# Repräsentation flexibel wiederverwendbarer multimedialer Dokumente in einem DBMS

Diplomarbeit an der Universität Ulm

Fakultät für Informatik


vorgelegt von:

Gerd Utz Westermann

1. Gutachter: *Prof. Dr. Wolfgang Klas*
2. Gutachter: *Prof. Dr. Peter Dadam*

1998

**Danksagung**

Ich danke Susanne Boll für ihre hervorragende und aufopferungsvolle Betreuung während der Erstellung dieser Diplomarbeit. Ihre Tür stand für mich immer offen und sie half mir über die schwierigen Phasen dieser Arbeit mit Rat und Tat hinweg. Weiterer Dank gilt auch Prof. Dr. Klas, mit dem ich im Verlauf dieser Arbeit einige fruchtbare Diskussionen führen konnte. Er war es, der mein Interesse an dem Themengebiet der multimedialen Informationssysteme erst geweckt hat.

Name: Gerd Utz Westermann Matrikelnummer: 312280

**Erklärung**

Ich erkläre, daß ich die Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 11. September 1998

**Abstract**

This diploma thesis has evolved from the research project "Gallery of Cardiac Surgery" which aims at the development of a network-based database-driven repository of multimedia documents in the domain of cardiac surgery. It is required that these documents or parts of them – called *fragments* – can be flexibly composed to new individualized documents. On the one hand this composition can be performed by authors of document content. On the other hand the information system must be able to compose documents on-the-fly by user request. This functionality has to be supported by the underlying multimedia document model. This diploma thesis shows that in particular a multimedia document model has to offer support for *reusability*, for *interaction*, for *adaptation* to user specific needs, and for the *presentation-independent* description of multimedia document content. It is illustrated that the multimedia document standards MHEG-5, HyTime, and SMIL do not sufficiently support these features. Therefore, the new ZYX document model has been developed which allows for fine-grained reuse of document content, for software-based search and selection of reusable fragments, for modelling of interaction, for adaptation of document content to user specific needs, and for an easy conversion to other multimedia document standards. A database capable of managing ZYX documents is implemented as a DataBlade module for the ORDBMS Informix Dynamic Server/ Universal Data Option. Server-based functionality is provided to allow for easy access and creation of ZYX document content, for performing adaptation, and for conversion of ZYX documents to a XML-based format.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einleitung

Multimediale Dokumente sind Dokumente, die mehrere Medien als informationstragende Bestandteile enthalten und diese zueinander in Beziehung setzen. Richtig eingesetzt, erhofft man sich durch die Verwendung unterschiedlicher Medien und der damit verbundenen intensiven Stimulierung der menschlichen Sinnesorgane, eine für den Menschen adäquatere Weise der Informationsvermittlung als durch herkömmliche Textdokumente zu erreichen. Ein gutes Beispiel hierfür ist ein multimediales Dokument über eine Herzoperation, das neben dem eigentlichen Text ein Video der Operation enthält und abhängig vom Fortschritt der Präsentation des Videos eine Schemazeichnung über den jeweils als nächstes durchzuführenden Operationsschritt anzeigt. Ein Autor eines solchen multimedialen Dokuments benötigt ein Ausdrucksmittel, um das Dokument mit den in ihm enthaltenen Medien und deren Beziehungen untereinander beschreiben zu können: ein sogenanntes multimediales Dokumentmodell.

Es gibt einige Bemühungen, ein umfassendes Dokumentmodell für multimediale Dokumente zu standardisieren. Dieses würde den Austausch und die Verwaltung von multimedialen Dokumenten erheblich vereinfachen. Beispiele für solche multimedialen Dokumentstandards sind MHEG-5 [ISO95, ISO96], HyTime [DD94] und SMIL [PBD+98]. Da aber der Prozeß einer Standardisierung sehr langwierig ist und die mit der Standardisierung beauftragten Komitees meist unterschiedliche Zielgruppen mit verschiedenen Zielsetzungen repräsentieren, ist noch kein Standard vorhanden, der zum einen den neuesten Entwicklungen auf dem Gebiet der multimedialen Dokumente Rechnung trägt und zum anderen universell einsetzbar ist. So ist der HyTime-Standard auf die Strukturierung und Verknüpfung von Dokumentbestandteilen mit multimedialen Inhalten ausgelegt, bietet aber keine Beschreibungsmöglichkeiten für Interaktionen seitens des Lesers. Im Gegensatz dazu bietet der MHEG-5 Standard eine Vielzahl an Interaktionsformen. Jedoch wurde er von der Unterhaltungselektronikindustrie speziell für Kiosk und Video-On-Demand Anwendungen auf Set-Top-Boxen mit geringer Rechenleistung ausgelegt, so daß die Strukturierungsmöglichkeiten der multimedialen Dokumente auf die Anforderungen dieses Anwendungsgebietes eingeschränkt sind. Es ist also kein umfassender Dokumentstandard vorhanden und es ist auch fraglich, ob ein solcher jemals existieren wird.

Man kann nun überlegen, zur Verwaltung von multimedialen Dokumenten und den in ihnen enthaltenen Medien diese in einer multimedialen Datenbank abzulegen. Da es keinen umfassenden Dokumentstandard gibt, sollte eine universell einsetzba-

re multimediale Datenbank in der Lage sein, Dokumente verschiedener Standards abzulegen und zu verwalten. Eine interessante Idee ist nun, daß diese Dokumente und deren Bestandteile nicht isoliert in einer Datenbank liegen, sondern statt dessen flexibel miteinander, auch dokumentstandardübergreifend, verknüpfbar sein sollen. Die Verwirklichung dieses Gedankens ist eines der Ziele des Forschungsprojektes „Galerie der Herzchirurgie", welches von der Abteilung DBIS der Universität Ulm und dem FAW Ulm in Zusammenarbeit mit den Universitätskliniken in Ulm und Heidelberg, den Verlagen dpunkt und Barth sowie der Firma ENTEC aus St. Augustin durchgeführt wird und vor dessen Hintergrund diese Arbeit entstanden ist. Ziel dieses Projektes ist die Erstellung eines multimedialen Informationssystems, das Bestandteile multimedialer Dokumente rund um das Themengebiet der Herzchirurgie in einer multimedialen Datenbank verwaltet und deren Wiederverwendung in neuen multimedialen Dokumenten erlaubt. Hierzu bedarf es eines multimedialen Dokumentmodells, was dieses unterstützt.

## 1.1 Ziele dieser Arbeit

Die Ziele dieser Arbeit bestehen darin, zu untersuchen, welche konkreten Anforderungen an ein multimediales Dokumentmodell gestellt werden, wenn dieses die flexible Wiederverwendung von Dokumenten oder deren Bestandteilen erlauben soll. Nach der Analyse dieser Anforderungen ist zu überprüfen, inwieweit vorhandene Standards wie MHEG, HyTime und SMIL diesen Anforderungen gerecht werden. Sollten die untersuchten Dokumentstandards dieser Überprüfung nicht standhalten, so ist ein Dokumentmodell zu definieren, welches die gestellten Anforderungen erfüllt. Schließlich ist eine Datenbank in Form eines DataBlades für den Informix Dynamic Server/ Universal Data Option [Inf97b] zu implementieren, die Dokumente des gewählten Standards bzw. des definierten Dokumentmodells verwalten kann.

## 1.2 Gliederung

Die Arbeit ist im wesentlichen entlang der oben genannten Zielsetzungen gegliedert. Bevor jedoch mit der Analyse der Anforderung an ein flexibles multimediales Dokumentmodell in Kapitel 3 begonnen wird, werden in Kapitel 2 zunächst die grundlegenden Begriffe rund um die Themengebiete „Multimedia" und „multimediale Dokumente" erläutert. In Kapitel 4 werden die in der Praxis relevanten multimedialen Dokumentstandards MHEG-5, HyTime und SMIL näher vorgestellt und auf die Erfüllung der Anforderungen hin überprüft. Da diese die Anforderungen nicht zufriedenstellend erfüllen, definiert Kapitel 5 das ZYX-Dokumentmodell, welches sich zur Repräsentation flexibler multimedialer Dokumente eignet. Kapitel 6 beschreibt die Implementierung einer Datenbank zur Ablage von Dokumenten des ZYX-Dokumentmodells in Form eines Datablades für den Informix Dynamic Server/ Universal Data Option und Kapitel 7 faßt schließlich die Ergebnisse dieser Arbeit zusammen und liefert einen Ausblick.

## 1.3 Anforderungen an den Leser

Der Leser dieser Arbeit sollte sich mit den Konzepten und Begriffen des objektorientierten Paradigmas auskennen. Außerdem werden Kenntnisse über relationale

Datenbank-Management-Systeme verlangt.

# Kapitel 2

# Multimedia und multimediale Dokumente

Der Begriff Multimedia hat sich seit Anfang der 90er Jahre zu einem viel gebrauchten Modewort entwickelt. Unterschiedliche Personengruppen wie zum Beispiel Informatiker, Künstler, Soziologen, Jugendliche oder Journalisten verwenden diesen Begriff in unterschiedlichen Zusammenhängen, oft leider mit ebenso unterschiedlichen Bedeutungen. Das Problem liegt darin begründet, daß es keine einheitlich anerkannte Definition von Multimedia gibt. Die Aufgabe dieses Kapitel ist es deswegen, die grundlegenden Begriffe rund um das Gebiet Multimedia so darzulegen, wie sie in dieser Arbeit verwendet und verstanden werden.

## 2.1 Multimedia

Geht man rein von der Etymologie des Wortes Multimedia aus, so bedeutet es ,,viele Medien". Ein Medium ist ein Mittel zur Verbreitung und Darstellung von Information. In der Informationstechnik sind verschiedene Arten von Medien unterscheidbar [SN95]:

- *Perzeptionsmedien* dienen dazu, einem Menschen Information in einer von ihm wahrnehmbaren Art und Weise zu vermitteln. Beispiele hierfür sind Texte, Bilder, gesprochene Sprache und Videos.
- *Repräsentationsmedien* kodieren Information in einer von Computern verarbeitbaren Weise. Bekannte Vertreter dieser Medienart sind der ASCII-Zeichensatz oder der MPEG-2-Standard zur Kodierung von Videofilmen.
- *Präsentationsmedien* bilden die Brücke zwischen Perzeptions- und Repräsentationsmedien. Sie stellen Mittel und Wege zur Verfügung, im Rechner gespeicherte Information dem Benutzer zu präsentieren bzw. neue Information vom Benutzer entgegenzunehmen. Als Beispiel hierfür mögen Bildschirm und Tastatur dienen.
- *Speichermedien und Übertragungsmedien*: Speichermedien, beispielsweise Disketten, Festplatten oder CD-ROM, speichern Information. Übertragungsmedien erlauben den Transport von Information. Prominente Vertreter dieser

Gattung sind Netzwerke und Modems. Da jedoch Speichermedien (z.B. Disketten) auch zur Übertragung von Information genutzt werden können, ist die Grenze hier fließend.

Spricht man von Medien im Zusammenhang mit Multimedia, so sind hauptsächlich Perzeptionsmedien gemeint. Diese lassen sich in zwei Gruppen kategorisieren. *Diskrete Medien* wie Text oder Bilder sind zeitunabhängig. *Kontinuierliche Medien* wie Videos oder Musikstücke hingegen sind veränderlich mit der Zeit.

Definiert man, von der Herkunft des Wortes Multimedia ausgehend, ein multimediales System als ein System, welches mehrere Medien unterstützt, so stellt man fest, daß diese Definition nicht sonderlich befriedigend ist. Gemäß dieser Definition ist nämlich auch eine Textverarbeitung als multimedial anzusehen, die es erlaubt, Grafiken in einen Text einzubinden. Eine solche Textverarbeitung ist aber eigentlich nicht das, was man unter einem multimedialen System verstehen möchte. Steinmetz [SN95] definiert deshalb ein multimediales System folgendermaßen:

*Ein multimediales System ist durch die rechnergesteuerte, integrierte Erzeugung, Darstellung, Speicherung und Kommunikation von unabhängigen Informationen gekennzeichnet, die in mindestens einem diskreten und einem kontinuierlichen Medium kodiert sind.*

Die wesentlichen Punkte dieser Definition sollen noch einmal klar herausgestellt werden:

- Ein multimediales System muß mindestens ein diskretes und ein kontinuierliches Medium unterstützen.
- Die Informationen müssen rechnergesteuert verarbeitet werden. Es ist beispielsweise nicht ausreichend, daß ein System über einen Adapter einen externen Videorekorder startet.
- Die verschiedenen Medien müssen unabhängig voneinander verarbeitet werden können.
- Die einzelnen Medien müssen integriert werden können. Es muß möglich sein, Beziehungen zwischen Medien herzustellen.

