
Seminar Multimediale Informationssysteme

Arbeitsgruppe Datenbanken und Informationssysteme
Universität Kaiserslautern

# Hypermedia: Modelle und Anwendungen

Ausarbeitung von

**Andreas Kümpel**
eMail: andreas.kuempel@gmx.de

30. Juni 2002

# INHALT

# Abstrakt

Hypermedia-Anwendungen, zusammengesetzte und miteinander vernetzten diskrete und kontinuierliche Medien mit der Möglichkeit zur Benutzerinteraktion, haben in heutiger Zeit immer mehr an Bedeutung gewonnen und finden immer mehr Einzug in das tägliche Leben. Angefangen vom World Wide Web, über interaktive Lernprogramme bis hin zum digitalen Fernsehen mit Möglichkeit zur Interaktion.

Ohne ein Modell als Grundlage sind Hypermedia-Anwendungen nur sehr schwierig oder sogar gar nicht zu realisieren. Zudem ist die Portabilität zwischen unterschiedlichen Systemen ohne festgelegte Standards, nach denen sich die Entwickler für Hypermedia-Anwendungen richten können, kaum oder nur mit viel Aufwand möglich. Diese Ausarbeitung soll einen kurzen Überblick über die bekanntesten Hypermedia-Modelle bringen, wie beispielsweise Dexter, HyTime und MHEG, und eine kleine Auswahl von Anwendungen, darunter das Cardio-OP-Projekt, genauer erläutern.

# 1 Einleitung

Der Begriff „Hypermedia“ ist eine Kombination der Wörter „Hypertext“ und „Multimedia“. Die Geschichte des „Hypertextes“ begann im Jahre 1945, als Vannevar Bush einem Artikel im „Atlantic Monthly“ über das Memex-System („memory extension“) schrieb. Sein System, bei der die ständig wachsende Menge an Literatur auf Mikrofilm gespeichert werden sollte, erlaubte es zum ersten Mal, eine automatische Suche durchzuführen.

Ted Nelson prägte 1965 den Begriff „Hypertext“ durch sein System „Xanadu“. Hypertext war demnach die Präsentation von Informationen als ein lineares Netzwerk von Knoten, durch das der Leser frei und „nicht-linear“ navigieren konnte, indem innerhalb eines Hypertext-Dokumente sogenannte Anker („anchors“) direkt auf andere Dokumente verwiesen. Auf diese Weise waren mehrere Lesepfade möglich. Bei den im Hypertext benutzten Medien handelte es sich um sogenannte diskrete Medien, d.h. von der Zeit unabhängige statische Medien, wie Text oder Bilder.

Die Einbettung von kontinuierlichen Medien, d.h. Medien, die abhängig von der Zeit sind und ihre akustische oder optische Darstellung abhängig von der Zeit ändern, wie Videos oder Audio-Ströme, in diskrete Medien, führte zur Bildung des Wortes „Hypermedia“.

Hypermedia hat in der heutigen Zeit immer mehr Einzug in das tägliche Leben gefunden. Angefangen vom World Wide Web, über interaktive Lernprogramme am PC bis hin zum digitalen Fernsehen mit der Möglichkeit zur Interaktion. Aus diesem Grund soll diese Ausarbeitung einen kleinen Einblick in Hypermedia-Modelle und deren Anwendungen geben.

Im zweiten Kapitel dieser Ausarbeitung werden die Modelle von Dexter, das Amsterdam-Hypermedia-Modell, HyTime (zusammen mit SGML), MHEG-5 und das $Z_YX$-Modell vorgestellt. Im dritten Kapitel werden zwei Anwendungen, die mit Hilfe von in Kapitel zwei vorgestellten Modellen realisiert wurden, näher erläutert, darunter das Cardio-Op-Projekt, bei dem das $Z_YX$-Modell zum Einsatz kam, sowie das GLASS-System, eine MHEG-Anwendung. Im vierten Kapitel erfolgt ein kurzer Ausblick auf die weitere Entwicklung von Hypermedia-Modellen und -Anwendungen.

# 2 Hypermedia-Modelle

Ein Hypermedia-Modell dient dazu, Hypermedia-Anwendungen nach einem bestimmten Muster zu einer bestimmten Struktur entwickeln. Durch diese vereinheitliche Entwicklung und der Struktur einer Hypermedia-Anwendung, also einer Standardisierung, soll eine Verbreitung von Hypermedia-Anwendungen vereinfacht werden.

Grundlegende Idee bei der Entwicklung solcher Hypertext-Modelle (und später Hypermedia-Modelle) war es, den Informationsgehalt eines Dokumentes von dessen Darstellung zu trennen, um so eine einfache Portierung von Dokumenten unabhängig von der verwendeten Hardware oder Software zu ermöglichen.

Grundsätzlich fallen unter den Begriff Hypermedia-Modelle zwei verschiedene Modelltypen: Zum einen Modelle für Hypertext (z.B. SGML), d.h. vernetzte Dokumente, die ausschließlich diskrete Medien beinhalten, sowie Modelle für Hypermedia (z.B. $Z_YX$), vernetzte Dokumente, die neben diskreten Medien auch kontinuierliche Medien beinhalten. D.h. Hypermedia-Modelle sind als Erweiterung von Hypertext-Modellen zu betrachten.

Abbildung 1 zeigt die zeitliche Entwicklung und die Abhängigkeiten der im Kapitel zwei vorgestellten Hypermedia-Modelle.

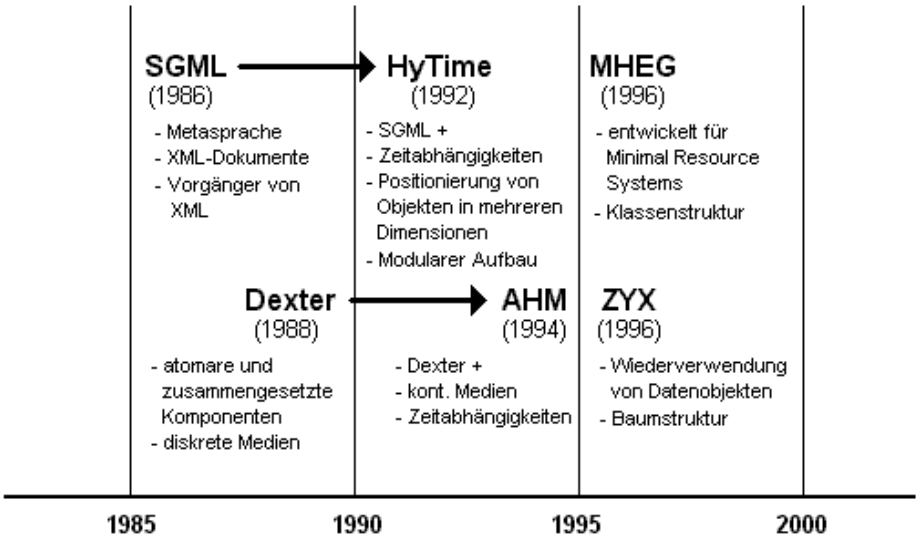


Abb. 1
Entwicklung von Hypertext- und Hypermedia-Modellen

## 2.1 Das Dexter Hypertext Modell

### 2.1.1 Geschichte

Eine Gruppe von Hypertextsystem-Entwicklern traf 1988 zu einem Workshop zusammen, um ein einheitliches Hypertext-Konzept zu schaffen. Der Zweck dieses Konzeptes war ein möglicher Vergleich zwischen bereits bestehenden Hypertextsystemen (KMS, Hypercard, Augment, Intermedia, Neptune [11]), sowie die Entwicklung neuer Systeme mittels eines einheitlichen Modells. Trotz der unterschiedlichen Eigenschaften der einzelnen Systeme, gelang es der Gruppe ein Modell zu entwickeln, das abstrakte Terminologien und Definitionen verwendete, die auf die meisten bereits bestehenden Systeme zutrafen [11]. Das Dexter-Modell ist allerdings kein Standard im Sinn von Normierung, sondern dient als Referenz-Modell für andere Modelle. Das Dexter-Modell (der Workshop fand im Dexter Inn in Sunapee, New Hamshire, statt) wurde seit dem immer weiter verfeinert. Auf der Basis des Dexter-Modells sind inzwischen weitere Modelle entwickelt worden, wie zum Beispiel das „Amsterdam-Hyper-Media-Modell“ (siehe Kapitel 2.2).

### 2.1.2 Beschreibung

Das Dexter-Modell unterteilt ein Hypertextsystem in drei Schichten: *das „within-component Layer*“, das „*storage Layer*“ und das „*runtime Layer*“. Die drei Schichten sind untereinander mittels festgelegter Schnittstellen verbunden. Abbildung 2 zeigt die Schichten des Dexter-Modells. Die Aufgaben der einzelnen Schichten werden im folgenden kurz erläutert.

#### storage Layer

Im *storage Layer* befindet sich die Datenbasis, die eine Hierarchie von Komponenten mit unterschiedlichen Daten speichert. Dazu werden abstrakte Komponenten definiert, die zu den Multimedia-Daten (den Datenobjekten) Attribute und Darstellungsinformationen besitzen. Der genaue Aufbau der einzelnen Datenobjekte und der darin verwendeten Medientypen wird nicht im *storage Layer* definiert, sondern im w*ithin-component Layer*. Zur Verknüpfung der Komponenten

untereinander werden sogenannte Links definiert, wodurch ein Netz von Komponenten entsteht. Bei den Komponenten unterscheidet man zwischen atomaren Komponenten und zusammengesetzten Komponenten.

