick)
        get_tick° |
        ((send_sequence_parameters+;
         (set_parameters; get_parameters))°)
presentation_control
INTERFACE DERIVED FROM step_control, any_sink_control
        CONSUMER INVOKES
        get_seq_s_rate;
        get_seq_t_rate;
        get_seq_start_step;
        get_seq_duration;

        COOPERATION PROTOCOL
        set_parameters = (set_rate; set_start_tick)
        get_parameters = (get_rate; get_start_tick)
        get_seq_parameters =
```

```
        (get_seq_s_rate; get_seq_t_rate;
         get_seq_start_step; get_seq_duration)

        get_tick° |
        (set_parameters; get_parameters; get_seq_parameters)°

simple_write_control
INTERFACE DERIVED FROM any_sink_control
        CONSUMER INVOKES
        create;
        concat;
        destroy;

simple_read_control
INTERFACE DERIVED FROM any_source_control
        CONSUMER INVOKES
        extract;
        send_seq;

simple_read_write_control
INTERFACE DERIVED FROM simple_write_control;simple_read_control
        COOPERATION PROTOCOL
        (create;concat)+ < (extract; send_seq)+

seq_write_control
INTERFACE DERIVED FROM simple_write_control
        CONSUMER INVOKES
        set_S_rate;
        set_T_rate;
        set_start_step;
        set_duration;
        add;
        del_elem;

        set_el_value;
        set_el_start;
        set_el_duration;
        COOPERATION PROTOCOL

seq_read_control
INTERFACE DERIVED FROM simple_read_control
        CONSUMER INVOKES
        get_S_rate;
        get_T_rate;
        get_start_step;
        get_duration;

        first;
        next;
        prev;
        set_pos;
        COOPERATION PROTOCOL
```

Verarbeitungsfunktionen

```
control_f
INTERFACE
        CONSUMER INVOKES
        set_parameter;
```

A.4 Kommunikationsanforderungen

- 600 HZ Ticker mit 4 byte Integer ⇒ 19200 bit/s
  dazu
  1 Farbbild mit 512×512 Punkten und je 1byte Farbinformation in 1,5 Sekunden ⇒ 1398102 bit/s
  Es ergibt sich eine Anforderung von 1417301 bit/s oder 1,5 Mbit/s
- 600 HZ Ticker mit 4 byte Integer ⇒ bit/s
  dazu
  ein Sprachkanal in Telephonqualität : 64 Kbit/s.
  Es ergibt sich eine Anforderung von 85 Kbit/s

•

```
   ContMedFunctionControlContext
COMMUNICATIONCONTEXT
 TRANSPORT(ARBITRARY)
 SINGLE-PARTY
 CONNECTION-MANAGEMENT(
   CONNECTIONORIENTED(
        ESTABLISH-RESPONDER
        RELEASE-RESPONDER
        REESTABLISHING))
 INTERACTION (SERVER
   OPERATIONS(
             ASYNCHRONOUS
             REPLY
             RETRANSMITTING
             CANCEL
             NO_LINKED_OPERATION
             NO_ENCRYPTION
             NO_TOKENSUPPORT))
::=ContMedFunctionControlContext
```