Typische Anwendungsgebiete für multimediale Systeme sind Spiele und Unterhaltung, Werbung und Firmenpräsentationen, Lernsoftware, Elektronische Publikationen und Online-Hilfe-Systeme. Man möchte in diesen Gebieten durch die kombinierte Verwendung unterschiedlicher Medien eine erhöhte Stimulanz der Sinne des Konsumenten erreichen. Davon erhofft man sich im Bereich der Unterhaltung eine Steigerung des Erlebnisgefühls, während im Gebiet der elektronischen Publikationen und der Lernsoftware eher die Aufnahmefähigkeit des Konsumenten gesteigert werden soll.

## 2.2 Multimediale Dokumentmodelle

Ein *multimediales Dokument* ist ein Dokument, welches aus Informationseinheiten besteht, die in mindestens einem diskreten und einem kontinuierlichen Medium kodiert sind [SN95]. Ein multimediales Dokument integriert diese Informationseinheiten und bringt sie in einen Zusammenhang, d.h. es definiert Beziehungen zwischen

diesen. Die Darstellung der Informationseinheiten gemäß dieser Beziehungen nennt man *Präsentation* des multimedialen Dokumentes. Die Aufgabe eines *multimedialen Dokumentmodells* ist es, ein Beschreibungsmittel für multimediale Dokumente zur Verfügung zu stellen. Mit ihm erfolgt die Beschreibung der in einem Dokument enthaltenen Informationseinheiten sowie die Beschreibung der Beziehungen zwischen diesen.

Im folgenden werden unter Anlehnung an objektorientierte Terminologie die Informationseinheiten eines multimedialen Dokumentes *Medienobjekte* genannt. Das Medium, in dem ein Medienobjekt kodiert ist, wird als dessen *Medientyp* bezeichnet.

In Anlehnung an [Bol94] lassen sich Beziehungen zwischen Medienobjekten wie folgt klassifizieren:

- *Zeitliche Beziehungen* beschreiben die zeitliche Abfolge der Präsentation der beteiligten Medienobjekte. Ein Beispiel hiefür ist die Beschreibung der gleichzeitigen Ausgabe eines Videoclips mit einer Hintergrundmusik.
- *Gestalterische Beziehungen* legen fest, wie etwas präsentiert wird. So sind unter anderem bei visuellen Medien die Farbtiefe, Bildgröße und der gezeigte Bildausschnitt von Belang, während bei akustischen Medien die Lautstärke eine nicht unwichtige Rolle spielt. Die Beschreibung der Ausgabe eines Hintergrundmusikstückes bei halber Lautstärke eines synchron dargestellten Videos ist ein Beispiel für eine gestalterische Beziehung.
- Während die beiden vorgehenden Beziehungsarten festlegen, *was*, *wann* und *wie* präsentiert wird, läßt sich mit *räumlichen Beziehungen* festlegen, *wo* etwas Präsentiert wird. Ein Beispiel hiefür ist die Beschreibung zweier nebeneinander auf dem Bildschirm dargestellter Bilder.
- *Interaktionsbeziehungen* modellieren die Auswirkungen von Benutzerinteraktionen auf die beteiligten Objekte. Solche Interaktionen können sowohl zeitliche, gestalterische als auch räumliche Auswirkungen auf die Präsentation eines multimedialen Dokumentes haben. Als Beispiel für eine gestalterische Interaktionsbeziehung möge ein Lautstärkeregler für einen Audio-Clip dienen.
- *Indirekte Beziehungen* sind Beziehungen zwischen Medienobjekten, die sich nicht unmittelbar sondern erst zu einem späteren Zeitpunkt auf deren Präsentation auswirken. Man kann sich unter anderem hierunter die Anpassung eines Dokumentes an den Kenntnisstand des Betrachters vorstellen.

Ein multimediales Dokumentmodell sollte Beschreibungsmittel für möglichst viele dieser Beziehungskategorien bereitstellen. Die Menge der von einem Dokumentmodell zur Verfügung gestellten Beschreibungsmittel für zeitliche, räumliche, gestalterische und indirekte Beziehungen sowie für Interaktionsbeziehungen zwischen den im Dokument enthaltenen Medienobjekten wird im folgenden als die *multimediale Funktionalität* des Dokumentmodells bezeichnet.

# Kapitel 3

# Flexible multimediale Dokumente

Wie schon eingangs erwähnt, entstand diese Diplomarbeit vor dem Hintergrund des Forschungsprojektes „Galerie der Herzchirurgie". Die Zielsetzung des Projektes ist die Entwicklung eines netzwerkfähigen, datenbank-orientierten, multimedialen Informationssystems zur Herzchirurgie. Dabei sollen Medienobjekte und Dokumentbestandteile *kontextabhängig* und *modular* zu neuen, multimedialen Dokumenten zusammensetzbar sein.

Kontextabhängig meint, daß auf ein multimediales Dokument verschiedene, zielgruppenabhängige Sichten existieren können. So sollte ein Patient, der sich über eine Bypass-Operation informieren möchte, auf einem eher populärwissenschaftlichen Niveau informiert werden, während einen sich auf eine Prüfung vorbereitenden Studenten Details interessieren. Desweiteren sollte Rücksicht auf die einem Betrachter zur Verfügung stehende Hardware und Netzwerkanbindung genommen werden. Ein Student verfügt zu Hause in der Regel über eine deutlich leitstungschwächere Hardware und Netzwerkanbindung als auf dem Campus.

Modular bedeutet, daß neue Dokumente aus klar definierten, gegeneinander abgegrenzbaren Einheiten zusammengesetzt werden können, den *Fragmenten*. Diese Zusammensetzung soll auch dynamisch, zum Betrachtungszeitpunkt eines Dokumentes, erfolgen können. Es muß möglich sein, sich vom Informationssystem ein multimediales Dokument über ein vom Betrachter bestimmtes Thema zusammenstellen zu lassen. Multimediale Dokumentmodelle, die solche Dokumente unterstützen, werden im folgenden als *flexibel* bezeichnet.

Im Rahmen dieser Aufgabenstellung ergeben sich Anforderungen an ein multimediales Dokumentmodell, welches die flexible, kontextabhängige Erstellung von multimedialen Dokumenten aus Fragmenten gewährleisten soll. Diese Anforderungen werden in den nachfolgenden Abschnitten näher beleuchtet.

## 3.1 Wiederverwendbarkeit

Wenn vorhandene Fragmente modular zu neuen multimedialen Dokumenten zusammengesetzt werden, so ist dies eine Form von *Wiederverwendung*. Ein flexibles

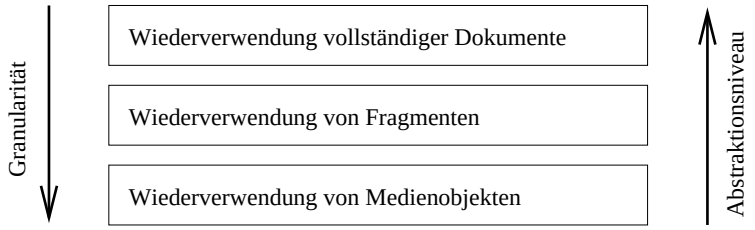


Abbildung 3.1: Ebenen der Wiederverwendbarkeit

multimediales Dokumentmodell muß also Wiederverwendbarkeit erlauben. Dabei ist Wiederverwendbarkeit ein sehr vielschichtiges Konzept. Im folgenden werden nun verschiedene Aspekte von Wiederverwendbarkeit vorgestellt und untersucht.

### 3.1.1 Granularität

Die *Ebene* der Wiederverwendbarkeit [Som96, Weg87] ist ein wesentlicher Aspekt. Die Ebene der Wiederverwendbarkeit bestimmt, was wiederverwendet werden kann und damit auch die Granularität der Wiederverwendbarkeit. Im Rahmen von multimedialen Dokumenten lassen sich mindestens drei Ebenen unterscheiden (siehe Abbildung 3.1):

1. *Wiederverwendung vollständiger multimedialer Dokumente*: Dies ermöglicht die Wiederverwendung von vollständigen multimedialen Dokumenten. Diese Ebene stellt damit Wiederverwendbarkeit auf dem obersten Abstraktionsniveau dar. Problematisch ist jedoch die sehr grobe Granularität ganzer Dokumente. Es ist nicht möglich, nur Teile eines multimedialen Dokumentes wiederzuverwenden. Wenn ein multimediales Dokument nicht exakt den von einem Autor gestellten Anforderungen genügt, kann es nicht wiederverwendet werden. Dadurch kommt es zu unnötigen Redundanzen zwischen Dokumenten, deren Inhalt sich lediglich überschneidet, aber nicht identisch ist.

2. *Wiederverwendung von Fragmenten*: Auf dieser Ebene kann man Teile von multimedialen Dokumenten wiederverwenden. Man faßt Dokumentbestandteile in logische Gruppen, den Fragmenten, zusammen, die dann in verschiedenen Dokumenten benutzt werden können. Durch diese feinere Granularität ist ein hoher Grad an Wiederverwendung und damit an Redundanzfreiheit möglich.

3. *Wiederverwendung von Medienobjekten*: Auf dieser Ebene findet Wiederverwendung auf dem untersten Abstraktionsniveau, den Medienobjekten, statt. Obwohl auf diesem Niveau keine Beziehungen zwischen Medienobjekten berücksichtigt werden, ist die Wiederverwendbarkeit von einzelnen Medienobjekten aufgrund des hohen Speicherbedarfs kontinuierlicher und visueller Medien sehr effektiv.

Um einen optimalen Grad an Wiederverwendbarkeit zu erreichen, sollte ein flexibles multimediales Dokumentmodell Wiederverwendung auf möglichst vielen der oben genannten Ebenen emöglichen.

### 3.1.2 Art der Wiederverwendbarkeit

Nachdem im vorigen Abschnitt behandelt wurde, *was* wiederverwendet werden kann, geht es nun um die Art der Wiederverwendbarkeit, i.e. *wie* etwas wiederverwendet wird. Eine Möglichkeit ist die *identische Wiederverwendbarkeit*. Bei der identischen Wiederverwendbarkeit werden Fragmente mit allen in ihnen spezifizierten temporalen, räumlichen, gestalterischen, indirekten und Interaktionsbeziehungen wiederverwendet. Das bedeutet, daß ein Fragment exakt so präsentiert wird, wie es dessen Autor festgelegt hat.

Dies ist eine sehr natürliche und intuitive Art der Wiederverwendbarkeit, jedoch kann es Anwendungsgebiete geben, wo dieses nicht wünschenswert ist. Firmen zum Beispiel verfügen heutzutage über eine sogenannte „Corporate Identity". Diese gibt Designrichtlinien vor, die in allen Dokumenten der Firma einzuhalten sind (zum Beispiel räumliches Layout, Schriftzüge und Zeichensätze). Das soll beim Konsumenten einen Wiedererkennungseffekt bewirken. So druckt seit mehreren Jahren Daimler-Benz in Tageszeitungen ganzseitige Anzeigen ab, die alle dasselbe Design haben (grauer Hintergrund, spezieller Zeichensatz, Mercedesstern unten zentriert). Wenn eine Firma einen Imagewechsel vollziehen möchte, ist es sehr aufwendig, alle Fragmente an die neuen Designrichtlinien anzupassen, wie es bei identischer Wiederverwendbarkeit der Fall wäre.

Ein anderer Ansatz ist deshalb die *semantische Wiederverwendbarkeit*. Bei dieser Art der Wiederverwendbarkeit beschreibt ein Fragment lediglich seine inhaltliche *Struktur* und keine räumlichen und gestalterischen Beziehungen, die im folgenden als das *Layout* des Fragments bezeichnet werden. Es wird somit nur die Struktur eines Fragments wiederverwendet. Ändern sich die Layout-Richtlinien einer Organisation, werden die Fragmente nicht von den Änderungen betroffen. Die strikte Trennung zwischen Struktur und Layout findet sich u.a. bei SGML- [Gol90] und XML-Dokumenten [BPSM98, BD97, BM98b] wieder.

Da Layout aber ein nicht zu unterschätzendes rhetorisches Mittel [Gra97] bei der Präsentation multimedialer Dokumente darstellt und die Signifikanz und die logische Struktur der dargestellten Information unterstreichen kann, müssen ästhetisch ansprechende Layoutinformationen in einem zusätzlichen Verarbeitungschritt dem multimedialen Dokument und seinen Fragmenten hinzugefügt werden. Dieser Verarbeitungsschritt sollte möglichst automatisiert durchführbar sein.