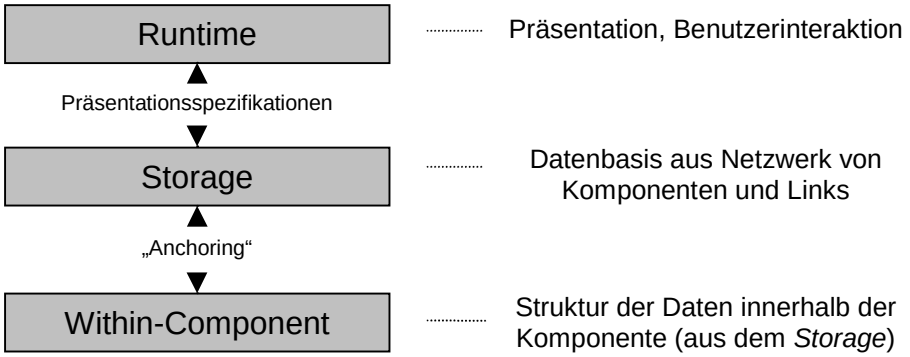


Abb. 2
Dexter Modell - Schichten

*Atomare Komponenten* bestehen aus drei Teilen: dem Dateninhalt, z.B. Text, Grafiken, Videos oder Programmfragmenten; den Attributen (semantische Beschreibung); drittens den Darstellungsspezifikationen, in denen beschrieben wird, wie die Komponenten auf dem Bildschirm dargestellt werden sollen, wobei sich die Spezifikation unterscheiden kann, je nachdem, auf welchen Weg (über unterschiedliche Komponenten und Links) der Benutzer zu der entsprechenden Komponente gelangt ist. So könnte sich beispielsweise die Darstellung unterscheiden, wenn die Komponente mittels eines *Textbetrachters* von einem normalen Benutzer betrachtet wird, oder wenn die Komponente mittels eines *Editors* von einem Lehrer oder Autor betrachtet wird.

*Zusammengesetzte Komponenten* beinhalten neben dem Dateninhalt Verweise zu anderen atomaren oder wiederum zusammengesetzten Komponenten, den *Children*. Diese Children können innerhalb des Inhalts eingesetzt werden. Komponenten dürfen dabei nicht sich selber als Teilkomponenten enthalten.

Links stellen Verbindungen zwischen den Komponenten dar. Dabei wird ein Link zwischen zwei oder mehreren Komponenten, den Endpunkten, gebildet. Um diese Endpunkte zu definieren, bedient sich das Dexter Modell des „*Anchoring*“. Zur Festlegung eines Start- oder Endpunktes innerhalb einer Komponente ist es notwendig, den sogenannten *Unique-Identifier (UID)* der Komponente, sowie die *Anchor-ID* anzugeben. Ein *Anker* gibt eine eindeutig bestimmte Region innerhalb der Komponente an. Zu diesem Zweck enthält der *Anker* einen Wert, wobei dies die Position eines Wortes innerhalb eines Textes sein kann. Da die Inhalte einer Komponente nicht im s*torage Layer*, sondern im w*ithin-component Layer* definiert sind, können die Werte im *Anker*, die in direkter Beziehung zu den Inhalten stehen, auch erst im w*ithin-component Layer* genauer bestimmt werden und bleiben im s*torage Layer* abstrakt. Neben der Definition der Endpunkte eines Links ist es im Dexter-Modell möglich, eine Präsentationsspezifikation und eine Richtung des Links anzugeben („From“ oder „To“ alternativ zu „Bidirect“).

In Abbildung 3 ist der Aufbau von atomaren und zusammengesetzten Komponenten im Dexter-Modell veranschaulicht, sowie die Verbindung zwischen zwei Komponenten mit Hilfe von „*Links*“.

## runtime Layer

Die zweite Schicht des Dexter Modells, das *runtime Layer*, hat die Aufgabe, die gespeicherten Daten dem Benutzer darzustellen und auf mögliche Interaktionen des Benutzers zu reagieren, wobei die

Interaktionsmöglichkeiten von den Autoren des Dexter-Modells nicht genauer spezifiziert wurden. Um den parallelen Zugriff mehrerer Benutzer zu ermöglichen, arbeitet das *runtime Layer* nicht direkt mit den im *storage Layer* gespeicherten Daten, sondern mit einzelnen Instanzen der Daten, d.h. Duplikaten, die im Falle einer Änderung dann wieder in das *storage Layer* geschrieben werden.

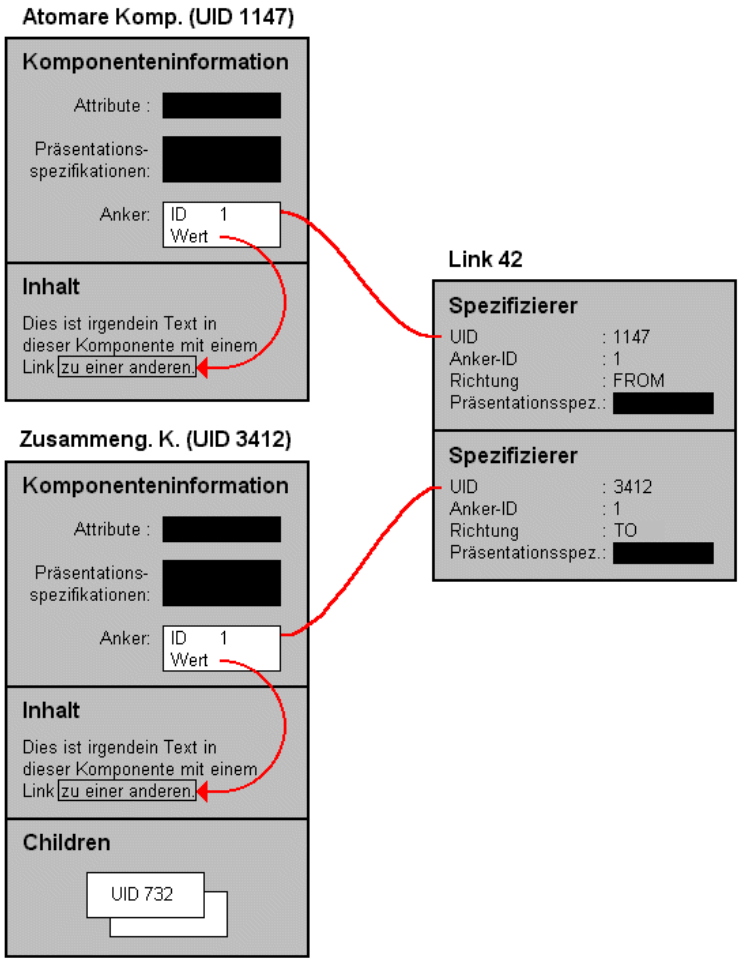


Abb. 3
verlinkte atomare und zusammengesetzte Komponente im Dexter-Modell

### within-component Layer

Das *within-component Layer* wird im Dexter-Modell ebenfalls nur rudimentär beschrieben. Es definiert die Struktur der Datenobjekte. Zudem wird festgelegt, welche Typen von Medien unterstützt werden sollen (Grafik-Formate, etc.). Aufgrund der hohen Anzahl an unterschiedlichen Typen und deren unterschiedlichen Eigenschaften, haben die Entwickler des Dexter-Modells diese Schicht nicht genauer beschrieben und überlassen dies den Entwicklern von Hypermedia-Anwendungen.

### 2.1.3 Zusammenfassung

Das Dexter-Modell hat sich in der Praxis zum Vergleich bestehender Hypertextsysteme bewährt. Allerdings können mit dem Dexter-Modell lediglich die Datenbasis und die Verbindung der Komponenten verglichen werden. Auf die Darstellungsfunktionen und die möglichen Benutzerinteraktionen wird hingegen seitens der Autoren des Modells nicht genauer eingegangen.

## 2.2 Das Amsterdam Hypermedia Modell

Aufgrund der fehlenden Unterstützung von Multimedia im Dexter-Modell, das sich nur auf Hyper*text*-Systeme beschränkt, wurde ein Modell für Hypermedia entwickelt, das Amsterdam Hypermedia Modell (AHM). Das AHM baut direkt auf das Dexter-Modell und das CMIF-Modell [6] auf. Das CMIF-Modell (CWI Multimedia Interchange Format Model) konzentriert sich auf die zeitorientierte Organisation von Informationen, also die zeitlichen Abhängigkeiten (beispielsweise die Einblendung eines Textes zu einem bestimmten Zeitpunkt beim Abspielen eines Videos), und die hierarchische Strukturierung von Dokumenten. Auf das CMIF-Modell wird an dieser Stelle jedoch nicht genauer eingegangen.

Um eine Unterstützung für Hypermedia zu gewährleisten, waren eine Reihe von Erweiterungen am Dexter-Modell notwendig.

## Komposition von mehreren dynamischen Daten

Bei einem Hypermedia-Modell muss die Möglichkeit geboten werden, einzelne Elemente (Text, Bilder, Buttons, etc.) in eine Präsentation zusammen zu fassen. Dazu sollen diese Elemente in eine zeitliche Abhängigkeit gesetzt werden können. Zudem wird bei den zusammengesetzten Komponenten (siehe Dexter, 2.1) der Dateninhalt zum Zwecke besserer Strukturierung entfernt, so dass sich nur noch in atomaren Komponenten die eigentlichen Daten befinden.