```
   UserControlContext
COMMUNICATIONCONTEXT
 TRANSPORT (ARBITRARY)
 SINGLE-PARTY
 CONNECTION-MANAGEMENT(
   CONNECTIONORIENTED(
        ESTABLISH-INITIATOR
        RELEASE-INITIATOR
        REESTABLISHING))
 INTERACTION (CLIENT
        OPERATIONS(
             ASYNCHRONOUS
             REPLY
             RETRANSMITTING
             CANCEL
             NO_LINKED_OPERATION
             NO_ENCRYPTION
             NO_TOKENSUPPORT))
::=UserControlContext
```

TightMedFunctionControlContext Template
COMMUNICATIONCONTEXT

TRANSPORT(
DOUPLEX,
THROUGPUT: *<23000> Bit/sec*,
TRANSMITDELAY: *<100> msec,*
DELAYJITTER: *<0.8> msec*
TRANSFERFAILUREPROB: *<0> %*
)
SINGLE-PARTY
CONNECTION-MANAGEMENT(
CONNECTIONORIENTED(
ESTABLISH-RESPONDER
RELEASE-RESPONDER
REESTABLISHING)
)
INTERACTION (**SERVER**
OPERATIONS(
ASYNCHRONOUS
REPLY
RETRANSMITTING
CANCEL
NO_LINKED_OPERATION
NO_ENCRYPTION
NO_TOKENSUPPORT)
)
::=TightMedFunctionControlContext

TightUserControlContextTemplate
COMMUNICATIONCONTEXT

TRANSPORT (
DOUPLEX,
THROUGPUT: *<23000> Bit/sec*,
TRANSMITDELAY: *<100> msec,*
DELAYJITTER: *<0.8> msec*
TRANSFERFAILUREPROB: *<0> %*
)
SINGLE-PARTY
CONNECTION-MANAGEMENT(
CONNECTIONORIENTED(
ESTABLISH-INITIATOR
RELEASE-INITIATOR
REESTABLISHING)
)
INTERACTION (**CLIENT**
OPERATIONS(
ASYNCHRONOUS
REPLY
RETRANSMITTING
CANCEL
NO_LINKED_OPERATION
NO_ENCRYPTION
NO_TOKENSUPPORT)
)
::=TightUserControlContext

StoreFunctionControlContextTemplate
COMMUNICATIONCONTEXT

TRANSPORT(
DOUPLEX,
THROUGPUT: *<64000> Bit/sec*,
TRANSMITDELAY: *<100> msec,*
DELAYJITTER: *<10> msec*
TRANSFERFAILUREPROB: *<0> %*
)
SINGLE-PARTY
CONNECTION-MANAGEMENT(
CONNECTIONORIENTED(
ESTABLISH-RESPONDER
RELEASE-RESPONDER
REESTABLISHING)
)
INTERACTION (**SERVER**
OPERATIONS(
ASYNCHRONOUS
REPLY
RETRANSMITTING
CANCEL
NO_LINKED_OPERATION
NO_ENCRYPTION
NO_TOKENSUPPORT)
)
::=StoreMedFunctionControlContext

StoreUserControlContextTemplate
COMMUNICATIONCONTEXT

TRANSPORT (
DOUPLEX,
THROUGPUT: *<64000> Bit/sec*,
TRANSMITDELAY: *<100> msec,*
DELAYJITTER: *<10> msec*
TRANSFERFAILUREPROB: *<0> %*
)
SINGLE-PARTY
CONNECTION-MANAGEMENT(
CONNECTIONORIENTED(
ESTABLISH-INITIATOR
RELEASE-INITIATOR
REESTABLISHING)
)
INTERACTION (**CLIENT**
OPERATIONS(
ASYNCHRONOUS
REPLY
RETRANSMITTING
CANCEL
NO_LINKED_OPERATION
NO_ENCRYPTION
NO_TOKENSUPPORT)
)
::=StoreUserControlContext

```
   ContinuousDataSourceContextTemplate
COMMUNICATIONCONTEXT
   TRANSPORT (
      SIMPLEX,
      THROUGHPUT:  <x> Bit/sec,
      TRANSITDELAY:  <130> msec,
      DELAYJITTER:  <33> msec
      TRANSFERFAILUREPROB: <20> %,
      )
MULTI-PARTY (
    STATIC CENTRAL (SENDER)
       GROUP_ONLY
       NO_FLOORCONTROL
       NONE
    )
   CONNECTION-MANAGEMENT(
  CONNECTIONORIENTED(
      ESTABLISH-INITIATOR
      RELEASE-INITIATOR
      REESTABLISHING)
       )
INTERACTION (CLIENT
    MESSAGES(
          UNCONFIRMED
          NO_RETRANSMITTING
          NO_CHECK_POINTS
          NO_ENCRYPTION
          NO_TOKEN_SUPPORT)
          )
 ::=ContinuousDataSourceContext
```

```
   ContinuousDataSinkContextTemplate
COMMUNICATIONCONTEXT
   TRANSPORT (
      SIMPLEX,
      THROUGHPUT:  <x> Bit/sec,
      TRANSITDELAY:  <130> msec,
      DELAYJITTER:  <33> msec
      TRANSFERFAILUREPROB: <20> %,
      )
MULTI-PARTY (
      STATIC CENTRAL (RECEIVER)
          GROUP_ONLY
          NO_FLOORCONTROL
          NONE
      )
   CONNECTION-MANAGEMENT(
   CONNECTIONORIENTED(
          ESTABLISH-RESPONDER
          RELEASE-RESPONDER
          REESTABLISHING)
   )
INTERACTION (SERVER
     MESSAGES(
          UNCONFIRMED
          NO_RETRANSMITTING
          NO_CHECK_POINTS
          NO_ENCRYPTION
          NO_TOKEN_SUPPORT)
      )
::= ContinuousDataSinkContext
```