Ein Lösungsansatz für diesen Verarbeitungsschritt ist das *Constraint-Based Multimedia Layout* [Gra92, Gra95]. Hier wird das Layoutproblem auf ein Constraint-Solving-Problem reduziert. Lösungen für solche Probleme stellt die KI bereit. Andere Ansätze stammen aus dem Bereich der SGML- und XML-Standards. Hier dienen die sogenannten *Style Sheets* zur Festlegung des Layouts eines Dokumentes. Es existieren verschiedene Varianten von Style Sheets. Es gibt zum einen die *Cascading Style Sheets* (CSS) [Tol96, BM98b], die das Layout des Dokuments über eine Menge von statischen Regeln bestimmen. Zum anderen existiert die *Document Style Semantics and Specification Language* (DSSSL) [BM98a, BM98b], die eine berechnungsvollständige, LISP-basierte Sprache ist und es erlaubt, XML- und SGML-Dokumente in ein beliebiges Ausgabeformat wie zum Beispiel Postscript zu konvertieren. Schließlich ist im Rahmen der XML-Standardisierung eine weitere Style-Sheet-Variante vorgesehen, nämlich XSL [ABC+97, BM98b]. Auf SGML und XML wird in Kapitel 4 noch näher eingegangen.

Obwohl die semantische Wiederverwendbarkeit durch Trennung von Layout und Struktur eines Fragmentes einen sehr mächtigen Mechanismus darstellt, soll je-

doch ein Problem nicht verschwiegen werden. Ein ästhetisch ansprechendes, automatisches Layout von multimedialen Dokumenten ist nur dann zu erwarten, wenn möglichst viel Wissen über die Struktur dieser Dokumente vorliegt und in die Generierung des Layouts eingebracht werden kann. Unterscheiden sich jedoch die Struktur erheblich, ist es schwierig, eine automatische Layout-Generierung zu erreichen, die den Betrachter eines multimedialen Dokuments in wirklich allen Fällen zufrieden stellt. Außerdem ist das Layout ein wichtiges rhetorisches Mittel eines Autors multimedialer Dokumente. Entzieht man das Layout seiner Kontrolle, so beraubt man ihn eines Teils seiner Ausdruckskraft.

### 3.1.3 Selektion wiederverwendbarer Komponenten

Bei einer großen Sammlung wiederverwendbarer Komponenten, seien es vollständige Dokumente, Fragmente oder Medienobjekte, ist es wichtig, daß Vorkehrungen getroffen werden, die eine schnelle Suche und Auswahl einzelner Komponenten durch einen Autor eines multimedialen Dokumentes ermöglichen [Kru92]. Ist die Suche und Auswahl einer Komponente, die *Selektion*, mühselig und langwierig, werden viele Autoren die Möglichkeiten der Wiederverwendbarkeit nicht nutzen. Dieses führt zu unnötiger Redundanz und mindert die Effizienz der Dokumenterstellung erheblich.

Problematisch bei der Selektion wiederverwendbarer Komponenten ist, daß Autoren oft gar nicht so genau wissen, wonach sie suchen und die Ziele ihrer Suche auch nur unpräzise beschreiben können [Hen94]. Sie müssen also mehrere in Frage kommende Komponenten auswählen und deren Inhalte mit ihren Wünschen vergleichen. Um auch bei einer großen Zahl von Komponenten die Selektion für die Autoren möglichst einfach und effizient zu gestalten, sollte sie softwaregestützt durchführbar sein. Damit ergibt sich als Anforderung an ein flexibles multimediales Dokumentmodell, Unterstützung für eine softwaregestütze Suche nach wiederverwendbaren Dokumentbestandteilen bereitzustellen. Eine solche Software muß in der Lage sein, Informationen über den Inhalt einer wiederverwendbaren Komponente zu erhalten.

Die einfachste Möglichkeit, dieses zu erreichen, besteht darin, wiederverwendbaren Komponenten aussagekräftige Namen geben zu können. Es ist jedoch fraglich, ob sich jede Komponente mit einem Namen ausreichend beschreiben läßt und ob dieser Name genau den Aspekt einer Komponente verdeutlicht, nach dem Autoren suchen. Desweiteren ergibt sich das sogenannte *Wortschatzproblem* [Hen94]. Verschiedene Autoren mit unterschiedlichen Hintergründen assoziieren mit einem Konzept unterschiedliche Begriffe. Sucht ein Autor mit dem Begriff nach einer Komponente, den er mit dieser verbindet, so kann es passieren, daß ein anderer Autor zwar eine passende Komponente bereitgestellt hat, jedoch unter einem anderen Namen. Die Suche schlägt damit fehl.

Man kann das Wortschatzproblem mit einem *Thesaurus* abmildern. Ein Thesaurus gibt zu einem Begriff eine Menge von synonym gebrauchten Begriffen an. Eine Suche bezieht sich damit nicht nur auf einen Begriff, sondern auch auf die Menge seiner Synonyme, was die Trefferwahrscheinlichkeit bei der Suche nach einer Komponente erhöht, jedoch auch mehr Ergebnisse zurückliefert, die ein Autor miteinander vergleichen muß.

Eine andere Möglichkeit der Unterstützung der softwarebasierten Selektion von wiederverwendbaren Komponenten besteht in der Gruppierung ähnlicher Komponenten zu *Klassen*. Die Bezeichnung einer Klasse beschreibt einen Aspekt, den alle

Komponten dieser Klasse gemeinsam haben. Da die Klassifizierung der Komponenten vorgegeben ist (und damit die Bezeichnungen der Klassen), ist auch das Vokabular für die Suche nach Komponenten festgelegt, wodurch das Wortschatzproblem entfällt. Da eine Komponente mehreren Klassen zugeordnet werden kann, ist diese mit mehreren, auch nicht-synonymen Begriffen beschreibbar. Prieto-Diaz [Pri89] unterscheidet zwischen der *hierarchischen* und der *facettenorientierten* Klassifikation.

Bei der hierarchischen Klassifikation werden die einzelnen Klassen in einer baumartigen Inklusionshierarchie angeordnet. Jeder Knoten repräsentiert dabei eine Klasse, jede Kante eine Inklusionsbeziehung. Die Komponenten, die einer durch einen Sohnknoten repräsentierten Klasse zugeordnet sind, bilden eine Teilmenge der Komponenten der durch seinen Vaterknoten repräsentierten Klasse. Die Traversierung von einem Knoten zu dessen Sohn reduziert damit die Zahl der betrachteten Komponenten. Bild 3.2 zeigt ein Klassifikationsbeispiel für eine Sammlung multimedialer Komponenten aus dem Bereich der Biologie.

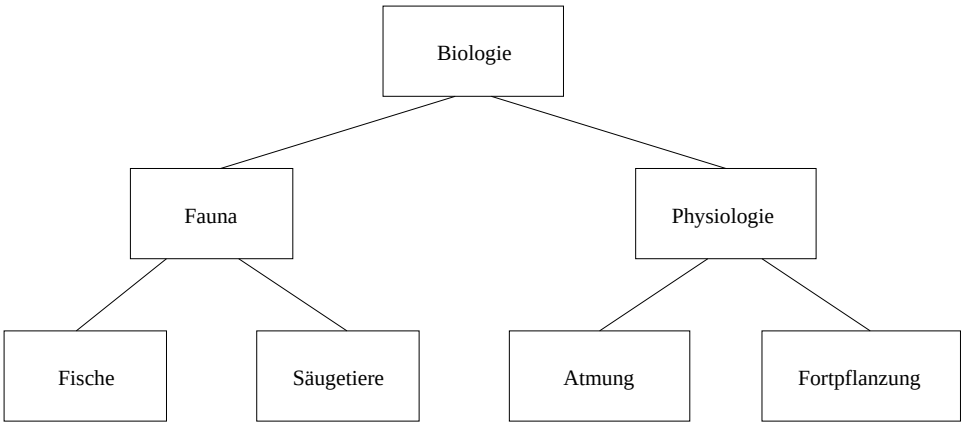


Abbildung 3.2: Beispiel für hierarchische Klassifikation

An diesem Beispiel zeigen sich die wesentlichen Probleme der hierarchischen Klassifikation. Viele Zusammenhänge sind nicht vernünftig hierarchisch beschreibbar. Man stelle sich zum Beispiel vor, man wolle ein multimediales Dokument über die Atmung von Fischen in die Hierarchie einordnen. Sollte dieses Dokument im Physiologie-Teilbaum unter Atmung oder im Fauna-Teilbaum unter Fischen eingeordnet werden? Wenn mehrere Dokumente zu diesem Thema existieren stellt sich die Frage, ob eine Einrichtung der Klasse „Atmung von Fischen" als Subklasse von Fisch bzw. von Atmung sinvoll ist. Welcher der Teilbäume für diese Subklasse gewählt wird, entscheidet über den Erfolg einer Suche, abhängig davon, ob der Suchende den über den Teilbaum der Fauna oder den der Physiologie traversiert. Eine hierarchische Klassifikation kann demnach schwer erweiterbar sein.

Die facettenorientierte Klassifikation umgeht das Problem der hierarchischen Beschreibbarkeit, indem eine Menge von *Facetten* definiert wird. Eine Facette ist eine Eigenschaft mit einem vorgebenen Wertebereich. Zur Klassifizierung einer wiederverwendbaren Komponente wird eine möglichst große Teilmenge von Facetten ausgewählt und mit denjenigen Werten belegt, die diese Komponente am treffendsten beschreiben.

Für die oben bereits erwähnte Sammlung von multimedialen Komponenten aus dem Bereich der Biologie ist die Definition der Facetten „Physiologie" und „Fauna" mit den Wertebereichen $D_{Physiologie} = \{Atmung, Fortpflanzung\}$ und $D_{Fauna} =$

$\{Fisch, Säugetier\}$ sinnvoll. Soll ein allgemeines Dokument über Fortpflanzung klassifiziert werden, so geschieht dies über die Wertbelegung der Facette „Physiologie": $Physiologie = Fortpflanzung$. Ein Dokument über die Fortpflanzung von Fischen wird hingegen mit den folgenden Wertbelegungen der Facetten „Physiologie" und „Fauna" klassifiziert: $Physiologie = Fortpflanzung, Fauna = Fisch$.

Die Vorteile der facettenorientierten Klassifikation liegen zum einen darin, daß sich sehr einfach die für eine Komponente „perfekte" Klassifikation bestimmen läßt. Zum anderen ist eine Klassifikation durch Hinzufügen von weiteren Facetten leicht erweiterbar, weil die verschiedenen Facetten in keiner Beziehung zueinander stehen [Pri89].

## 3.2 Interaktion

Während die Definition von Multimedia von Steinmetz [SN95] auf Seite 6 keine Interaktionsmöglichkeiten voraussetzt, so entfaltet die Präsentation multimedialer Dokumente erst ihren vollen Reiz durch die aktive Beeinflussung des Präsentationsverlaufs durch den Betrachter. Der Betrachter ist bei einem reizvollen multimedialen Dokument nicht an eine vom Autor vorgegebene Lesefolge gebunden, sondern kann bei der Präsentation des Dokumentes nach seinem Bedarf direkt Punkte anspringen, die ihn interessieren, nicht verstandene Abschnitte wiederholen, die Lautstärke von Musikstücken an seine Hörfähigkeit oder seine Stimmung anpassen und vieles mehr. Nicht-interaktive Dokumente sind dagegen weniger ansprechend. Deren Präsentation könnte auf einer Videokassette aufgenommen und über einen Fernseher abgespielt werden. Solche Präsentationen sind somit äquivalent zu einem Film. Ein Film ist aber nicht das, was man unter Multimedia verstehen möchte. Aus diesem Grund gibt es Definitionen von Multimedia, die Interaktion explizit verlangen [DGHL91].

*Interaktion* wird im folgenden als die Beeinflussung des Präsentationsverlaufs eines Dokumentes durch den Betrachter verstanden. Im nachfolgenden Abschnitt sollen nun in Anlehnung an [Bol94] die grundlegenden Begriffe dieser Mensch-Maschinen-Kommunikation herausgearbeitet werden.

Interaktion beginnt mit einem *Interaktionsziel*. Der Benutzer möchte etwas erreichen, sich beispielsweise über die gravierende Bedeutung von Bypass-Operationen in der modernen Medizin informieren. Dazu muß der Benutzer dieses Ziel in eine Menge einfacherer *Interaktionsaufgaben* zerlegen. Der Begriff Interaktionsaufgabe bezeichnet den Typ von Informationen, den ein Benutzer in den Computer eingeben kann. Es gibt fünf Basisinteraktionsaufgaben: Positionierung, Texteingabe, Auswahl, Quantifizierung und Bestätigung. Aus diesen lassen sich komplexere Interaktionsaufgaben wie Dialogboxen, Konstruktion und Manipulation herleiten.