## Higher Level Präsentationsspezifikationen („Channels“)

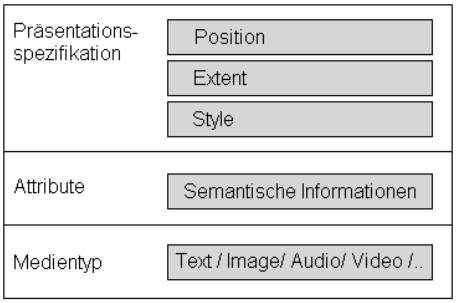


Abb. 4 - Channel

„*Channels*“ (vgl. CMIF [6]) sind abstrakte Ausgabegeräte, die zum Abspielen von Multimedia-Daten dienen. So enthält beispielsweise ein Textchannel Attribute wie die Schriftart, Schriftfarbe, Schriftgröße, sowie Angaben zur Position auf dem Bildschirm. Wird ein Textelement durch diesen *Channel* geleitet, so nimmt er alle Attribute des *Channels* an. Auf diese Weise müssen nicht für jedes einzelne Element die Attribute geändert werden. Zudem kann auf diese Weise das Layout „wieder benutzt“ werden. Beispielsweise befindet sich ein „Videochannel“ auf einem bestimmten Bereich des Bildschirms. Wird ein Video gestartet, so ist durch die Verbindung des Videos mit dem *Channel* bereits festgelegt, wo auf dem Bildschirm das neue Video ablaufen soll. Im AHM werden *Channels* global definiert und können von jedem beliebigen atomaren Element referenziert werden. Abbildung 4 zeigt den Aufbau eines *Channels*.

## Kombinieren von „zusammengesetzten Komponenten“

Wenn mehrere Komponenten zusammengesetzt werden, so muss darauf geachtet werden, ob für die zusammengesetzte Komponente genug Ressourcen vorhanden sind. Wird zum Beispiel ein Text, ein Button und ein bildschirmfüllendes Video zu einer Komponente zusammen gefasst, so kann entweder das Video nicht korrekt dargestellt werden, oder der Text und die Buttons sind nicht zu sehen. Um die benötigten Ressourcen und eventuelle Konflikte zu berechnen, können die Informationen aus den *Channels* und den Zeitabhängigkeiten der einzelnen atomaren Komponenten zu Hilfe genommen werden.

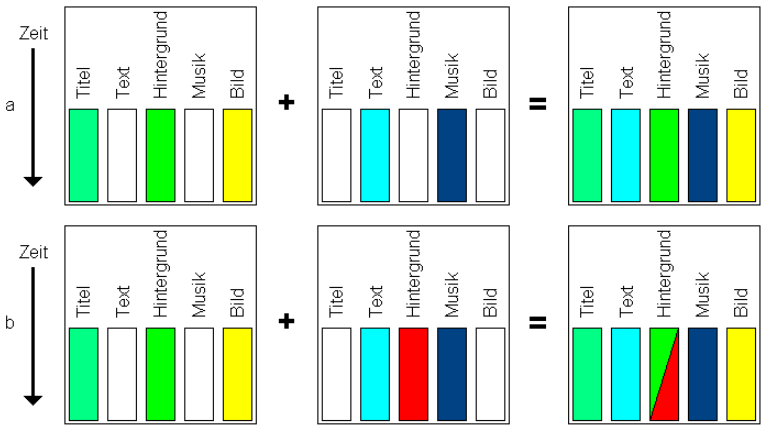


Abb. 5
Ressourcenkonflikte im AHM

Im AHM wird dies nicht explizit durch Erweiterungen ermöglicht, sondern durch die Benutzung der bereits eingeführten *Channels*. Abbildung 5 zeigte mögliche Kombination von zusammengesetzten Komponenten und eine nicht mögliche Kombination. Die Komponenten belegen dabei spezielle

*Channels* (farbig markiert), zum Beispiel „*Titel*“, „*Hintergrund*“ und „*Bild*“. Wird solche eine Komponente mit einer Komponente kombiniert, die beispielsweise ebenfalls den *Channel* „*Hintergrund*“ belegt, kommt es zum Konflikt, hier rot-grün dargestellt. Dieser Konflikt müsste dann durch die zu realisierende HyTime-Engine behandelt werden.

### Zeitliche Abhängigkeiten

Um dynamische, also zeitabhängige Daten einbinden zu können, müssen zeitliche Abhängigkeiten modelliert werden können. So soll es beispielsweise in einer Hypermedia-Anwendung möglich sein, beim Abspielen eines Videos zu einem bestimmten Zeitpunkt einen Button zu drücken, um genauere Informationen zu einem zu diesem Zeitpunkt im Video vorgestellten Thema zu erhalten. Um dies im AHM modellieren zu können, wurden die „*Synchronization Arcs*“ [6] neu eingeführt. Diese „*Synchronization Arcs*“ enthalten einen Zeitpunkt und eine mögliche Abweichung, in der sich die Synchronisation zweier Komponenten bewegen kann.

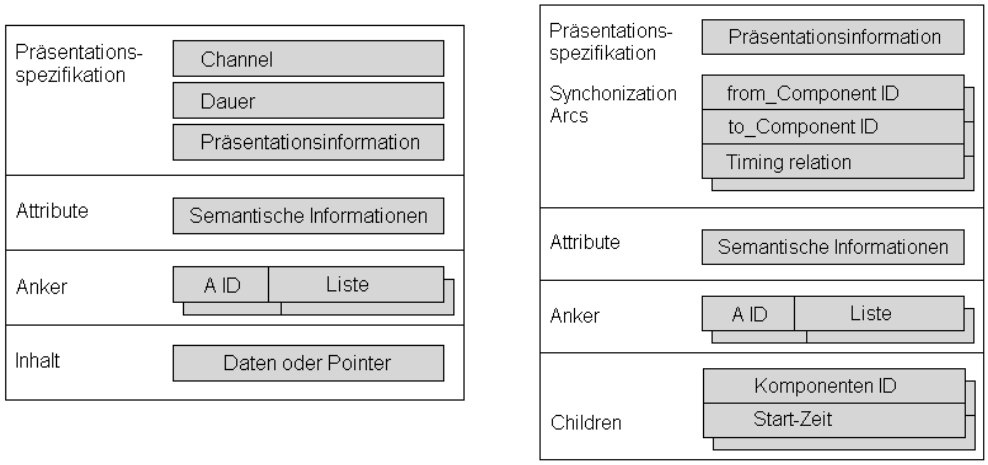


Abb. 6
Atomare und zusammengesetzte Komponenten im AHM [6]

### Kontext für „Anker“

Bei bisherigen Hypertext-Systemen ist keine explizite Notation vorgesehen, was sich von den dargestellten Informationen ändert und was bestehen bleibt, sobald ein Benutzer einem Link folgt. Der Kontext eines Ankers ist der Teil einer Dokumentenstruktur, der betroffen ist, sobald einem Link gefolgt wird. Dies wird im AHM dadurch ermöglicht, dass bei den *Ankern* mehrere betroffene Komponenten eingetragen werden können.

Abbildung 6 zeigt die vom Dexter-Modell (Abb. 3) überarbeiteten atomaren und zusammengesetzten Komponenten im Amsterdam Hypermedia Modell.

## 2.3 HyTime

HyTime (*Hypermedia / Time-based Document Structuring Language*) ist seit 1992 ein Standard (ISO 10744) zur Strukturierung von Multimedia Dokumenten. HyTime basiert auf dem Standard SGML (*Structured Generalized Markup Language*) und ermöglicht es die Beziehungen zwischen diskreten und kontinuierlichen Daten zu beschreiben. Ein HyTime-Dokument ist ein SGML-Dokument, dessen Eigenschaften, wie im HyTime Standard definiert, repräsentiert sind. Es ist also ein SGML-Dokument, das HyTime-Konstrukte beinhaltet. Aus diesem Grund wird im folgenden kurz SGML näher erläutert.

### 2.3.1 Der Standard: SGML

SGML ist eine Metasprache, ISO-Standard, der Syntax und Semantik zur Konstruktion von Dokumentenbeschreibungssprachen zur Verfügung stellt. Ein SGML-konformes Dokument besteht aus drei verschiedenen Teilen: Der eigentliche Inhalt (Text) des Dokumentes, die logische Struktur des Dokumentes (DTD, *Document Type Definition*), sowie die Darstellungsform des Dokumentes (*SGML Declaration*), worin die in den anderen beiden Teilen zu verwendenden Symbole definiert werden. Die *SGML Declaration* ist optional. Wird sie nicht extra definiert, so wird automatisch eine Standard-Deklaration verwendet, die durch das Einsetzen einer eigenen Deklaration außer Kraft gesetzt wird.

In einem SGML-konformen Dokument werden die einzelnen Strukturelemente durch sogenannte Tags gekennzeichnet. Diese einzelnen Elemente haben einen Namen und können ggf. Attribute besitzen. Diese Namen und Attribute werden in einem separaten Teil des SGML-Dokuments (oftmals in einer eigenen Datei), der *Document Type Definition* definiert. Die Kodierung im eigentlichen Dokument, also der Inhalt, muss sich nach diesen in der DTD festgelegten Syntax richten und wird im SGML-Format (vergleichbar mit XML) gespeichert. Ein Parser kann mittels der DTD die Dokumente dann auslesen.

Im folgenden ist ein Beispiel für die Definition einer *Document Type Definition* und eines SGML-Dokumentes kurz aufgeführt. Auf die Darstellungsform wird hier nicht näher eingegangen.