```
   ContinuousDataSourceContext
COMMUNICATIONCONTEXT
   TRANSPORT (
      SIMPLEX,
      THROUGHPUT:  <19> MBit/sec,
      TRANSITDELAY:  <130> msec,
      DELAYJITTER:  <33> msec
      TRANSFERFAILUREPROB: <20> %,
      )
MULTI-PARTY (
      STATIC CENTRAL (SENDER)
      GROUP_ONLY
      NO_FLOORCONTROL
      NONE
      )
CONNECTION-MANAGEMENT(
    CONNECTIONORIENTED(
         ESTABLISH-INITIATOR
         RELEASE-INITIATOR
         REESTABLISHING)
    )
INTERACTION (CLIENT
   MESSAGES(
        UNCONFIRMED
        NO_RETRANSMITTING
        NO_CHECK_POINTS
        NO_ENCRYPTION
        NO_TOKEN_SUPPORT)
        )
 ::=ContinuousDataSourceContext
```

```
   ContinuousDataSinkContext
COMMUNICATIONCONTEXT
   TRANSPORT (
      SIMPLEX,
      THROUGHPUT:  <190> MBit/sec,
      TRANSITDELAY:  <130> msec,
      DELAYJITTER:  <33> msec
      TRANSFERFAILUREPROB: <20> %,
      )
MULTI-PARTY (
      STATIC CENTRAL (RECEIVER)
      GROUP_ONLY
      NO_FLOORCONTROL
      NONE
      )
CONNECTION-MANAGEMENT(
      CONNECTIONORIENTED(
           ESTABLISH-RESPONDER
           RELEASE-RESPONDER
           REESTABLISHING)
   )
INTERACTION (SERVER
     MESSAGES(
        UNCONFIRMED
        NO_RETRANSMITTING
        NO_CHECK_POINTS
        NO_ENCRYPTION
        NO_TOKEN_SUPPORT)
        )
::= ContinuousDataSinkContext
```

Anhang B
Komponentenbeispiele

An dieser Stelle sind alle Komponenten–Definitionen und –Schablonen aus der Dissertation wiederholt sowie weitere für die Grundfunktionen und Verarbeitungsfunktionen eingefügt.

Lose gekoppelte Quellenkomponenten lassen sich aus folgender Schablone ableiten:

```
<source_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
      data     : <continuous_data>;
    SUPPLIER AT
      control  : <any_source_control>;
  RESOURCES
      <source device>, <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
      data.setup <
      (control.?;
       data.receive_seq_parameters; data.receive_seq_elemen
      )°
      control.send_sequence_parameters =>
      data.receive_seq_parameters
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
      manage   : <cont_media_management>;
  COOPERATION PROTOCOL
      (manage.init | data.setup) <
      ((control.?;
       data.receive_seq_parameters; data.receive_seq_elemen
       )°
       control.send_sequence_parameters =>
       data.receive_seq_parameters
      )
      < manage.release
      manage.init => data.setup
COMMUNICATION PROPERTIES
      <ContinuousDataSourceContextTemplate> at data;
      <ContMedFunctionControlContextTemplate> at control;
      ManagementContext at manage;
```

Der lesende Speicher als Schablone einer lose gekoppelten Quellenkomponente lautet:

```
<seq_read> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
       data     : <continuous_data>;
    SUPPLIER AT
       control  : <seq_read_control>;
  RESOURCES
       <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
       data.setup <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
       control.extract =>
       (data.receive_seq_parameters<data.receive_seq_element°)
       control.send_seq =>
       (data.receive_seq_parameters<data.receive_seq_element°)
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       (manage.init | data.setup) <
       ((control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
        control.extract =>
        (data.receive_seq_parameters<data.receive_seq_element°)
        control.send_seq =>
        (data.receive_seq_parameters<data.receive_seq_element°)
       )
       < manage.release
       manage.init => data.setup
COMMUNICATION PROPERTIES
       ContinuousDataSourceContext at data;
       ContMedFunctionControlContext at control;
       ManagementContext at manage;
```

Eng gekoppelte Quellenkomponenten lassen sich aus folgender Schablone ableiten:

```
<tightly_coupled_source_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
       data     : <continuous_data>;
    SUPPLIER AT
       control  : <any_source_control>;
  RESOURCES
       <source device>, <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
       data.setup <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen
       )°
       control.send_sequence_parameters =>
       data.receive_seq_parameters
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       (manage.init | data.setup) <
       ((control.?;
        data.receive_seq_parameters; data.receive_seq_elemen
        )°
        control.send_sequence_parameters =>
        data.receive_seq_parameters
       )
       < manage.release
       manage.init => data.setup
COMMUNICATION PROPERTIES
       <ContinuousDataSourceContextTemplate> at data;
       <TightMedFunctionControlContextTemplate> at control;
       ManagementContext at manage;
```

Die Perzeptions–Schablone als eng gekoppelte Quellenkomponente lautet:

```
<cont_perception> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
       data     : <continuous_data;>
    SUPPLIER AT
       control  : <perception_control>;
  RESOURCES
       <source device>, <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
       data.setup <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
       control.send_sequence_parameters =>
       data.receive_seq_parameters;
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       (manage.init | data.setup) <
       ((control.?;
        data.receive_seq_parameters; data.receive_seq_elemen
        )°
       control.send_sequence_parameters =>
       data.receive_seq_parameters)
       < manage.release
       manage.init => data.setup;
COMMUNICATION PROPERTIES
       <ContinuousDataSourceContextTemplate> at data;
       <TightMedFunctionControlContextTemplate> at control;
       ManagementContext at manage;
```

Lose gekoppelte Senkenkonponenten lassen sich aus folgender Schablone ableiten:

```
<sink_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
      SUPPLIER AT
       data     : <continuous_data>
       control  : <any_sink_control>
  RESOURCES
       <sink device>, <memory>, <schedule requirements>;
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
       < manage.release
COMMUNICATION PROPERTIES
       <ContinuousDataSinkContextTemplate> at data
       <ContMedFunctionControlContextTemplate> at control
       ManagementContext at manage;
```

Der schreibende Speicher als lose gekoppelte Senkenkomponente lautet:

```
<seq_write> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    SUPPLIER AT
       data     : <continuous_data>;
       control  : <seq_write_control>;
  RESOURCES
       <memory>, <schedule requirements>;
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
       < manage.release
COMMUNICATION PROPERTIES
       ContinuousDataSinkContext at data;
       ContMedFunctionControlContext at control;
       ManagementContext at manage;
```

Eng gekoppelte Senkenkomponenten lassen sich aus folgernder Schablone ableiten:

```
<tightly_coupled_sink_component_template> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       SUPPLIER AT
       data     : <continuous_data>;
       control  : <any_sink_control>;
  RESOURCES
       <sink device>, <memory>, <schedule requirements>;
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
       < manage.release
COMMUNICATION PROPERTIES
       <ContinuousDataSinkContextTemplate> at data
       <TightMedFunctionControlContextTemplate> at control
       ManagementContext at manage;
```

Die Schablone für die eng gekoppeltePräsentation lautet:

```
<cont_presentation> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
       SUPPLIER AT
       data     : <continuous_data>;
       control  : <presentation_control>;
  RESOURCES
       <sink device>, <memory>, <schedule requirements>;
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       manage.init <
       (control.?;
        data.receive_seq_parameters; data.receive_seq_elemen)°
       < manage.release
COMMUNICATION PROPERTIES
       <ContinuousDataSinkContextTemplate> at data
       <TightMedFunctionControlContextTemplate> at control
       ManagementContext at manage;
```

Die Schablone für einen Schreib–Lese–Speicher lautet:

```
<simple_read_write> COMPONENT
APPLICATION PROPERTIES
  INTERFACES
    CONSUMER AT
       dataout  : <continuous_data>;
    SUPPLIER AT
       datain   : <continuous_data>;
       control  : <simple_read_write_control>;
  RESOURCES
       <memory>, <schedule requirements>;
  COOPERATION PROTOCOL
       dataout.setup|datain.setup <
       (control.?;
       ((dataout.receive_seq_parameters;
         dataout.receive_seq_elem)
       |(datain.receive_seq_parameters;
         datain.receive_seq_elem)))
       control.extract =>
       (data.receive_seq_parameters<data.receive_seq_element°)
       control.send_seq =>
       (data.receive_seq_parameters<data.receive_seq_element°)
MANAGEMENT PROPERTIES
  INTERFACES
    SUPPLIER AT
       manage   : <cont_media_management>;
  COOPERATION PROTOCOL
       (manage.init | dataout.setup) <
       ((control.?;(dataout.? | datain.?))
        control.extract =>
        (dataout.receive_seq_parameters <
         dataout.receive_seq_element°)
        control.send_seq =>
        (dataout.receive_seq_parameters <
         dataout.receive_seq_element°)
       )°
       < manage.release
       manage.init => dataout.setup
COMMUNICATION PROPERTIES
       ContinuousDataSinkContext at datain;
       ContinuousDataSourceContext at dataout;
       ContMedFunctionControlContext at control;
       ManagementContext at manage;
```

Die Schablone für eine Verteilte Anwendung lautet:

```
<Name> DISTRIBUTED APPLICATION
COMPONENTS
       <Name> : <Komponententyp>;
                :
       <Name> : <Komponententyp>;
BINDINGS
<Komponente>.<Schnittstelle> -- <Komponente>.<Schnittstelle>
                           :
<Komponente>.<Schnittstelle> -- <Komponente>.<Schnittstelle>
CONSTRAINTAS
<Liste der Konfigurationsbedingungen>
PLACEMENT
       <Plazierungsbedingungen>
```

Anhang C
Klassendefinitionen

Deklaration von **c_Periodicker,** der gemeinsamen Basisklasse für Ticker und Schrittgeber:

```
class obj (); // dummy class

typedef  void (obj:: *t_voidmethod) ();
// pointer to dummy class method taking and returning void

class  c_Periodicker {
protected:
       t_clock *p_Value;          //Value of Clock
       t_clock *p_MasterValue;    //and Masterclock
       t_speed *p_Rate;           //Rate of Clock
       t_speed *p_MasterRate;     //and Masterclock
       t_clock *p_Ofset;          //Masterclock Value of Clock 0

public:
       c_Periodicker (
                t_speed Rate,
                t_speed MasterRate,
                t_clock Offset = 0,
                t_clock Value  = 0
       );

       ~c_Periodicker ();

       int      set_Value (t_clock_Value);

       t_clock  get_Value (bool wait);
       //Read value. If <wait>, wait for next change of value.

       int      set_Rate (t_speed Rate);

       t_speed  get_Rate (void);

       int      set_Offset (t_clock Offset);

       t_clock  get_Offste (viod);

}
```

Lebenslauf

Persönliche Daten

Johannes Christian Fritzsche
geboren am 2. Mai 1961
in Nürnberg
verheiratet, ein Kind

Schulbesuch

September	1967–Juli 1969	Schönbergschule, Volksschule Freiburg im Breisgau
September	1969–Juli 1971	Schneeburgschule, Volksschule Freiburg i. Breisgau
September	1971–Juni 1980	Kepler Gymnasium Freiburg im Breisgau Abschluß: Abitur

Praktikum

Juli 1980	–September 1980	Fraunhofer–Institut für angewandte Festkörper–Physik (IAF) der Fraunhofer-Gesellschaft in Freiburg im Breisgau

Studium

1. Oktober	1980	Beginn des Informatikstudiums an der Universität Fridericiana Karlsruhe (TH)
12. Oktober	1983	Abschluß der Diplom–Vorprüfung Gesamtnote: befriedigend
19. Januar	1988	Abschluß der Diplom–Hauptprüfung Gesamtnote: gut

Wehrdienst

1. April 1988	–30. Juni 1988	Grundausbildung
1. Juli 1988	–15. Januar 1989	Lehrtätigkeit in der Technischen Schule der Luftwaffe 1

Berufstätigkeit

1. Februar 1989-31.Januar 1994	wissenschaftlicher Mitarbeiter, an der Professur für Architektur und Betrieb verteilter Systeme (ABVS) und Telematik bei Prof. Dr. O. Drobnik an der Johann Wolfgang Goethe–Universität in Frankfurt am Main
1. Januar 1995-30. Juni 1995	System Manager bei Michael Conrad und Leo Burnett, Werbeagentur in Frankfurt am Main.
seit 1. Januar 1996	Seminarleiter bei Computer Technik Czakay (cTc GmbH) Hasselroth