Die für die Präsentation eines multimedialen Dokumentes zuständige Software muß die oben genannten Interaktionsaufgaben unterstützen. Dieses wird durch die sogenannten *Interaktionsformen* erreicht. Im Bereich der Präsentation multimedialer Dokumente dominieren im wesentlichen die User-Interface-Komponenten. UI-Komponenten haben weite Verbreitung durch die graphischen Benutzerschnittstellen wie zum Beispiel Windows und X-Window gefunden. Typische UI-Elemente sind Buttons, Eingabefelder, Schieberegler und Menüs.

Eine Interaktionsform stellt dem Benutzer eine Eingabemöglichkeit zur Erfüllung einer Interaktionsaufgabe zur Verfügung und muß mittels einer *Interaktionstechnik*

bedient werden. Zum Beispiel erfolgt die Auswahl eines Menüpunktes über einen Mausklick, während die Texteingabe in einem Eingabefeld über die Tastatur erfolgt. Zur Durchführung von Interaktionstechniken benötigt man *Eingabegeräte*. Typische Eingabegeräte sind Tastatur und Maus. In letzter Zeit ist aber auch die Technik der Spracherkennung merklich gereift, so daß der Spracheingabe wohl eine ständig wachsende Bedeutung zukommen wird. Abbildung 3.3 zeigt nochmal die Zusammenhänge zwischen den oben erläuterten Begriffen.

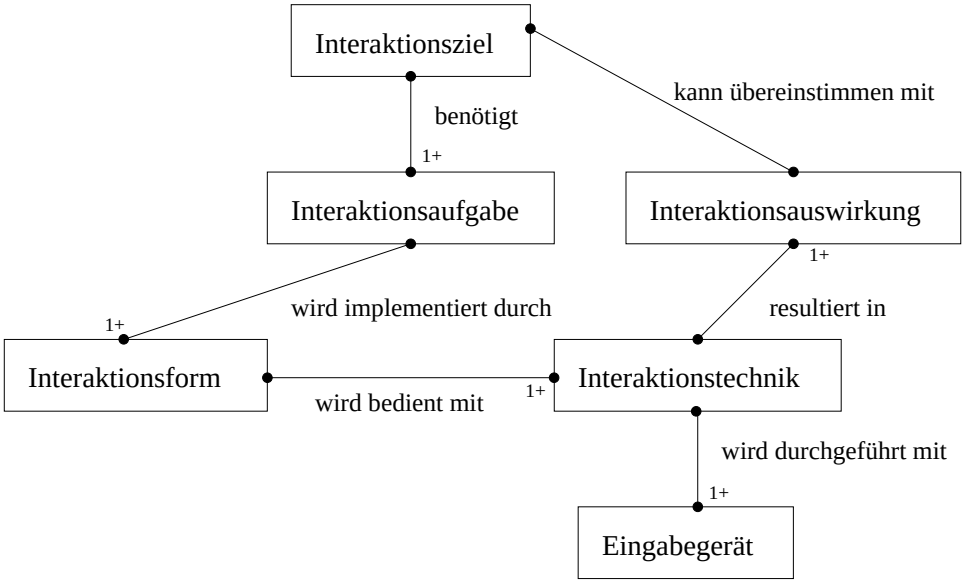


Abbildung 3.3: Übersicht über die Terminologie von Interaktion in OMT-Notation

Die Bedienung der Interaktionsformen durch den Benutzer resultiert in einem gewissen Verhalten der Präsentation eines multimedialen Dokuments, welches im Idealfall mit seinem Interaktionsziel übereinstimmt, es aber nicht muß. Dieses Verhalten wird *Interaktionsauswirkung* genannt. Die für die Präsentation multimedialer Dokumente relevanten Interaktionsauswirkungen lassen sich in folgende Gruppen einteilen [Bol94]:

- *Navigationsinteraktionen*: Solche Interaktionen haben Einfluß auf den zeitlichen Ablauf einer Präsentation. Ein Beispiel für eine Navigationsinteraktion ist die Aktivierung eines Links in einem Hypertext-System. Dieses führt zur Präsentation des Dokumentes, auf welches das Link-Ende verweist. Ein anderes Beispiel ist ein Auswahlmenü. Je nach Wahl wird ein anderes Video abgespielt.

- *Gestaltungsinteraktionen*: Interaktionen dieser Art haben Einfluß auf die Gestaltung einer multimedialen Präsentation. Typische Gestaltungsinteraktionen sind ein Schieberegler zur Manipulation der Lautstärke eines Audios oder ein Menü zur Zeichensatzauswahl.

Es wird gefordert, daß ein flexibles multimediales Dokumentmodell Konzepte zur Beschreibung dieser Interaktionsarten bereitstellt.

## 3.3 Adaption

Da das Informationssystem zur Herzchirurgie kontextabhängig sein soll, sollte sich die Präsentation der multimedialen Dokumente an das Bildungsniveau, an die Vorlieben, an die Interessen und an die zur Verfügung stehende technische Ausrüstung eines Betrachters anpassen. Wie oben schon angedeutet wird ein Professor ein Dokument über eine Herzoperation auf einem anderen Niveau betrachten wollen als ein Student im Grundstudium. Außerdem ist es nicht unwahrscheinlich, daß der Professor in seinem Büro über einen schnelleren Netzzugang verfügt als der Student in seinem Zimmer im Studentenwohnheim.

Zur Beschreibung eines Benutzers dient das sogenannte *Benutzerprofil* [Wei97]. Das Benutzerprofil ist die Zusammenfassung aller für eine Anwendung wesentlichen Eigenschaften eines Benutzers, in der Regel in der Form von Attributen. Die Anpassung der Präsentation eines multimedialen Dokumentes an dieses Benutzerprofil nennt man *Adaption*. Man kann zwei grundlegendene Arten von Adaption unterscheiden [Wei97].

Bei der *clientbasierten Adaption* können in einem multimedialen Dokument mehrere Varianten seines Inhalts vom Autor spezifiziert werden. Das multimediale Dokument wird zusammen mit sämtlichen Varianten in die Präsentationssoftware, hier Client genannt, geladen, die dann aufgrund des Benutzerprofils sich für die Präsentation einer Variante entscheidet. Dieser Adaptionsansatz macht dann Sinn, wenn ein Autor sämtliche möglichen Varianten eines multimedialen Dokumentes überblicken kann und diese fest vorgeben möchte.

Bei der *serverbasierten Adaption* erfolgt die Auswertung des Benutzerprofils bei der Instanz, im folgenden Server genannt, welche die multimedialen Dokumente verwaltet. Im Falle des Projekts „Galerie der Herzchirurgie" ist dies die Datenbank, in der die multimedialen Dokumente und deren Bestandteile abgelegt sind. Jedes multimediale Dokument, das auf dem Server abgelegt ist, wird gemäß der Eigenschaften der sich für das Dokument interessierenden Benutzer klassifiziert (zum Beispiel mit einer der im vorigen Abschitt vorgestellten Methoden). Soll ein auf dem Server gehaltenes Dokument von der Präsentationssoftware präsentiert werden, so wird das Benutzerprofil dem Server übergeben. Dieser wählt anhand vom Benutzerprofil das am besten zum Betrachter passende Dokument aus. Dieses Dokument wird daraufhin von der Präsentationssoftware präsentiert. Dieser Adaptionsansatz berücksichtigt sämtliche, beim Server gehaltenen, multimedialen Dokumente. Ein Autor muß demnach nicht alle Varianten des Inhalts eines Dokuments überschauen und fest vorgeben können. Vielmehr ist für jede Variante des Inhalts eines Dokumentes ein neues multimediales Dokument zu erstellen, das lediglich korrekt klassifiziert werden muß, um bei zukünftigen Adaptionen berücksichtigt zu werden.

## 3.4 Präsentationsneutralität

Das zu entwickelnde Informationssystem „Galerie der Herzchirurgie" soll netzwerkbasiert sein. Man muß dabei eine heterogene Netzwerklandschaft annehmen, in der sich Rechner mit unterschiedlichen Betriebssystemen und Softwareinstallationen befinden. Um Portierungsarbeiten zu vermeiden, ist es wünschenswert, schon auf diesen Rechnern vorhandene Präsentationssoftware für andere Dokumentmodelle wiederzuverwenden. Dieses bringt die Forderung nach *Präsentationsneutralität* mit sich. Präsentationsneutralität meint die Unabhängigkeit der Verwaltung eines multime-

dialen Dokumentes von dessen Präsentation. Es soll möglich sein, vor der Präsentation eines Dokumentes, dieses in ein anderes Dokumentmodell zu konvertieren. Dabei ergeben sich zwei wesentliche Probleme, die eine vollständige, automatische Konvertierung behindern können.

Einerseits kann es passieren, daß das Zieldokumentmodell weniger multimediale Funktionalität als das Dokumentmodell des Quelldokumentes hat. Eine verlustfreie Konvertierung ist dann in der Regel nicht möglich. Zum Beispiel erlaubt SMIL als multimediales Dokumentmodell die Beschreibung der zeitlich synchronisierten Darstellung von Medienobjekten, während in HTML [Tol96], das kein multimediales Dokumentmodell darstellt, zeitliche Synchronisation nicht beschreibbar ist.

Andererseits kann es vorkommen, daß das Zieldokumentmodell multimediale Dokumente auf einer höheren *semantischen Ebene* als das Dokumentmodell des Quelldokumentes beschreibt. Eine Beschreibung auf semantisch *hoher* Ebene meint, daß weniger die Präsentation eines Dokumentes als vielmehr dessen Inhalt und Struktur beschrieben wird. Das bedeutet, daß ein Konverter die Spezifikation des Quelldokumentes analysieren und deren Semantik in Form der sematisch höher stehenden Konstrukte des Zieldokumentmodells herleiten muß. Diese Analyse erfordert i.a. das Wissen des Autors des über ein Dokument, weswegen eine vollständige Automatisierung dieses Prozesses kaum möglich ist [RvOB97].

Im Gegensatz dazu ist die automatische Konvertierung eines Dokumentes, welches auf einer semantisch höheren Ebene beschrieben ist, in ein Dokumentmodell, das auf semantisch niedrigeren Konstrukten beruht, relativ einfach zu bewerkstelligen.

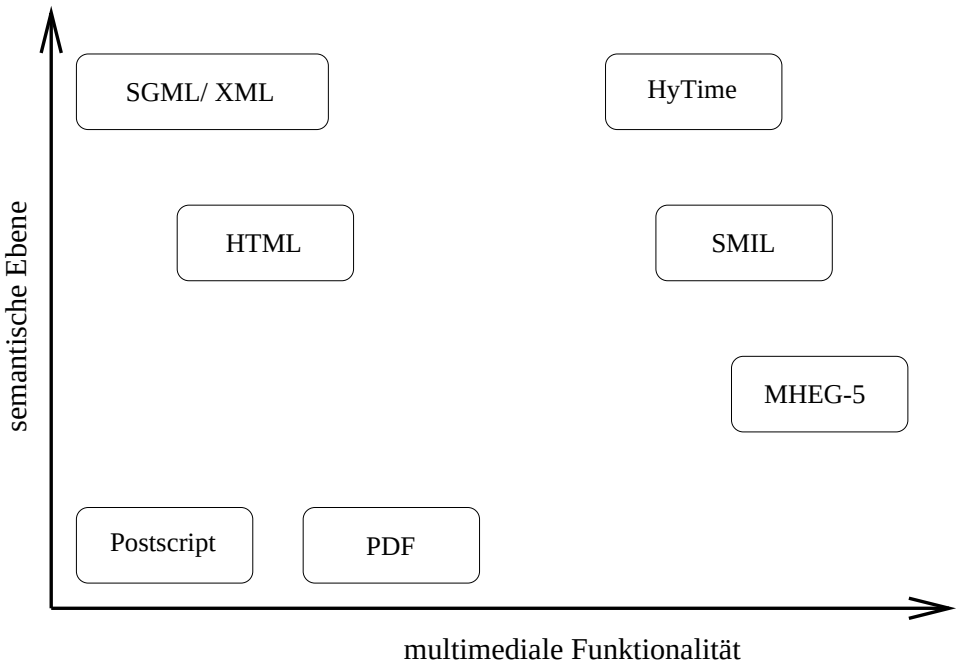


Abbildung 3.4: Präsentationsneutralität und multimediale Funktionalität von Dokumentmodellen

Die Problematik der automatischen Konvertierung zwischen Dokumentmodellen soll anhand von Abbildung 3.4 veranschaulicht werden. Angelehnt an Rutledge [RvOB97] werden in dieser Abbildung existierende, nicht unbedingt multimediale, Dokumentmodelle anhand ihrer semantischen Ebene und der von ihnen unterstütz-

ten multimedialen Funktionalität angeordnet. Aus der Abbildung ist beispielsweise ersichtlich, daß die Konvertierung von MHEG-5 Dokumenten nach Postscript ein Problem darstellt, da Postscript nur wenig multimediale Funktionalität hat. Anderseits stellt aber auch die Konvertierung von MHEG-5 nach HyTime ein Problem dar, weil HyTime eher die Struktur eines multimedialen Dokuments beschreibt und nicht wie MHEG-5 dessen Präsentationsablauf, also auf einer höheren semantischen Ebene arbeitet. Im Gegensatz dazu ist die automatische Konvertierung von HTML nach MHEG-5 kein Problem.