Beispiel DTD „*seminar.dtd*“:

```
<!DOCTYPE seminar [
<!ELEMENT seminar       - -   (kopf, text+)          -->
<!ELEMENT kopf          - -   (thema, person)        -->
<!ELEMENT thema         - -   (#PCDATA)              -->
<!ELEMENT person        - -   (#PCDATA)              -->
<!ELEMENT text          - o   (#PCDATA)              -->
                                                   ] -->
```

Das SGML-Dokument „*mein_seminar.sgml*“:

```
<!DOCTYPE seminar SYSTEM "seminar.dtd">
<SEMINAR>
<KOPF>
<THEMA>Mein erstes Seminar</THEMA>
<PERSON>Ich selber</PERSON>
</KOPF>
<TEXT>Der erste Satz.
<TEXT>Der zweite Satz.
</SEMINAR>
```

Bei diesem Beispiel muss ein Dokument des Typs Seminar einen Kopf haben, der aus einem (z.B.) Textelement Thema und Person besteht. Danach folgt der Text, wobei mehr als ein Text („+“) dem Kopf folgen kann. Durch die zwei Zeichen nach dem Namen des Elementes wird festgelegt, ob ein Start- bzw. Stop-Tag entweder erforderlich („-“) oder nicht notwendig („o“) ist.

Eine mit einer SGML-Anwendungen vergleichbare Anwendung ist das World Wide Web (HTML). Dort ist die DTD, die logische Struktur des Dokumentes allerdings durch die HTML-Syntax vorgegeben, die Darstellungsform ist vom jeweiligen Web-Browser abhängig.

Aus SGML ist später XML hervorgegangen, eine für das World Wide Web vereinfachte Form des SGML.

### 2.3.2 HyTime – Geschichte

Die ersten Versuche, Hypertext und Multimedia zusammen in strukturierten Darzustellen, begannen 1984. Charles F. Goldfarb, der Erfinder von SGML, schlug damals ein Projekt zur Standardisierung von Musikdarstellung in SGML vor, das später unter der Bezeichnung SMDL („*Standard Music Description Language*“) bekannt wurde. So wurde aus SMDL eine Sprache zur Beschreibung allgemeiner zeitlicher und synchroner Abläufe. 1989 wurden SMDL-Elemente, die zur Spezifikation zeitlicher Abhängigkeiten dienten, gewählt und in einen neuen, allgemeineren Standard, HyTime, übernommen.

### 2.3.3 HyTime

Der HyTime-Standard ermöglicht, aufbauend auf SGML, die Definition von Markup-Sprachen für Hypermedia-Dokumente. Dazu stellt der Standard eine Liste von Elementtyp-Definitionen zur Verfügung, deren Semantik festgelegt ist. Diese Elementtyp-Definitionen heißen *Architectural Forms* und müssen nicht in eine DTD übernommen werden, sondern können in Elementtypen einer DTD eingebunden werden. HyTime schreibt also keine feste DTD vor. Die Integration einer *Architectural Form* in einen Elementtyp bewirkt, dass ein SGML-Element mit HyTime-Semantik versehen wird. Ein solches SGML-Element wird *HyTime-Element*, ein Elementtyp, der eine *Architectural Form* einbindet, *HyTime-Elementtyp* genannt. Alle *Architectural Forms* werden als *Meta-DTD* bezeichnet. Bei den *Architectural Forms* wird zwischen zwei Arten unterschieden: Die *Elementtyp-Forms* und die *Attributlisten-Forms*. Letztere können als abkürzende Schreibweise für Attributlisten angesehen werden,. Zur Funktionalität von HyTime als Erweiterung von SGML gehört die Möglichkeit zur *Verlinkung* und zum *Scheduling* von Hyperdokument-Komponenten. So können Objekte mittels Queries adressiert werden oder Schedules für kontinuierliche Datenobjekte angegeben werden. Zudem können im Gegensatz zu SGML, wo nur ganze Elemente adressiert werden können, in HyTime beliebige Datenteile angesprochen werden [29].

Die *Architectural Forms* von HyTime sind modular aufgebaut, wobei es lediglich notwendig ist, für ein Dokument, das nur auf einzelne Module von HyTime zurückgreift, auch nur diese Module zur Verfügung zu stellen.

Im folgenden sollen die Funktionsweisen der einzelnen Module nur kurz angedeutet werden.

### 2.3.4 Die Module von HyTime

- **Base Module**

  Das Basis Modul bietet Funktionen, auf die andere Module aufbauen oder die zur Grundfunktionalität erforderlich sind. Dazu gehören zum Beispiel die Verwaltung von Identifikationsnummern der einzelnen Objekte zur Dimensionierung und Positionierung von Objekten. Zudem besteht die Möglichkeit zur Verwendung sogenannter *Activity-Reporting Facilities*. Dabei kann jedem SGML-Dokument ein activity-tracking Attribut zugeordnet werden, welches als Zeiger auf einen Log-Eintrag dient. Damit können folgende Ereignisse mit geloggt werden: Das Erschaffen von Elementen, das ändern von Elementen, das verlinken von Elementen (ein Link wird auf dieses oder von diesem auf ein anderes Element gelegt), das lesende Zugreifen auf ein Element, sowie das löschen von Elementen [11].

- **Location Address Module**

  Dieses Modul enthält *Architectural Forms*, um Datenobjekte und Teile in ihnen zu adressieren. Es gibt drei verschiedene Adressierungsarten: [29]

1. *Namensraum-Adressierung*
   Diese Art der Adressierung wurde aus SGML übernommen. Elemente können durch ihren Namen adressiert werden.
2. *Koordinatenadressierung*
   In HyTime wird zur Adressierung von Objekten ein Koordinatensystem verwendet. Mit Hilfe dieses Koordinatensystems ist es möglich, sowohl beliebige Teile von Objekten als auch mehrdimensionale Objekte zu adressieren. So kann die dritte und vierte Zahl in einem Satz referenziert werden. Der Satz liegt auf einer Achse, dessen Einheiten aus Zahlen bestehen. Die Adressierung der Zahlen im Satz erfolgt durch die Koordinateadresse, einer Positionsangabe auf der Achse.
3. *Semantische Adressierung*
   HyTime erlaubt die Adressierung von Objekten durch die Einbindung semantischer Konstrukte. Diese Einbindung erfolgt durch die indirekte Adressierung einer Notation. Statt semantische Konstrukte in anderen Notatonen einzubinden, kann die „*HyTime Query Language*“ (HyQ) verwendet werden, auf die hier aber nicht näher eingegangen werden soll. Siehe dazu [1]. Eine semantische Adresse beschreibt das gesuchte Objekt durch einige charakteristische Merkmale. Beispielsweise kann das erste Element in einem Dokument gesucht werden, das den Namen XY und ein Attribut Typ mit dem Wert Literaturverweis hat.

- **Hyperlink Module**

  Zur Navigation in HyTime-Dokumenten stellt dieses Modul bidirektionale Hyperlinks zur Verfügung. Zur Adressierung wird das „Location Address Module“ verwendet (siehe oben). Es existieren fünf Arten von Hyperlinks: *Independent Links* (ilinks), bei denen die Anzahl der Link-Enden beliebig ist; *Property Links* (plinks), haben jeweils genau zwei Enden und beschreiben *besitzmäßige* Abhängigkeiten, so zum Beispiel ein Attribut und dessen Belegung; *Contextual Links* (clinks) haben ebenfalls zwei Enden, eines dieser Enden ist immer die Stellen, an der der Link spezifiziert wird, wie zum Beispiel ein Link auf einer Fußnote; *Aggregate Location Links* (agglinks) verbinden mehrere Bereiche eines Dokumentes in der Form, so dass diese als ein Bereich betrachtet werden können; *Span Links* (spanlinks) verbinden Textelemente eines SGML-Dokumentes, falls diese durch SGML-Tags unterbrochen werden.

- **Measurement Module**

  Dieses Modul ist für die Ablaufsteuerung der Darstellung von Objekten zuständig. Die beiden Erweiterungen dieses Moduls, das „***Scheduling Module***“ und das „***Rendition Module***“ bieten weitere Mechanismen zur Ablauf- und Darstellungssteuerung an. Objekte erscheinen hier als Inhalte von „*Events*“. Die Position und Größe eines solchen „*Events*“ werden durch eine Menge von Koordinaten innerhalb eines Event-Ablaufplanes festgelegt [11]. Die Koordinaten werden dabei als Punkte in einem um mehrere (z.B. zeitliche) Dimensionen erweitertes Kartesisches Koordinatensystem betrachtet. Medienobjekte werden in diese endlichen Räume (*Finite Coordinate Spaces*) gesetzt, wobei diese Räume eine Reihe von anwendungsdefinierter Achsen sind.