Möchte man demnach die Probleme der automatischen Konvertierung umgehen und dennoch die präsentationsneutrale Ablage von Dokumenten erlauben, so ergeben sich als Forderungen an ein flexibles multimediales Dokumentmodell einerseits die Beschreibung von multimedialen Dokumenten auf einer möglichst hohen semantischen Ebene und andererseits die Bereitstellung von viel multimedialer Funktionalität. Da Multimedia ein sich schnell entwickelndes Gebiet ist, dürfte es schwer fallen, alle zukünftigen Entwicklungen vorherzusehen und eine wirklich vollständigen Menge an multimedialer Funktionalität zur Verfügung zu stellen. Deswegen sollte ein Dokumentmodell leicht um neue Funktionalität erweiterbar sein.

# Kapitel 4

# Multimediale Dokumentstandards

Im vorigen Kapitel wurden die verschiedenen Anforderungen, die ein flexibles multimediales Dokumentmodell erfüllen soll, vorgestellt und näher betrachtet. Im diesem Kapitel soll eine Einführung in die verbreitetsten multimediale Dokumentstandards erfolgen und untersucht werden, inwieweit diese Standards den gestellten Forderungen gerecht werden.

## 4.1 MHEG-5

Die MHEG-Standards bieten ein objektorientiertes Modell zur Beschreibung von interaktiven, multimedialen Dokumenten. Das Hauptziel ist hierbei, die Austauschbarkeit der multimedialen Dokumente zwischen unterschiedlichen Präsentationsumgebungen zu gewährleisten. Es soll beispielsweise keinen Unterschied machen, ob ein Dokument auf einem Apple Macintosh oder auf einem Windows-Rechner präsentiert wird. Die Präsentationssoftware für die multimedialen Dokumente der MHEG-Standards nennt man *MHEG-Engine*. Die Idee ist, daß ein multimediales Dokument auf den unterschiedlichsten Plattformen präsentiert werden kann, solange diese über eine MHEG-Engine verfügen.

Der MHEG-1-Standard [MBE95] bildet das Fundament der MHEG-Standards. Er beschreibt die Kernklassen, welche Beschreibungsmittel für grundlegende Medienobjekte (z.B. Video, Audio), für die Interaktion, für die zeitliche Synchronisation aber auch für die Verknüpfung von Medienobjekten zu komplexeren Einheiten zur Verfügung stellen. Eine Kombination von Instanzen dieser Klassen bildet eine sogenannte *MHEG-Applikation*, i.e. ein multimediales Dokument, welche auf jeder MHEG-Engine präsentiert werden kann. Problematisch ist jedoch, daß der MHEG-1-Standard sehr abstrakt und zudem noch sehr komplex ist. Für eine konkrete Anwendung muß er deswegen in der Regel an den speziellen Anwendungsbereich angepaßt werden [JR95].

Eine solche Anpassung stellt der MHEG-5-Standard [ISO95, JR95] dar. Die Zielgruppen von MHEG-5 sind hauptsächlich die sogennannten Kiosk- und Video-On-Demand-Anwendungen, die auf low-end PCs und Set-Top-Boxen ablaufen sollen. Mit Blick auf diesen Anwendungbereich ergaben sich für die Anpassung von MHEG-

1 an MHEG-5 einige Anforderungen. MHEG-5-Applikationen sollen in einer ,,Final Form" vorliegen. Das bedeutet, daß die Repräsentation einer MHEG-5-Applikation nicht mehr vom Menschen lesbar sein muß. In der Tat definiert der Standard zwei Beschreibungsformate: ein binäres beschrieben in der ASN.1-Notation, sowie ein textuelles Beschreibungsformat. Eine weitere Forderung ist die Plattformunabhängigkeit. Die Ausführung einer Applikation muß auf allen MHEG-5-Engines und allen Rechnern gleich aussehen und sich identisch verhalten. Dazu bedarf es einer exakten Beschreibung der Semantiken der MHEG-5-Klassen . Weiterhin muß der Standard Systeme mit geringer Leistungsfähigkeit und wenigen Ressourcen unterstützen, weswegen der Aufwand der Präsentation einer Applikation relativ gering sein muß.

### 4.1.1 Beschreibung

Eine MHEG-5-Applikation besteht in Analogie zu Filmen oder Theaterstücken aus einer Aneinandereihung einzelner *Szenen*. Eine Szene ist ein Behälter für die in ihr auftretenden Medienobjekte und der Beschreibung der Beziehungen zwischen diesen. Eine Szene nimmt bei ihrer Präsentation die gesamte Präsentationsfläche in Anspruch. Es kann somit immer nur eine Szene einer Applikation gleichzeitig präsentiert werden. Diese Szene wird die *aktive* Szene genannt.

Die Beschreibung einer MHEG-5-Applikation ist objektorientiert aufgebaut. Dazu stellt der Standard eine Menge von *MHEG-5-Klassen* zur Verfügung. Zur Beschreibung einer MHEG-5-Applikation können Instanzen der MHEG-5-Klassen erzeugt und zueinander in Beziehung gesetzt werden. Jede MHEG-5-Klasse definiert eine Menge von *Attributen*. Desweiteren ist jeder MHEG-5-Klasse eine Menge von *elementaren Aktionen* zugeordnet, die das Verhalten einer Instanz der Klasse definieren. Die Auswirkungen der elementaren Aktionen sind fest vorgegeben. Elementare Aktionen und Attribute entsprechen Methoden und Attributen in objektorientierten Programmiersprachen.

Jede MHEG-5-Klasse verfügt zudem über eine Menge von *Events*, die eine Instanz dieser Klasse *signalisiert*, wenn bei der Präsentation der Applikation gewisse Ereignisse auftreten. Ein Ereignis kann zum Beispiel ein Mausklick aber auch das Verstreichen einer gewissen Zeitspanne sein. An ein Event können elementare Aktionen gekoppelt werden, die ausgeführt werden, sobald das Event signalisiert wird. Einem Event kann bei der Signalisierung ein Wert mitübergeben werden. Es macht zum Beispiel Sinn, bei einem Mausklick anzugeben, welche Maustaste gedrückt wurde.

Die MHEG-5-Klassen sind in einer Vererbungshierarchie angeordnet. Eine Klasse erbt dabei von einer Elternklasse deren Attribute, assoziierte Events und elementare Aktionen. Zusätzlich kann eine Klasse weitere Attribute, Events und Aktionen definieren bzw. die Semantik der geerbeten Attribute, Events und Aktionen ändern. Die wesentlichen Teile der Klassenhierarchie von MHEG-5 sollen im folgenden vorgestellt und die wichtigsten Klassen kurz beschrieben werden.

#### Zentrale Klassen von MHEG-5

In diesem Abschnitt werden diejenigen Klassen von MHEG-5 vorgestellt, die den Kern der Beschreibung einer Applikation ausmachen. Die Hierarchie dieser zentralen Klassen zeigt Abbildung 4.1.

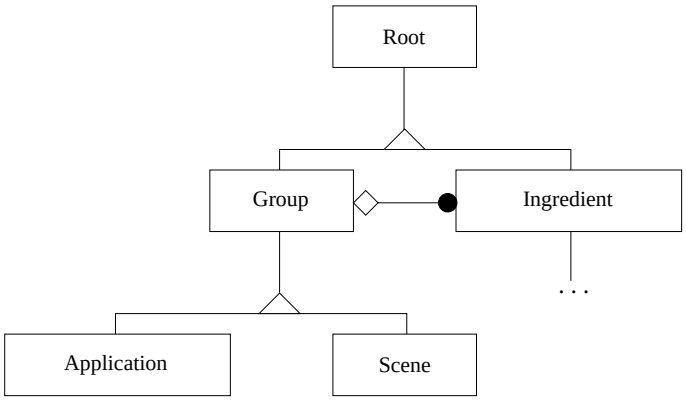


Abbildung 4.1: Zentrale Klassen von MHEG-5 in OMT-Notation

- **Root**

  Die Klasse `Root` bildet die Basisklasse für alle anderen MHEG-5-Klassen. Ihre Hauptaufgabe besteht darin, generisches Verhalten für die Vorbereitung, Aktivierung, Deaktivierung und Zerstörung einer Instanz einer Klasse zu definieren sowie Mittel zur deren Identifikation bereitzustellen. Dazu definiert die Klasse `Root` ein spezielles Attribut `ObjectIdentifier`, das eine Instanz einer MHEG-5-Klasse eindeutig identifiziert. Eine solche Identifikation besteht aus zwei Teilen: dem *Gruppenidentifikator* und der *Objektnummer*. Der Gruppenidentifikator kennzeichnet eindeutig diejenige Gruppe von Instanzen von MHEG-5-Klassen, zu der eine Instanz gehört, beispielsweise ein Szene. Die in dieser Gruppe enthaltenen Instanzen von MHEG-5-Klassen werden durch eine in der Gruppe eindeutige Objektnummer identifiziert. Eine Gruppe wird durch eine Instanz der Klasse `Group` (s.u.) realisiert, welche logisch selbst zur Gruppe gehört und innerhalb dieser mit $Objektnummer = 0$ angesprochen wird.

- **Group**

  Die Elemente einer Applikation, die sogenannten *Ingredients*, werden in Gruppen angeordnet. Diese Gruppen strukturieren damit die Beschreibung einer MHEG-5-Applikation. Gruppen können nicht ineinander verschachtelt werden, so daß die Strukturierungsmöglichkeiten einer Applikation begrenzt sind. Die Klasse `Group` ist abstrakt und kann nicht direkt instanziiert werden.

- **Scene**

  Die Klasse `Scene` stellt eine Spezialisierung der abstrakten Klasse `Group` dar. Sie realisiert die grundlegende Strukturierungsmetapher einer MHEG-5-Applikation, die Szene. Bei der Präsentation einer Applikation kann immer nur eine Instanz der Klasse `Scene` zu einem Zeitpunkt aktiv sein. Die Ingredients einer Instanz von `Scene` können nur dann präsentiert werden, wenn diese Instanz aktiv ist. Um den Übergang von einer Szene zur nächsten zur ermöglichen, bietet die Klasse `Scene` die elementare Aktion `TransitionTo` an.

- **Application**

  Jede MHEG-Applikation enthält exakt eine Instanz der Klasse `Application`, die eine Subklasse von `Group` bildet. Eine Instanz von `Application` hat im wesentlichen zwei Aufgaben. Zum einen legt sie diejenige Instanz der Klasse `Scene` fest, die bei Beginn der Präsentation der Applikation präsentiert wird.

Sie bildet also den Einstiegspunkt der Applikation. Zum anderen gruppiert sie Ingredients, die szenenübergreifend präsentiert werden sollen. Das macht zum Beispiel Sinn bei einer Hintergrundmusik. Es ist möglich, aus einer Szene die Ingredients einer Instanz von Application zu referenzieren. Die Klasse `Application` bietet die elementaren Aktionen `Spawn` und `Launch` an, die es erlauben, aus einer MHEG-5-Applikation heraus eine andere zu starten. `Launch` beendet dabei die Präsentation der aktuellen Applikation, während `Spawn` diese lediglich nur unterbricht.

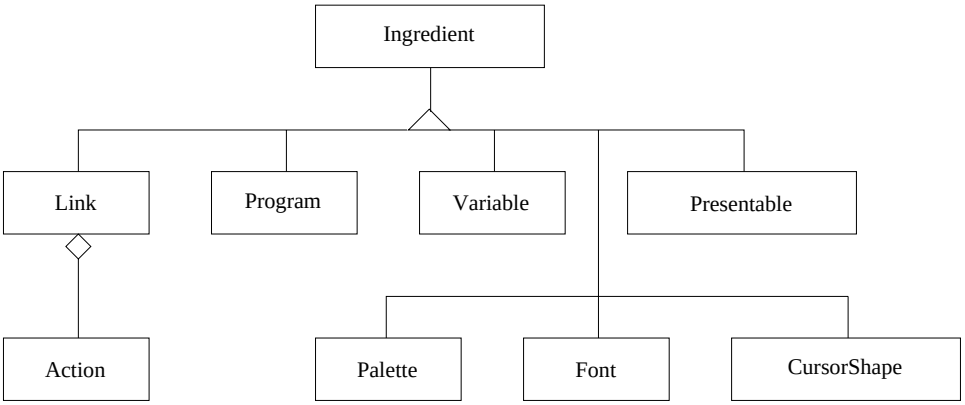


Abbildung 4.2: Ingredient-Teilbaum der MHEG-5 Klassenhierarchie in OMT-Notation

### Ingredient-Klassen von MHEG-5

In diesem Abschnitt werden die Subklassen von `Ingredient` vorgestellt. Abbildung 4.2 zeigt den zugehörigen Teilbaum der Klassenhierachie.

- **Variable**

  In Instanzen der Klasse `Variable` können Werte gespeichert und wieder ausgelesen werden. Für die unterschiedlichen Typen der zu speichernden Werte existieren entsprechende Subklassen von `Variable`. Es gibt Subklassen für String-, Boolean- und Integervariablen. Außerdem existiert die Subklasse `ObjectRef`, die Referenzen auf Instanzen von MHEG-5-Klassen aufnehmen kann. Dieses ermöglicht die indirekte Adressierung dieser Instanzen. Variablen können z.B. dazu dienen, Benutzereingaben zu speichern. Wird eine Instanz von `Variable` in einer Instanz von `Applikation` gruppiert, hat diese die Funktion einer globalen Variable in imperativen Programmiersprachen und kann beispielsweise dazu dienen nachfolgende Szenen und externe Programmaufrufe zu parametrisieren.

- **Program**

  Eine Instanz der Klasse `Program` kapselt ein externes Skript, ein Programm, oder einen Remote Procedure Call. Es besteht die Möglichkeit bei der Präsentation der Applikation, ein solches Programm synchron oder asynchron zum Präsentationsverlauf zu aktivieren. Außerdem können diesen Programmen Werte, Variableninhalte und Referenzen auf Instanzen von MHEG-5-Klassen übergeben werden. Hier setzt der MHEG-6-Standard [Hof96] an. Dieser

Standard definiert eine Schnittstelle zwischen einer MHEG-5-Engine und einer in ihr integrierten Java Virtual Machine. Diese Schnittstelle erlaubt es einerseits, über die Klasse `Program` Java-Applikationen zu aktivieren und diesen Parameter zu übergeben. Andererseits ist es aber auch möglich, aus der Java-Applikationen heraus Instanzen von MHEG-5-Klassen anzusprechen und zu manipulieren.

- **Link**

  Die Instanzen der Klasse `Link` beschreiben das Verhalten der MHEG-5-Applikation. Ein Link besteht aus einer *Bedingung* und einem *Aktionsobjekt*. Wenn bei der Präsentation der Applikation die Bedingung erfüllt ist, „feuert" der Link, d.h. es wird das Aktionsobjekt ausgeführt. Eine Bedingung besteht aus drei Teilen:

  1. das Event, auf das reagiert werden soll,
  2. die Instanz einer MHEG-5-Klasse, bei der das Event auftreten soll und
  3. der Wert, der dem Event mitübergeben soll, falls dieses über einen verfügt.

  Das Aktionsobjekt ist eine Instanz der Klasse `Action`. Es hat die Aufgabe, beim Feuern eines Links eine Sequenz von elementaren Aktionen synchron nacheinander aufzurufen. Sind während der Präsentation einer Applikation bei mehreren Links zum gleichen Zeitpunkt die Bedingungen erfüllt, so werden die Links nacheinander in einer nicht festgelegten Reihenfolge gefeuert.

  Abbildung 4.3 zeigt eine einfache Definition einer Szene (inklusive Links) in der textuellen Repräsentation[1] von MHEG-5 [Hof96]. Sie enthält zwei Buttons `button1` und `button2`. Wird `button1` gedrückt, so wird mit der Präsentation der Szene `scene2` fortgefahren. Wird hingegen `button2` gedrückt, so wird die Szene `scene3` präsentiert.

```
1     (:scene scene1
2       (:group-items
3         (:push-button button1
4           (:label "Press me")
5           (:original-position 10 50)
6           (:original-box-size 70 10))
7         (:push-button button2
8           (:label "Press me, too")
9           (:original-position 10 80)
10          (:original-box-size 70 10))
11        (:link button1-pressed
12          (:event-source button1)
13          (:event-type is-selected)
14          (:link-effect
15            (:action
16              (:transition-to scene2 () 12))
17        ))
18        (:link button2-pressed
19          (:event-source button2)
20          (:event-type is-selected)
21          (:link-effect
22            (:action
23              (:transition-to scene3 () 12))
24        ))
25    ))
```

Abbildung 4.3: Beispiel einer Szene in MHEG-5

[1] In der Draft-International- Standard-Notation (DISN). Mittlerweile gibt es eine International Standard Notation, die sich von der DISN unterscheidet.

- **Presentable**

  Die abstrakte Klasse `Presentable` umfaßt die präsentierbaren Medienobjekte, wie z. B. Videos, Audios und Bitmaps, aber auch Interaktionsformen wie Buttons und Sliders. Die zu einem Medienobjekt gehörenden Daten können einerseits direkt in das Medienobjekt integriert sein, andererseits aber auch extern vorliegen und vom Medienobjekt lediglich referenziert werden. Für genauere Beschreibungen der Subklassen von `Presentable` sei auf Joseph [JR95] verwiesen.

- **Palette, Font, CursorShape**

  Die Klasse `Palette` kapselt Color-Lookup-Tables, die dazu dienen, Indizes in echte Farbwerte zu übersetzen. Eine Instanz dieser Klasse kann jedem Ingredient mit visueller Komponente zugeordnet werden, um die Farben festzulegen, in denen dieses Indegrient dargestellt wird.

  Gleichsam dient die Klasse `Font` dazu, Zeichensätze zu kapseln. Eine Instanz dieser Klasse kann jedem Text zugeordnet werden.

  Eine Instanz der Klasse `CursorShape` legt die Bitmaps fest, mit denen der Cursor des Eingabegerätes (z. B. eine Maus) dargestellt wird. Jeder Szene kann ein solcher Cursor zugeordnet werden.

### 4.1.2 Bewertung

In diesem Abschnitt soll untersucht werden inwieweit MHEG-5 die in Kapitel 3 gestellten Anforderungen an ein flexibles multimediales Dokumentmodell unterstützt. Eine Zusammenfassung der Bewertung zeigt Tabelle 4.1 auf Seite 26.

#### Wiederverwendbarkeit

Zunächst einmal ist zu überprüfen, ob und auf welcher Ebene MHEG-5 Wiederverwendbarkeit ermöglicht. Einzelne Medienobjekte einer MHEG-Applikation werden durch Instanzen der Klasse `Presentable` repräsentiert. Sie sind damit auch Instanzen der Klasse `Ingredient` und deswegen einer MHEG-Gruppe zugeordnet und auch nur über diese Gruppe adressierbar. Es ist nicht möglich, daß ein und dasselbe Medienobjekt in mehreren Gruppen enthalten ist. Somit sind einzelne Medienobjekte nicht wiederverwendbar. Da aber eine Instanz von `Presentable` seine Daten nicht direkt kapseln muß, sondern diese referenzieren kann, ist zumindest eine Wiederverwendung der Daten eines Medienobjektes möglich.

Man kann in einer MHEG-5-Applikation über die elementaren Aktionen `Launch` oder `Spawn` den Aufruf einer anderen MHEG-5-Applikation modellieren. Es ist also möglich, von einem MHEG-5-Dokument aus ein anderes Dokument aufzurufen und dieses dadurch wiederzuverwenden.

Es bleibt noch zu klären, inwieweit MHEG-5 die Wiederverwendung von Fragmenten eines multimedialen Dokuments erlaubt. Es gibt in MHEG-5 innerhalb einer Applikation nur eine Strukturierungsmetapher, die Szene. Eine Szene ist eine Gruppe und damit auch eindeutig adressierbar. Eine Szene kann damit über die elementare Aktion `TransitionTo` prinzipiell wiederverwendet werden. Da eine Szene aber keine weiteren Szenen enthalten kann, ist ein Autor bei der Gliederung seines Dokumentes auf die Flache szenenbasierte Strukturierung beschränkt. Sowohl logisch

zusammenhängende Einheiten mit feinerer Granularität als eine Szene als auch solche, die mehrere Szenen umfassen, sind nicht modellierbar.

Problematisch ist die Wiederverwendbarkeit von Szenen, die Ingredients einer Instanz von `Application` referenzieren (zum Beispiel Variablen). Diese setzen das Vorhandensein gewisser Ingredients in der Applikation voraus, die gerade präsentiert wird. Es ist nicht möglich, solche Szenen aus einer Applikation herauszutrennen und in einer anderen Applikation zu verwenden, deren Instanzen von `Application` über diese Ingredients nicht verfügen. Damit ist nur die Wiederverwendung solcher Szenen realisierbar, die solche Ingredients nicht referenzieren.

Ein weiterer Aspekt der Wiederverwendbarkeit betrifft deren Art. Da MHEG-5 weniger die Struktur eines Dokumentes als vielmehr dessen Präsentation beschreibt, ist das Aussehen einer Applikation oder Szene schon exakt festgelegt. Wiederverwendbarkeit in MHEG-5 ist somit nur identisch. Schließlich bleibt noch zu bemerken, daß MHEG-5 keinerlei Unterstützung für eine softwaregestützte Selektion wiederverwendbarer Szenen und Applikationen bietet.

### Interaktion

Eine weitere Forderung aus Kapitel 3 ist die Unterstützung der Modellierung von Interaktion in multimedialen Dokumenten. Dabei sollten sowohl die Beschreibung von navigierender als auch von gestalterische Einflußnahme auf die Präsentation eines Dokumentes durch den Betrachter möglich sein. Hierzu bietet MHEG-5 verschiedene Klassen, die Interaktionsformen zur Verfügung stellen (z.B. Buttons, Sliders, Hypertext und Eingabefelder). Die Interaktionsauswirkungen sind von diesen Interaktionsformen getrennt und lassen sich durch Instanzen der Klasse `Link` spezifizieren. Da das Feuern eines Links eine Folge elementarer Aktionen von MHEG-Objekten auslöst, ist die Modellierung von navigierender Interaktionen (z.B. durch `TransitionTo`-Aktionen bei Szenen bzw. `Launch`-Aktionen bei Applikationsobjekten), aber auch gestalterischer Interaktionen (beispielsweise durch `SetVolume`-Aktionen bei Audioobjekten oder `SetFontRef`-Aktionen bei Textobjekten) möglich.

### Adaption

Außerdem ist zu untersuchen, inwieweit MHEG-5 die Modellierung von Adaption in multimedialen Dokumenten erlaubt. Der MHEG-5-Standard bietet die Klasse `Variable` und deren Unterklassen an. Es ist möglich, den Verlauf einer Präsentation einer MHEG-5-Applikation von den Inhalten von Instanzen dieser Klassen abhängig zu machen. Dieses geschieht durch Instanzen der Klasse `Link`, welche die Inhalte der Variablen testen und entsprechende Aktionen einleiten können. Links können insbesondere dazu dienen, zwischen zwei Varianten eines multimedialen Dokumentes anhand von Variableninhalten zu wählen. Das könnte man zur Modellierung von clientbasierter Adaption nutzen, indem das Benutzerprofil in Form von Variablen im multimedialen Dokument modelliert wird. Problematisch ist jedoch, wie diese Variablen mit den zum Betrachter des Dokumentes passenden Werten belegt werden. Der MHEG-5-Standard erlaubt nur die Manipulation der Variablen in der MHEG-5-Applikation selbst, d.h. die Bestimmung des Profils muß Teil des multimedialen Dokuments sein. Diese Profilermittlung ist damit nicht unsichtbar für den Betrachter und muß außerdem bei einer erneuten Präsentation des multimedialen Dokumentes wiederholt werden, weswegen man nicht von echter Adaption sprechen kann. Einen Ausweg bietet der MHEG-6 Standard. Hier ist der Aufruf ei-

| | | |
|---|---|---|
| Wiederverwendbarkeit | Granularität | Wiederverwendbarkeit der Daten von Medienobjekten und vollständigen Dokumenten. Bedingt können Fragmente in Form von einzelnen Szenen wiederverwendet werden. |
| | Art | Identisch. |
| | Selektion | Keinerlei Unterstützung für softwaregestütze Suche von wiederverwendbaren Komponenten. |
| Interaktion | | Navigierende und gestalterische Interaktionen modellierbar. |
| Adaption | | Mit MHEG-6 clientbasierte Adaption möglich. |
| Präsentationsneutralität | | Spezifikation von Dokumenten auf niedriger semantischer Ebene. Dadurch schlecht geeignet zur präsentationsneutralen Verwaltung von Dokumenten. |

Tabelle 4.1: Zusammenfassung der Bewertung von MHEG-5

nes Java-Programms möglich, welches das Benutzerprofil ermittelt (z. B. aus einer Datenbank) und die entsprechenden Variablen initialisiert. Somit ist zumindest mit MHEG-6 clientbasierte Adaption möglich.