### 2.3.5 Beispiel

Das Folgende Beispiel (aus [29]) zeigt eine gekürzte Architectural Form. Dieser „*contextual link*“ (clink) ist einer der Link-Typen in HyTime ([11]/ [27]). Jedes HyTime-Element muss ein Attribut HyTime besitzen, dessen Wert der Name des Architectural Forms ist:

```
<!element   clink       - o         (%HyBrid;)* >
<!attlist   clink       HyTime      NAME        clink
                        id          ID          #implied -- default:none--
                        linked      IDREF       #REQUIRED >
```

In dem folgenden Beispiel ist die Deklaration des Elementtyps Fußnote dargestellt, der eine Architectural Form einbindet:

```
<!ELEMENT   Fussnote    - -         (#PCDATA) >
<!ATTLIST   Fussnote    HyTime      NAME        #FIXED      „clink“
                        id          ID          #REQUIRED
                        linkend     IDREF       #REQUIRED >
```

### 2.3.6 Präsentation von HyTime Dokumenten

Zur Präsentation eines HyTime Dokumentes soll eine HyTime Engine in das Anwendungsprogramm eingefügt werden. Will eine Anwendung ein HyTime-Dokument lesen, startet diese die HyTime Engine, die einen SGML-Parser aufruft. Der Parser gibt alle aus dem Dokument gewonnenen Informationen weiter an die Anwendung, wobei die Kontrolle beim Parsen bei der Anwendung liegt. Die HyTime-Engine erzeugt aus den ausgelesenen Daten eigene Datenstrukturen, die weiter von der Anwendung genutzt werden können. Nach dem Parsen des HyTime-Dokumentes bestimmt wiederum die Anwendung, welche Aktionen die HyTime-Engine durchzuführen hat [11]. Die folgende Abbildung 7 veranschaulicht dies. Aufgrund der relativ hohen Komplexität von HyTime existieren bis heute allerdings noch keine kommerziellen HyTime-Engines.

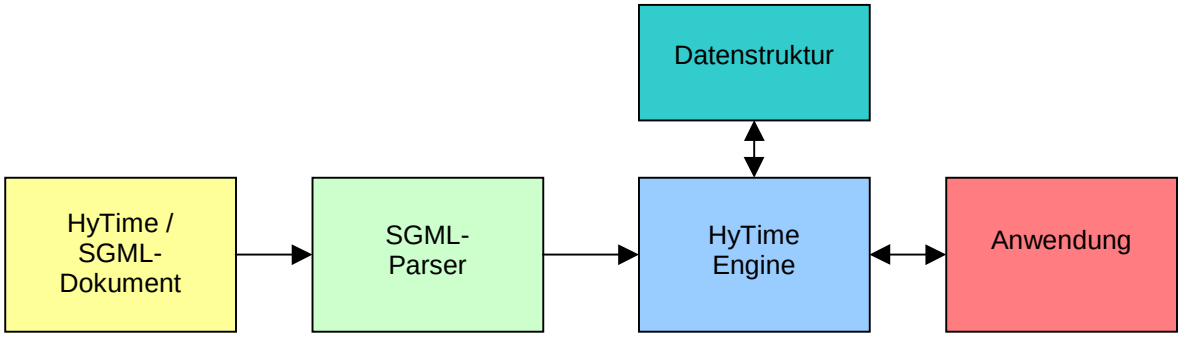


Abbildung 7
Präsentation eines HyTime Dokumentes [11]

## 2.4 MHEG-5

Das digitale Fernsehen ist immer weiter auf dem Vormarsch. In den USA schon längst Standard, hat es hier in Europa erst in wenigen Ländern (Großbritannien) Einzug gefunden. Doch auch in den anderen Ländern Europas wird an digitalen Systemen gefeilt.

Um den Datenaustausch zwischen einem TV-Anbieter und den TV-Geräten, insbesondere den „Set-Top Boxen“ zu ermöglichen und zu vereinfachen, werden Standards benötigt, um eine schnelle und einfache Verbreitung des digitalen Angebots zu gewährleisten. MHEG ist so ein Standard.

Von der ISO wurden bereits bekannte Standards zur Kompression von Bildern oder Videoströmen definiert. MHEG, von der *ISO / IEC Multimedia and Hypermedia information coding Expert Group*

erstellt, ist hingegen ein Standard (ISO 13522), der sich nicht mit der Beschreibung von Medienobjekten beschäftigt, sondern mit deren statischer, zeitlicher und räumlicher Beziehung zueinander. Mittels MHEG soll es möglich sein, komplette Multimedia-Anwendungen mittels einer bestimmten Kodierung portabel und hardware-unabhängig zu machen. So soll eine mit einem beliebigen Autorensystem erstellte Anwendung mittels eines „MHEG-Players“ auf unterschiedlichen Systemen, insbesondere (ab MHEG-5) auf sogenannten „*minimal resource systems*“ (CPU mit 40 MIPS, 4 MB RAM), lauffähig sein. Dazu wird mit einem Autorensystem eine endgültige und ausführbare Anwendung erstellt, die mittels eines Interpreters relativ schnell umgesetzt werden kann. Eine Änderung der ausführbaren Anwendung ist dann nicht mehr möglich. MHEG entspricht daher am ehesten einem Ausgabeformat, vergleichbar mit Postscript.

### 2.4.1 Die einzelnen Stufen von MHEG

Im folgenden sollen einige wichtige Stufen in der Entwicklung von MHEG kurz erläutern werden:

MHEG-1 ist die *kodierte Repräsentation von Multimedia und Hypermedia Objekten*. Es definiert ein generelles Framework für ein Format zum Austausch zwischen Applikationen. MHEG-Objekte werden von Autorensystemen für MHEG-Anwendungen instanziiert. Diese Objekte werden dann zwischen dem (kommerziellen) Anbieter und dem Endsystem beim Benutzer ausgetauscht.

In MHEG-1 definiert die Klassenhierarchie und die Kodierung der MHEG-Klassen. Die in MHEG-1 definierten Klassen dienen für verteilte, interaktive Multimedia-Präsentationen. Sogenannte *Events* könnten dabei ausgelöst werden durch Timer, Eingabegeräte des Benutzers, eintreffende Objekte, bestimmte in der Anwendung eingebettete Zeitpunkte, Zustandsübergängen von MHEG-Objekten oder Aktivitäten des darstellenden Systems. Durch Verbindung mit diesen *Events* mit einer Liste von Aktionen, die beim jeweiligen *Event* ausgelöst werden, sobald dieser *Event* eintritt, wird die Objekt-Interaktion beschrieben.

MHEG-2 sollte ein alternative Kodierung von MHEG-Objekten ermöglichen. Aufgrund des geringen Interesses wurde MHEG-2 aber offiziell nicht fertig gestellt. So war geplant die Klassen in SGML zu definieren, es blieb aber bei ASN.1 (Abstract Syntax Notation, ISO), mit der seit MHEG 1 die Klassen definiert werden..

MHEG-3 definiert Programmier-Erweiterungen für MHEG-1. Damit soll die Unterstützung einer Scriptsprache in Kombination mit MHEG-1 ermöglicht werden. Dazu wurde eine „Virtual Machine“ definiert, auf der Variablen definiert werden können und arithmetische und logische Operationen möglich sind.

MHEG-5 zielt direkt auf die „*minimal resource systems*“ ab, insbesondere Set-Top-Boxen für interaktives, digitales Fernsehen. Zusammen mit dem „Digital Audio-Visual Council“ (DAVIC), definierte die MHEG eine neue Klassenhierarchie, basierend auf MHEG-1. Die Klassenhierarchie von MHEG-5 ist nicht abwärts kompatibel. Als neues Konzept wurden in MHEG-5 zum Beispiel plattformabhängige Interface-Komponenten eingeführt, wie „Scrollbars“, das Layout von Buttons oder Interfaces für den Zugriff auf systemspezifische Funktionen.

### 2.4.2 Die Klassenstruktur von MHEG-5

#### Die Application-Klasse:

Eine MHEG-Applikation wird aus einem Objekt der Klasse *Application* erstellt. Das Objekt, eine Instanz der Klasse Application, definiert einen Einstiegspunkt für eine Reihe von MHEG-Szenen-Objekten. Das Application-Objekt besitzt zusätzlich Variablen, zum Beispiel um Benutzereingaben zu

speichern, sowie gemeinsame Interface-Objekte, die von allen Szenen einer Applikation genutzt werden können.

## Die Klasse „Scene“:

MHEG-5 besitzt ein seiten-orientierte Darstellungskonzept, das von der Klasse Scene realisiert wird. Sobald ein Übergang von einer zu einer anderen Szene erfolgt, werden alle Elemente der vorangegangenen Szene entfernt und die entsprechenden Ressourcen freigegeben.

## Die „Scene Component Objects“:

Jede Szene beinhaltet eine Anzahl von Objekten der Klasse *Ingredients*, die wie folgt unterteilt werden können: „Presentable“ ist eine Überklasse für alle Klassen, deren Objekte sichtbar für den Benutzer dargestellt werden können, wie Bilder, Texte, Videos, Buttons und Eingabefelder. „Links“: Links besitzen einen Trigger, der einen Event auslöst. Das Auftreten eines Events führt zum Ausführen einer bestimmten Aktion („Action“). Objekte der Klasse „Procedure“ dienen als Schnittstelle zwischen der Applikation und den plattformabhängigen Geräten, um z.B. bei einer bestimmten Set-Top-Box einen bestimmten Kanal einzustellen. Die folgende Abbildung 8 zeigt die MHEG-5 Klassenstruktur:

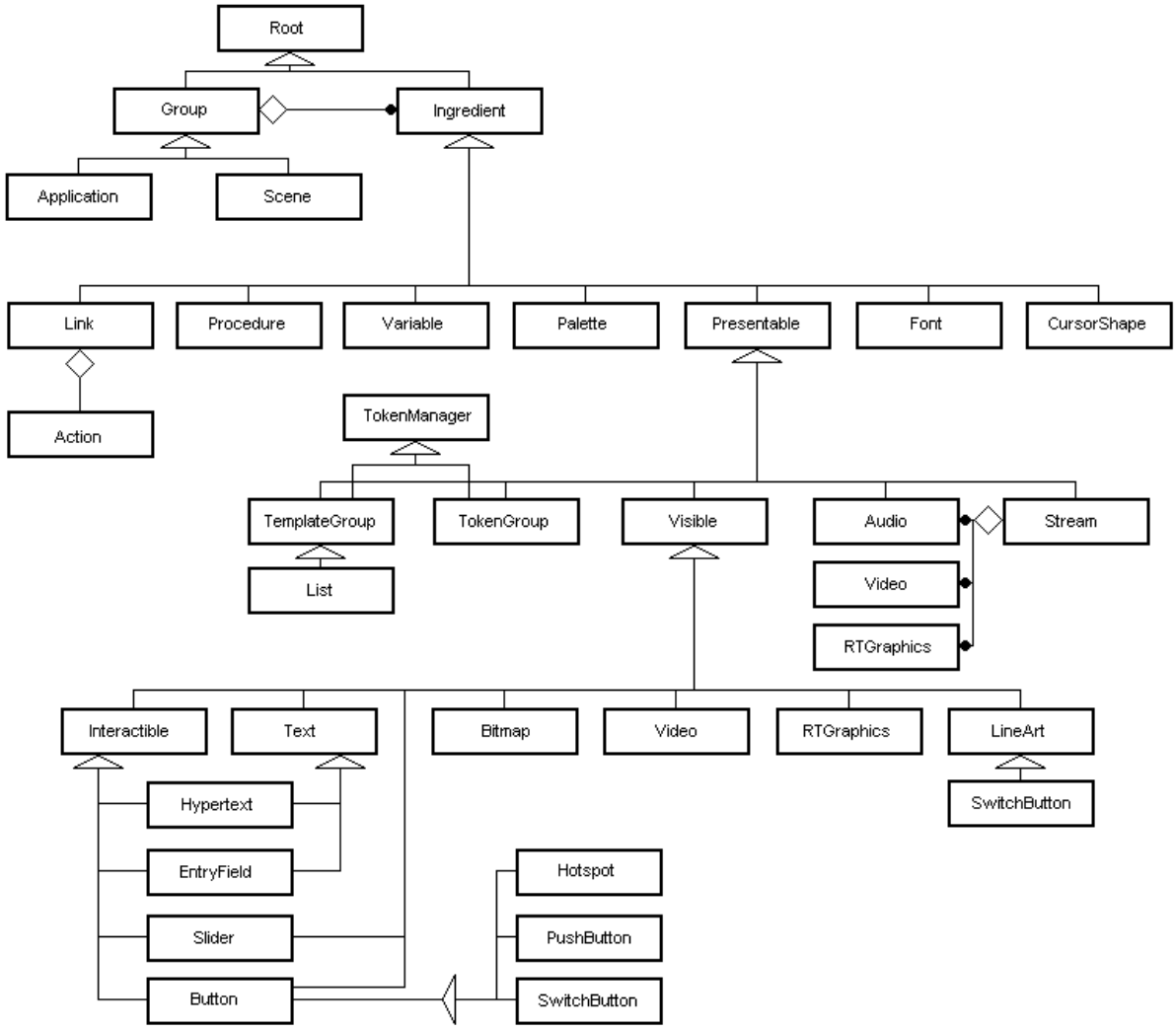


Abb. 8
MHEG-5 Klassenstruktur [2]

### MHEG-5 Terminal Struktur

Abbildung 9 stellt die vereinfachte Struktur eines MHEG-5 Terminals dar. Das System erhält die Objekte (MHEG-Objekte) und Daten (JPEGS, MPEGS, etc.) über das Netzwerk mittels eines Zugriffprotokolls (z.B. MPEG/DSM-CC [2]). Während die Daten direkt an das *Presentation System* durchgereicht werden, werden die MHEG-5-Objekte im ASN.1 Decoder decodiert, in interne Objekte umgewandelt und im *Object Area* der *Engine* gespeichert. Eintreffende Link-Objekte werden im *Link-Prozessor* gespeichert. Darstellbare Objekte werden über den Channel „*Presentation Control*" an das *Presentation System* geleitet, um es dem Benutzer darzustellen. Sobald ein Trigger eines Links ausgelöst wird, wird die zu auslösende Aktion in der *Action Queue* eingereiht und schließlich ausgeführt. Ereignisse des *Presentation Systems* werden automatisch in der Queue eingereiht.

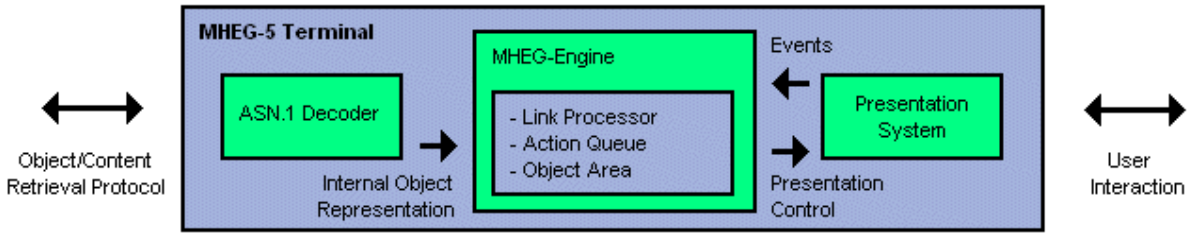


Abb. 9
MHEG-5 Terminal Struktur [2]

### 2.4.3 MHEG-6

MHEG-6 erweitert MHEG-5 um die Möglichkeit, Java-Programmcode innerhalb einer MHEG-5 Applikation auszuführen. Zusätzlich wird in MHEG-6 die Architektur beschrieben, die sich aus der MHEG-5-Engine und einer Java Virtual Machine (dem Java Bytecode-Interpreter) zusammensetzt. Mittels eines Class-Mappers werden MHEG-Objekte zu Java-Objekten umgewandelt und umgekehrt. Auf diese Weise kann mit Java auf MHEG-Objekte zwecks weiterer Verarbeitung zugegriffen werden.

MHEG-5 findet heute bereits Verwendung im terrestrischen Fernsehen in Großbritannien.

## 2.5 $Z_YX$

Bei der Entwicklung eines computer-basierten Ausbildungssystems im Bereich der Medizin (siehe Kapitel 3.1) sollte die multiple Wiederverwendung von Multimedia-Daten für unterschiedliche Ausbildungsszenarien möglich sein. Heutige existierende Modelle und Formate, wie HTML, MHEG, HyTime oder XML genügten den Anforderungen nicht. So sollten Dokumente (Text, Bilder, Videos, etc.) wieder verwendet werden können, sowohl Teile, als auch das komplette Dokument. Zeitliche Abhängigkeiten und Interaktionsmöglichkeiten, vorgegeben durch den Autor, sollten mehrfach genutzt werden können. Mittels Klassifizierung und Indexierung sollen die Suche und Identifikation von Komponenten ermöglicht werden. Interaktionen des Benutzers sollten möglich sein, sowohl beim Navigieren durch die Dokumente, als auch beim Ändern des Designs. Zusätzlich sollten benutzerspezifische Charakteristika eingefügt werden können, so dass sich die Darstellung zum Beispiel entsprechend der fachlichen Kompetenz des Benutzers anpassen lässt. Alle Dokumente sollen zudem auf einer heterogenen Software- und Hardwareumgebung dargestellt werden können. Um diesen Anforderungen gerecht zu werden, entwickelte eine Reihe von Informatikern [26] ein neues Hypermedia-Modell, auf dem die Entwickler des Ausbildungssystems aufbauen konnten, das $Z_YX$-Modell [26].

Das $Z_YX$-Modell beschreibt ein Multimedia-Dokument als einen Baum. Die Knoten dieses Baumes stellen die „Präsentationselemente“ dar, die Kanten verbinden diese Elemente zu einer hierarchischen Struktur. Jedes dieser Elemente hat einen Verbindungspunkt zu einem hierarchisch höher gelegenen Element, sowie eine oder mehrere Variablen, an denen sich wiederum einzelne Elemente befinden können. Einzelne Elemente können dabei Medien oder Teile von Medien enthalten, aber auch Informationen über zeitliche, semantische Abhängigkeiten oder das Layout. Eine Gruppe von Elementen kann zu komplexen Medien-Elementen zusammengesetzt werden, wobei bei den eingebundenen Elementen nicht alle Variablen belegt werden müssen. Auf diese Weise können im Nachhinein weitere Elemente bei der Wiederverwendung hinzugefügt werden. Abbildung 10 veranschaulicht diesen Zusammenhang.

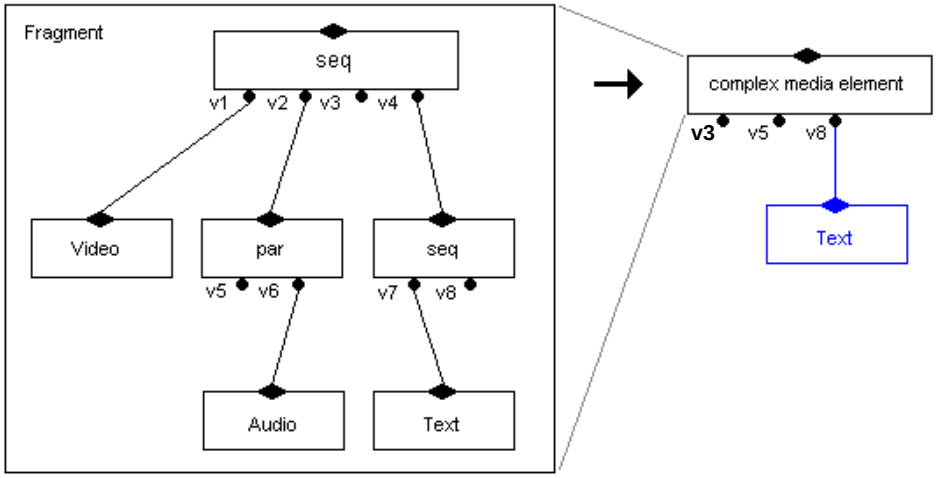


Abb. 10
Elemente in $Z_YX$

Zusätzlich zu den Präsentationselementen, stehen in ZYX eine ganze Reihe von anderen Elementen für unterschiedliche Aufgaben zur Verfügung. Angefangen von „*Sequential Elements*“ (alle an den Variablen angehängten Elemente werden sequentiell abgespielt) von „*Parallel Elements*“ (alle an den Variablen angehängten Elemente werden parallel abgespielt), über „*Projector Elements*“ bis hin zu Zeitelementen, womit Verzögerungen oder Schleifen in Präsentationen ermöglicht werden können. Auf die Funktionsweise der einzelnen Elemente soll an dieser Stelle jedoch nicht genauer eingegangen werden. Siehe dazu [26].

# 3 Anwendungen

## 3.1 Das Cardio-O-Projekt

Beim Cardio-Op-Projekt handelt es sich um die Entwicklung eines computer-basierten Ausbildungssystems für die Herzchirurgie, welches die multiple Wiederverwendung von Multimedia-Daten in unterschiedlichen Ausbildungsszenarien, sowie die flexible Komposition von Inhalten für verschiedene Benutzergruppen zum Ziel hat [25, 28].

Gerade in diesem Bereich in der Medizin entwickeln sich Operationstechniken und Technologien relativ schnell, was dazu führt, dass sich alle Beteiligten, angefangen vom operierenden Arzt, über die Studenten bis hin zu den Technikern, ständig fortbilden müssen, um die komplexen Vorgänge zu verstehen. Aber auch die Patienten wollen oftmals detaillierte Informationen über ihre Krankheiten und mögliche Behandlungen erhalten.

Mitarbeiter der Universität Ulm und Heidelberg, sowie die Entec GmbH [25] entwickelten ein Anwendungsprogramm, das es erlaubte Multimedia-Daten in unterschiedlichen Szenarien wiederzuverwenden und den Inhalt entsprechend der fachlichen Erfahrung des Benutzers (Studenten, Ärzte, Patienten) darzustellen. Abbildung 11 zeigt dieses Programm.

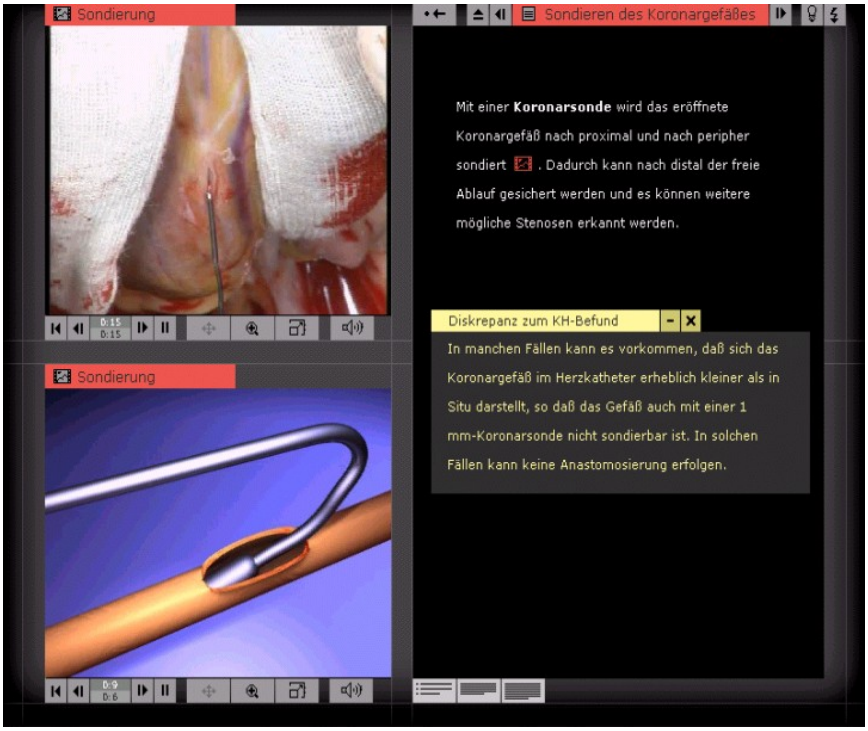


Abb. 11 - Screenshot der Anwendung [25]

Dazu wurde eine entsprechende Unterstützung durch eine Datenbank-Technologie benötigt, um die Verwaltung, die Suche und die Bereitstellung von unterschiedlichen Medientypen zu gewährleisten. Obwohl es in dieser Richtung bereits viele Entwicklungen gegeben hat, haben sich die Forschungsarbeiten in der Regel darauf beschränkt, nur Medientypen einer bestimmten Art, beispielsweise Videos oder Bilder, zu unterstützen. So wurde das QVID System entwickelt, welches es erlaubte, Sets von „Frames“ als Videoobjekte zu definieren, auf die mittels der Abfragesprache *VideoSQL* [24] zugegriffen werden konnte. So lange Applikationen nur einen Typ von Medien benötigen, zeichnen sich keine Probleme ab. Sollen aber mehrere Typen unterstützt werden, so wird die Verwaltung, die Suche und Bereitstellung der Daten erheblich kompliziert. Mediendaten können auf Webservern, Fileservern oder als binäre große Objekte in Datenbanken gespeichert werden. Jeder dieser Speicher hat seine eigenen Zugriffsmethoden und eigene Arten, die Metadaten zu speichern.
Zu diesem Zwecke wurde ein „*Media Integration Blade (MIB) DataBlade Module*“ für das objekt-relationale DBMS Informix entwickelt. Das MIB etablierte dazu eine Zwischenschicht zwischen der Anwendung und den einzelnen medientypen-spezifischen bereits zur Verfügung stehenden *DataBlade* Modulen. Diese Zwischenschicht („Information Layer“) bietet eine homogene Ansicht auf die technischen Metadaten und die unterschiedlichen Medientypen. Auf die genaue Funktionsweise des MIB wird hier jedoch nicht näher eingegangen (siehe [24]).

## System-Übersicht

Der Autor- und Produktionsprozess einer Präsentation durchläuft folgende Schritte:

1. Der Unterrichtsinhalt, Text-Struktur, Hypermedia-Funktionen, zu benutzende Medien werden im digitalen Storyboard festgehalten.

2. Die einzelnen Medien, wie Videos, Grafiken oder 3D-Modelle werden erstellt (Media-Produktion) und gespeichert.
3. Mit Autorensystemen, die mit dem digitalen Storyboard als Basis arbeiten, werden Präsentationen erstellt. Mittels eines Media-Browsers können alle in der Datenbank vorhandenen Medien durchsucht und eingebunden werden.
4. Die fertige Präsentation wird in der Datenbank zur weiteren Verwendung abgelegt. Dabei werden die Dokumente nach dem $Z_YX$-Modell (siehe 2.5) in der Datenbank gespeichert.
5. Letztendlich stellt die Präsentations-Engine die Multimedia-Dokumente entsprechend dem Benutzer da. Der Benutzer kann in dieser Stelle mit dem Dokument interagieren.

Abbildung 12 stellt die On-Line-Architektur des Cardio-Op dar.

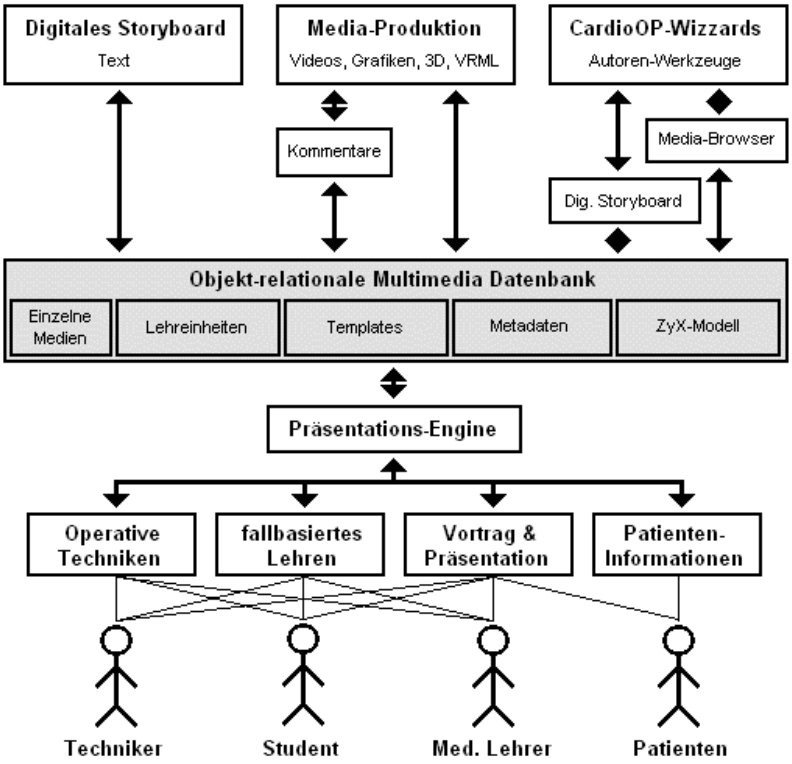


Abb. 12
Cardio-OP [25]

## 3.2 GLASS

Das „Globally Accessible Services (GLASS) System“[17] ist ein Prototyp für das digitale Fernsehen und basiert auf dem MHEG-Standard. Das GLASS-System beinhaltet folgende Komponenten eines digitalen Multimedia-Systems: Clients für den Endbenutzer, Application Sever, Video-Server, System-Management und Gateways zu anderen externen Services (eMail, WWW, FAX, etc.). Typische Applikationen für das GLASS-System sind „Video-on-demand“, TV-Übertragung (Broadcast), Multimedia-Präsentationen, Spiele und Tele-Shopping. GLASS-Clients laufen auf unterschiedlichen Hardware- und Software-Systemen (DEC Alpha, Intel 80x86, Sun Sparc; Linux, MacOs, Windows, OS/2, Solaris).