#### Präsentationsneutralität

Die letzte Forderung aus Kapitel 3 betrifft die Nutzbarkeit eines Modells für die präsentationsneutrale Verwaltung von Dokumenten. Hierfür ist MHEG-5 weniger gut geeignet, da die Spezifikation eines MHEG-5-Dokumentes auf einem sehr niedrigen semantischen Niveau erfolgt. Die Spezifikation beschreibt nicht die Semantik und Struktur des Dokumentes, sondern gibt konkrete Präsentationsanweisungen. Böse Zungen nennen MHEG-5 in diesem Zusammenhang einen „Multimedia-Assembler".

## 4.2 SGML/ XML

SGML [Gol90] ist weniger ein multimediales Dokumentmodell, als vielmehr ein Standard, der es erlaubt, den Inhalt und die Struktur von (nicht unbedingt multimedialen) Dokumenten zu spezifizieren. Da einige multimediale Dokumentstandards auf SGML basieren, soll SGML hier näher beschrieben werden. Da SGML selbst kein multimediales Dokumentmodell darstellt, wird auf eine Bewertung gemäß der Anforderungen aus Kapitel 3 verzichtet.

### 4.2.1 Beschreibung

Die grundlegende Idee hinter SGML (Standard Generalized Markup Language) ist die strikte Trennung zwischen Struktur und Layout eines Dokumentes. Ein Autor er-

stellt den Inhalt eines Dokumentes und strukturiert diesen (zum Beispiel in Kapitel, Abschnitte etc.). Durch diese Strukturierung wird jedoch nicht dessen tatsächliche Präsentation bestimmt. Die Präsentation basiert zwar auf dieser Strukturierung, erfolgt aber in einem unabhängigen Schritt, eventuell auch durch andere Personen. Der Vorteil dieser Trennung von Struktur und Präsentation liegt darin, daß unterschiedliche Darstellungen und Sichten von demselben Dokument erstellt werden können.

Um die Struktur eines Dokumentes zu spezifizieren, werden *Markups* benutzt, die in SGML definiert und in das Dokument integriert werden. Jeder SGML-konforme Parser kann diese Markups in einem Dokument finden und deren Anordnung und damit auch die Struktur des Dokumentes wiedergeben.

Ein Markup spezifiert ein *Element*. Ein Element stellt einen logisch sinnvollen Bestandteil eines Dokumentes dar und wird durch ein *Start-* und ein *End-Tag* begrenzt. Die Bezeichnung der Tags gibt den *Typ* des Elementes wieder (zum Beispiel `TITLE` oder `AUTHOR`), während der vom Start- und vom End-Tag umschlossene Dokumentteil den *Inhalt* des Elementes bildet. Da Elemente verschachtelbar sind, kann der Inhalt eines Elementes weitere Elemente umfassen. Die Verschachtelung der Elemente eines Dokumentes bildet somit einen Baum. Abbildung 4.4 zeigt ein Beispiel für ein SGML-Markup. Start-Tags werden durch `<>` markiert und `</>` markiert ein End-Tag.

```
1    <DOCUMENT>
2    <TITLE> Cleopatra Schwartz </TITLE>
3    <AUTHOR> S. Spielbergo </AUTHOR>
4    <PRODUCER> Samuel L. Bronkowitz </PRODUCER>
5    </DOCUMENT>
```

Abbildung 4.4: Beispiel für ein SGML-Markup

SGML erlaubt außerdem die Parametrisierung von Elementen über *Attribute*. So ist als Alternative zu Abbildung 4.4 auch Abbildung 4.5 möglich.

```
1    <DOCUMENT Title = "Cleopatra Schwartz" Author = "S. Spielbergo"
2              Producer = "Samuel L. Bronkowitz">
3    </DOCUMENT>
```

Abbildung 4.5: Beispiel für ein SGML-Element mit Attributen

Regeln im Vorspann eines Dokumentes, die sogenannte *Document Type Definition* (DTD), legen fest, aus welchen Elementen ein Dokument bestehen kann und wie diese zu verschachteln sind[2]. Die DTD besteht aus erweiterten regulären Ausdrücken und spezifiziert die Form (Syntax) der Elemente, nicht jedoch deren Bedeutung (Semantik). Die Semantik der Elemente muß dem Autor bekannt sein. Ein SGML-konformer Parser kann anhand der DTD die Struktur eines Dokumentes auf deren Korrektheit überprüfen.

Die DTD für Abbildung 4.4 zeigt Abbildung 4.6. In diesem Fall besteht ein Element des Typs `DOCUMENT` aus der sequentiellen Folge von Elementen der Typen `TITLE` und `AUTHOR`, während die Angabe eines `PRODUCER`-Elementes optional ist. Elemente vom Typ `TITLE`, `AUTHOR` und `PRODUCER` können einen beliebigen Inhalt inklusive weiterer Markups umfassen.

[2] Die DTD kann in eine Datei ausgelagert werden. In diesem Fall muß im Vorspann des Dokuments die Datei referenziert werden.

```
1     <!ELEMENT DOCUMENT (TITLE, AUTHOR, PRODUCER?)>
2     <!ELEMENT TITLE (#PCDATA)>
3     <!ELEMENT AUTHOR (#PCDATA)>
4     <!ELEMENT PRODUCER (#PCDATA)>
```

Abbildung 4.6: Beispiel für SGML-DTD

Im Vergleich dazu sieht die DTD der parametrisierten Variante aus Abbildung 4.5 wie in Abbildung 4.7 aus.

```
1     <!ELEMENT DOCUMENT (#PCDATA)>
2     <!ATTLIST DOCUMENT
3               Title    CDATA #REQUIRED
4               Author   CDATA #REQUIRED
5               Producer CDATA>
```

Abbildung 4.7: Beispiel für SGML-DTD mit Attributen

SGML bietet einen einfachen Referenzierungsmechanismus an. Jedem SGML-Elementtyp kann ein Attribut vom Typ `ID` zugeordnet werden. Dieses Attribut erlaubt es, Elementen einen innerhalb des Dokumentes eindeutigen Bezeichner zuzuordnen. Elemente mit einem solchen Bezeichner können von anderen Elementen referenziert werden. Dazu können Elementtypen Attribute des Typs `IDREF` definieren. Diesen Attributen können die Bezeichner anderer Elemente zugewiesen werden. Ein SGML-Parser kann automatisch prüfen, ob alle Referenzen auf gültige Elemente verweisen. Dazu betrachte man das Beispiel in Abbildung 4.8.

```
1     <!ELEMENT CHAPTER (#PCDATA)>
2     <!ATTLIST CHAPTER
3               id    ID #REQUIRED
4               title CDATA>
5
6     <!ELEMENT REF (#PCDATA)>
7     <!ATTLIST REF
8               target IDREF #REQUIRED>
9
10    ...
11
12    <CHAPTER id="chap1" title="Wichtige Regisseure">
13
14    ...
15
16    </CHAPTER>
17
18    ...
19
20    <!-- Eine gueltige Referenz -->
21
22    Ein <REF target="chap1"> wichtiger Regisseur </REF>
23    ist S. Spielbergo.
24
25    <!-- Eine ungueltige Referenz, chap2 ist nicht definiert -->
26
27    Man beachte auch die Dokumentation
28    <REF target="chap2"> Zinkoxid und Du </REF>.
```

Abbildung 4.8: Beispiel für Referenzierung in SGML

Desweiteren erlaubt SGML die Referenzierung von Entitäten, die nicht im Dokument selbst enthalten sind, zum Beispiel Bilder, Videos oder andere SGML-Dokumente. Dazu ist es möglich, im Vorspann eines Dokumentes externe Entities zu deklarieren (siehe Abbildung 4.9). Auf diese Entitäten kann über Attribute des Typs `ENTITY` Bezug genommen werden.

```
1     <!ENTITY Video
2               SYSTEM "http://www.bronkowitz.com/schwartz.mpeg"
3               NDATA "mpeg">
```

Abbildung 4.9: Beispiel für externe Entitäten in SGML

SGML verfügt noch über eine große Zahl weiterer Features, auf die der Kompaktheit der Darstellung wegen nicht näher eingegangen werden kann. Die Vielzahl der Möglichkeiten, die SGML bietet, macht es für Softwarehersteller schwierig, diesen Standard vollständig zu implementieren. Dies führte zur Entwicklung der Extensible Markup Language (XML) [BD97, BPSM98]. XML stellt eine Teilmenge von SGML dar. Ziel von XML ist es, SGML so zu vereinfachen, daß die Implementierung XML-konformer Systeme im Vergleich zu SGML-konformen Systemen wesentlich erleichtert wird, ohne jedoch die Kompatibiltät zu SGML zu verlieren. In der Tat ist jedes XML-konforme Dokument auch SGML-konform. Die in diesem Abschnitt vorgestellten Konzepte von SGML gelten ohne Einschränkungen auch für XML.

## 4.3 HyTime

HyTime [DD94, NKN91] ist ein Standard zur Beschreibung der Struktur von multimedialen Dokumenten. Aufbauend auf SGML bietet HyTime eine Sammlung von Konstrukten an, die eine hypertextartige Verknüpfung von Medienobjekten erlauben, ohne jedoch deren Art oder Kodierungsformat festzulegen. HyTime gestattet so die Integration von unterschiedlichsten Medienobjekten zu multimedialen Dokumenten. Unter anderem ist HyTime für folgende Zwecke verwendbar:

- Für die Verknüpfung von SGML-Elementen mit anderen Elementen, sowohl innerhalb eines Dokumentes, als auch zwischen verschiedenen SGML-Dokumenten.
- Für die Verknüpfung von SGML-Elementen mit Medienobjekten.
- Für die zeitliche und räumliche Koordination von verschiedenen Medienobjekten.
- Für die Produktion von multimedialen Dokumenten. Es ist möglich, innerhalb eines Dokumentes Vorgehensweisen und Verantwortlichkeiten für Änderungen des Dokuments zu spezifizieren. Außerdem können Zugriffsrechte auf Teile eines Dokumentes festgelegt werden.

### 4.3.1 Beschreibung

Da HyTime auf SGML aufbauen soll, wurde bei der Entwicklung des Standards zunächst versucht, diesen in Form einer DTD zu spezifizieren. Dieser Ansatz wurde jedoch aus verschiedenen Gründen wieder verworfen. Zum einen sollen bereits existierende SGML-Dokumente einfach nach HyTime überführt werden können. Dadurch ergibt sich bei der Definition einer DTD die Gefahr von Namenskonflikten mit Elementen des zu überführenden Dokuments. Zum anderen kann es nötig sein, feinere Unterscheidungen zwischen Elementen zu treffen, als bei der Erstellung des

Standards vorhersehbar ist. So kann es Dokumentarten geben, bei denen die Definition von 10 unterschiedlichen Link-Typen sinnvoll ist, die alle unterschiedlich verarbeitet werden, während wiederum andere Dokumente mit einem einzigen Link-Typ auskommen. Es ist in einem solchen Fall praktisch, einen Vererbungsmechanismus zur Verfügung zu stellen, was mit einer DTD nicht möglich ist.

Anstelle der Definition einer DTD führt HyTime die sogenannten *Architectural Forms* (AF) ein. Eine AF ist ein HyTime-Element mit einem Namen sowie einer definierten Menge an Attributen. In der objektorientierten Terminologie kann man eine AF als abstrakte Basisklasse auffassen. Eine AF wird durch einstufiges Subclassing benutzt. Dazu wird in der dokumentspezifischen DTD einem SGML-Element ein Attribut `HyTime` hinzugefügt und mit dem Namen der gewünschten AF belegt. Desweiteren werden diesem Element die verwendeten Attribute der AF hinzugefügt. Zusätzlich kann das Element noch weitere Attribute definieren, um eine dokumentspezifische, von der AF abweichende Semantik zu unterstützen.