Abbildung 13 zeigt die Komponenten die im GLASS-System implementiert wurden. Das GLASS End-System (gelb dargestellt) besteht aus folgenden Modulen:

- MHEG Engine
- Control Agent
- User Interface Agent
- Presentation Objects

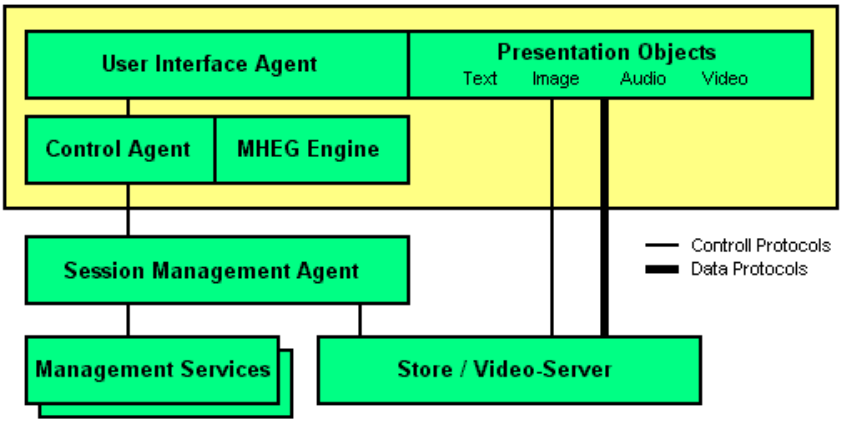


Abb. 13 - Aufbau GLASS-System

Die *MHEG-Engine* ist die zentrale Recheneinheit im Client. Sie erhält und interpretiert MHEG-Objekte und kontrolliert das Benutzer-Interface. Der *Control Agent (CA)* übernimmt die Kommunikationsaufgaben der *MHEG-Engine*. Er stellt eine Verbindung zum *Session Management Agent* her und initialisiert die MHEG-Engine. Zudem ist der CA für das Fehlerhandling verantwortlich. Der *User Interface Agent* (UIA) ist verantwortlich für das verwalten der einzelnen *Presentation Objects*, wie Texte, Bilder, Audio oder Video. Bei den *Presentation Objects* (PO) wird zudem unterschieden nach sichtbarer, hörbarer und interagierbarer Funktionalität. POs handhaben die Präsentation der Datenobjekte und die Benutzerinteraktion. Sie sind dafür verantwortlich die Multimedia-Daten dem Benutzer darzustellen. POs werden durch den UIA bei Aufforderung durch die MHEG-Engine erstellt, modifiziert und zerstört.

Auf Seite des Servers wird zwischen zwei Komponenten entschieden: Zum einen dem Application Server, mit spezifischen Aufgaben zu einer bestimmten MHEG-Session (einer Verbindung zwischen Server und Client), wie „Session Management“ oder „MHEG-Object Retrieval“; zum anderen dem Media-Server, auf dem sich die eigentlichen Daten (Bilder, Videos, etc.) befinden.

# 4 Ausblick

Diese Ausarbeitung sollte einen kleinen Überblick über einige Hypertext- und Hypermedia-Modelle geben. Hypertext-Modelle werden aufgrund höherer Bandbreiten und größerer Datenbank- und Festplattenkapazitäten wohl immer weiter in den Hintergrund treten und höchstens als Grundlage für neue Hyper*media*-Modelle dienen, wie zum Beispiel beim AHM.

Hypermedia-Modelle hingegen dürften, insbesondere MHEG, in naher Zukunft stärker an Bedeutung gewinnen, besonders im Bereich des digitalen Fernsehens. In Deutschland wird allerdings eher MHP (*Multimedia Home Plattform*), vom European Telecommunications Standards Institute (ETSI) spezifiziert und normiert, Einzug halten. Eingesetzt wird bei MHP eine erweiterte Variante von Personal Java als Settop-Box-Betriebssystem (genauere Informationen und Spezifikationen zu MHP

lagen dem Autor allerdings bei Erstellung dieser Ausarbeitung noch nicht vor). Dennoch hat auch MHEG noch Chancen, neben Großbritannien auch in anderen europäischen Ländern eingesetzt zu werden, da es bereits erste Ansätze gibt, MHEG in MHP zu unterstützten [14].

Aber auch Modelle wie $Z_YX$ dürften für zukünftige Hypermedia-Anwendung im Bereich der Lernprogramme von großen Interesse sein, da gerade heutzutage die Wiederverwendung aufgrund der enorm hohen Masse an Informationen immer mehr an Bedeutung gewinnen wird.

# Referenzen


[1] John F. Buford. Evaluation of a Query Language for Structured Hypermedia Documents. 1995
[2] Peter Hofmann. MHEG-5 and MHEG-6: Multimedia Standards for Minimal Resource Systems. Technical University Berlin, Germany. April 1996
[3] Susanne Boll, Wolfgang Klas, Utz Westermann. Multimedia Document Models - Sealed Fate or Setting Out for New Shores? University of Ulm, Germany. June 1999
[4] Lloyd Rutledge, Jacco van Ossenbruggen, Lynda Hardman, Dick Bulterman. Cooperative Use of MHEG and HyTime in Hypermedia Enviroments. September 1997
[5] Dick C.A. Bulterman. Head, Multimedia and Human-Computer Interaction: Modelling Support for Network-Based Multimedia Presentation.
[6] Lynda Hardman, Dick C.A. Bulterman, Guido van Rossum. The Amsterdam Hypermedia Model: extending hypertext to support real multimedia. 1991
[7] Dick C.A. Bulterman. Specification and Support of Adaptable Networked Multimedia. 1993
[8] A. Hatzimanikatis, I. Gaviotis, D. Christodoulakis. Distributed Documents: an architecture for open distributed hypertext. March 1994
[9] Robert van Liere, Guido van Rossum, D.C.A. Bulterman. A Structure for Transportable, Dynamic Multimedia Dokuments. 1991
[10] Susanne Boll, Wolfgang Klas, Utz Westermann. A Comparison of Multimedia Dokument Models Concerning Advanced Requirements. University of Ulm. Germany
[11] Dietrich Boles. Multimedia-Systeme. 1998
[12] C. Bouras, V. Kapoulas, D. Miras, V. Ouzounis, P. Spirakis. An Architecture for Interactive Distributed Multimedia Information Services
[13] Ruben Gonzales, Peter Beadle, Hught Bradlow, Rei Safavi-Naini. The Design of Distributed Hypermedia Information Systems
[14] J.C. Newell, I. Childs. Strategies for supporting the DVB MHP in an MHEG-5 enviroment
[16] Deepak Murthy, Aidong Zhang. WebView: A Multimedia Database Resource Integration and Search System over Web. 1997
[17] H. Cossmann, C. Griwodz, G. Grassel, M. Pühlhöfer, M. Schreiber, R. Steinmetz, H. Wittig, L. Wolf. Interoperable ITV Systems based on MHEG.
[18] Antoine Rizk, Francis Malezieux, Alain Leger. Using the MHEG standard in the hypermedia system Multicard. 1994
[19] Seungtaek Oh, Yung Yi, Seunghoon Jeong, Yanghee Choi. Experiments with MHEG Player/Studio: An Interactive Hypermedia Visualization and Authoring System
[20] Stephen R. Mounce. A Brief Discussion of Standard Music Description Language
[21] Lynda Hardman, Dick C.A. Bulterman. Using the Amsterdam Hypermedia Model for Abstracting Presentation Behavior. November 1995
[22] Franz Burger, Gerald Quirchmayr, Siegfried Reich, A Min Tjoa. Using HyTime for Modeling Publishing Workflows.
[23] M. Tamer Özsu, Paul Iglinski, Duane Szafron, Sherine El-Medani, Manuela Junghanns. An Object-Oriented SGML/HyTime compliant Multimedia Database Management System
[24] Utz Westermann, Wolfgang Klas. Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets. 1999
[25] R. Friedl, W. Klas, T. Rose, O. Gödje, M. Preisack, J. Tremper, C.F. Vahl, K.J. Quast, A. Hannekum. Individualized Learning and Teaching in Heart Surgery: The Cardio-OP-Project.
[26] Susanne Boll, Wolfgang Klas. ZYX – a semantic model for multimedia documents and presentations. 1999
[27] Carsten Oberscheid. SGML, HyTime und DSSSL. 1996
[28] Das Cardio-Op-Projekt: http://www.cardio-op.de
[29] Isabel Scheiding. Modellierung von HyTime Architectural Forms als Grundlage für das Annotieren strukturierter Dokumente. 1996