Das Beispiel in Abbildung 4.10 zeigt die Verwendung der AF `clink`, die eine einfache Hyperlink-Semantik zur Verfügung stellt, zur Definition eines dokumentspezifischen Link-Elementes. Ein `clink` drückt aus, daß der von ihm umschlossene Dokumentteil, der Anker, in Beziehung zu einem anderen SGML-Element steht, dem Link-Ziel. Das Attribut `linkend` ist von der AF vorgegeben und muß eine Referenz auf das Link-Ziel enthalten.

```
1     <!ELEMENT MYLINK (#PCDATA)>
2     <!ATTLIST MYLINK
3               HyTime  NAME  #FIXED "clink"
4               linkend IDREF #REQUIRED>
```

Abbildung 4.10: Beispiel zur Verwendung einer AF in einer DTD

Eine Anwendung des dokumentspezifischen Link-Elementes aus Abbildung 4.10 zeigt Abbildung 4.11.

```
1     Der Produzent des Filmklassikers ,,F"ur eine handvoll Yen'' ist
2     <MYLINK linkend="samuel"> Samuel L. Bronkowitz </MYLINK>.
```

Abbildung 4.11: Beispiel zur Verwendung eines Link-Elementes

Der HyTime Standard ist modular aufgebaut. So müssen nur die Moduln des Standards für eine Anwendung implementiert werden, die auch wirklich benötigt werden. Die Abbildung 4.12 zeigt die verschiedenen Moduln des HyTime-Standards und ihre Abhängigkeiten untereinander [NKN91].

In den folgenden Abschnitten sollen die einzelnen Moduln näher beschrieben werden.

### Base-Modul

Das Basismodul definiert die grundlegenden Konzepte von HyTime. Dazu gehört, neben SGML selbst, das Konzept der Architectural Forms. Außerdem wird eine Grundmenge an Architectural Forms definiert, die in den anderen HyTime-Moduln gebraucht werden, wie zum Beispiel die `HyDoc`-AF. HyTime stellt nahezu keine Forderungen an ein SGML-Dokument, damit dieses ein auch korrektes HyTime-

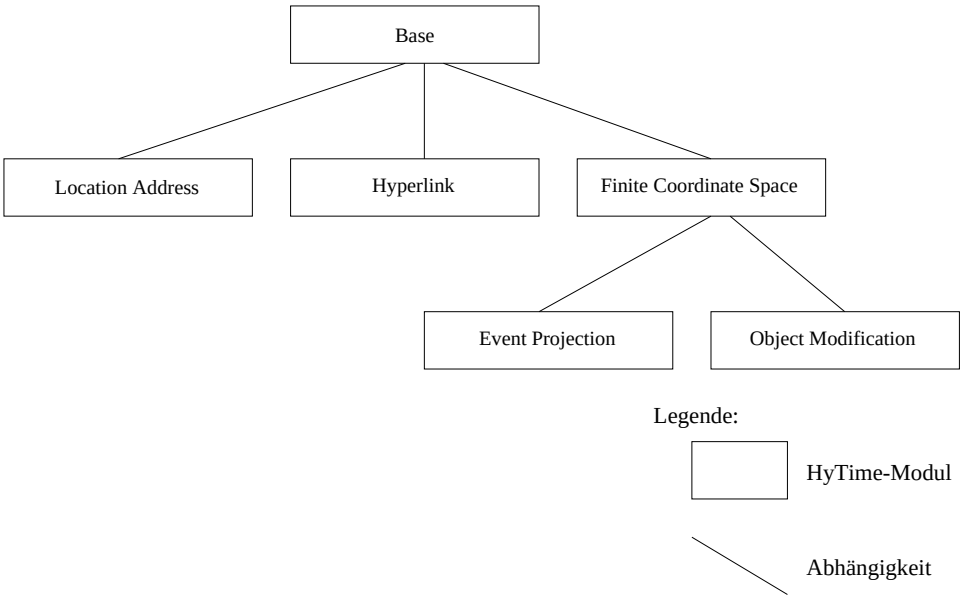


Abbildung 4.12: Die Moduln des HyTime-Standards

Dokument bildet. Eine Forderung ist jedoch, daß das Wurzelelement des Dokumentes von der `HyDoc`-AF erbt.

### Location-Address-Modul

Wie schon erläutert, verfügt SGML über einen Mechanismus, der es erlaubt, Elemente mit einem eindeutigen Bezeichner zu versehen und über diesen Bezeichner zu adressieren. Problematisch hierbei ist jedoch, daß diese Bezeichner nur innerhalb eines Dokumentes eindeutig und bekannt sind. Folglich können keine Elemente aus anderen Dokumenten referenziert werden. Außerdem können externe Entitäten, zum Beispiel Videos oder Bilder, nicht mit einem solchen Bezeichner versehen werden. Somit sind externe Entitäten nicht wie Elemente referenzierbar, obwohl dieses im Bereich multimedialer Dokumente wünschenswert ist.

Um diese Probleme zu lösen, bietet HyTime die sogenannten *Locators* an. Locators sind Indirektionen, die mit einem eindeutigen Bezeichner versehen werden und die zu referenzierende Information kapseln. Referenziert man einen Locator, so erhält man keinen Verweis auf den Locator selbst, sondern auf die von diesem gekapselte Information. HyTime bietet verschiedene Arten von Locators an, von denen der *Name Locator*, definiert durch die `nameloc`-AF, wohl der gebräuchlichste ist. Er erlaubt die indirekte Adressierung von Elementen bzw. Entitäten über deren Bezeichner, wie anhand eines Beispiels in Abbildung 4.13 demonstriert[3].

Weiterhin können mehrere Locators zu sogenannten *Location Ladders* kombiniert werden. Eine Location Ladder ist eine mehrstufige Indirektion. So wird im von Abbildung 4.14 gezeigten Beispiel ein Element mit dem Bezeichner `chapter2` in einem externen Dokument referenziert.

Neben dem Name Locator definiert HyTime noch weitere Locators. Der *Tree Loca-*

[3] Im folgenden wird der Einfachheit halber davon ausgegangen, daß zu jeder Architectural Form in der DTD des Dokumentes ein gleichnamiger Elementtyp definiert ist.

```
1   <!ENTITY for-a-fistfull-yen SYSTEM "ffy.mpeg" NDATA "mpeg">
2
3   ...
4
5   <NAMELOC id="ffy">
6   <NMLIST nametype="entity">
7   for-a-fistfull-yen
8   </NMLIST>
9   </NAMELOC>
10
11  ...
12
13  <!-- Nachstehende Referenz bezieht sich direkt auf die Entity
14       for-a-fistfull-yen und nicht auf das NAMELOC-Element! -->
15
16  Schauen Sie sich dieses <REF target="ffy"> Meisterwerk </REF> an.
```

Abbildung 4.13: Beispiel zur Verwendung eines Name Locators

```
1   <!ENTITY book SYSTEM "book.sgm" NDATA "SGML">
2
3   ...
4
5   <NAMELOC id="book">
6   <NMLIST nametype="entity">
7   book
8   </NMLIST>
9   </NAMELOC>
10
11  <!-- Das Attribut locsrc spezifiziert, dass der nachfolgende Locator
12       relativ zum Locator book zu sehen ist -->
13
14  <NAMELOC id="chapter2">
15  <NMLIST nametype="element" locsrc="book">
16  chapter2
17  </NMLIST>
18  </NAMELOC>
19
20  ...
21
22  Diese Aussage bezieht sich auf <REF target="chapter2">
23  Kapitel 2 </REF> <REF target="book"> meines Buches </REF>.
```

Abbildung 4.14: Beispiel für eine Location Ladder

*tor* erlaubt beispielsweise die Adressierung von Elementen über deren Position im Verschachtelungsbaum der Elemente eines SGML-Dokumentes. Der *FCS Locator* dient zur Referenzierung von Teilen eines Medienobjektes über die Definition des gewünschten Bereiches in Form von Koordinatenangaben eines über das Medienobjekt gelegten Koordinatensystems. So wird in Abbildung 4.15 von einem Video der Ausschnitt beginnend mit Frame 18000 und der Dauer von 9000 Frames referenziert.

Der mächtiger Locator ist der *Query Locator*, der über die AF `nmquery` bereitgestellt wird. Dieser erlaubt, das Ergebnis einer Anfrage in einer Query-Sprache zu referenzieren. HyTime definiert eine eigene Query-Sprache für SGML-Dokumente, genannt HyQ. Jedoch können auch andere Sprachen definiert und mit der AF `nmquery` verwendet werden. Es gibt noch weitere Locators, die hier jedoch nicht weiter behandelt werden sollen.

### Hyperlink-Modul

HyTime bietet mehrere *Link*-Arten an. Ein Link verbindet ein oder mehrere Informationsobjekte, zum Beispiel SGML-Elemente oder externe Entitäten. Es ist jedoch

```
<!ENTITY for-a-fistfull-yen SYSTEM "ffy.mpeg" NDATA "mpeg">

...

<NAMELOC id="ffy">
<NMLIST nametype="entity">
for-a-fistfull-yen
</NMLIST>
</NAMELOC>

<!-- video-fcs ist ein diskreter, eindimensionaler Koordinatenraum,
     mit Video-Frames als Punkten -->

<FCSLOC id="movie" locsrc="ffy" impfcs="video-fcs">
<DIMSPEC>
18000 9000
</DIMSPEC>
</FCSLOC>
```

Abbildung 4.15: Beispiel für einen FCS Locator

keine exakte Semantik vorgegeben. Ein Link drückt lediglich aus, daß Informationsobjekte zueinander in Beziehung stehen. Von welcher Art diese Beziehung ist und wie sich eine Anwendung zu verhalten hat, die auf einen solchen Link trifft, wird nicht definiert. Eine HyTime-Implementierung hat lediglich dafür zu sorgen, daß ein Link traversiert werden kann.

Die `clink`-AF wurde schon vorgestellt. Ein `clink` verbindet ein Ankerelement mit einem Zielelement. Er wird beim Anker deklariert. HyTime verfügt aber auch über komplexere Links. So verbindet der `ilink` eine beliebige Zahl von Elementen. Jedem Element, das an solch einem Link beteiligt ist, kann ein Rollenname zugewiesen werden, um eine sinnvolle Traversierung zu ermöglichen. Da mehrere Elemente gleichwertig an einem `ilink` beteiligt sind, kann dieser im Gegensatz zum `clink` nicht mehr sinnvoll einem Element zugordnet werden. Deshalb wird ein `ilink` unabhängig von den beteiligten Elementen deklariert, wie Abbildung 4.16 zeigt.

```
<!ENTITY for-a-fistfull-yen-1 SYSTEM "ffy1.mpeg" NDATA "mpeg">
<!ENTITY for-a-fistfull-yen-2 SYSTEM "ffy2.mpeg" NDATA "mpeg">
...

<NAMELOC id="ffy1">
<NMLIST nametype="entity">
for-a-fistfull-yen-1
</NMLIST>
</NAMELOC>

<NAMELOC id="ffy2">
<NMLIST nametype="entity">
for-a-fistfull-yen-2
</NMLIST>
</NAMELOC>

...

<ILINK linkends="ffy1 ffy2" anchrole="Vorgaenger Nachfolger">
</ILINK>
```

Abbildung 4.16: Beispiel für einen `ilink`

Es gibt noch weitere Linkarten, auf die hier jedoch der Kürze wegen nicht näher eingegangen wird.

### Finite-Coordinate-Space-Modul

Das *Finite-Coordinate-Space-Modul* (FCS-Modul) stellt Mittel zur Verfügung, um die koordinierte Präsentation von Medienobjekten zu beschreiben. Dazu kann ein $n$-dimensionaler, endlicher Koordinatenraum definiert werden, der FCS. Jeder Dimension des FCS kann eine Maßeinheit und eine Ausdehnung (in dieser Maßeinheit) zugewiesen werden. HyTime definiert eine Menge von Standardmaßeinheiten, wie z.B. Sekunden und Meter. Von diesen Standardmaßeinheiten können außerdem neue Einheiten abgeleitet werden.

Wenn der FCS der Präsentation definiert ist, kann ein Medienobjekt in diesen Raum an einem festen Punkt mit einer festen Ausdehnung positioniert werden. Diese Positionierung nennt man *Event*. Mehrere Events können zu sogenannten *Event Schedules* gruppiert werden. Ein solcher Event Schedule beschreibt damit den Ablauf der Präsentation. Da schon bei Definition der Präsentation sämtliche Events (und damit auch die Präsentationszeitpunkte von Medienobjekten) festgelegt werden müssen, ist es nicht möglich, interaktive Präsentationen zu beschreiben. Interaktionszeitpunkte des Benutzers sind nämlich nicht vorhersehbar und können so nicht vorab definiert werden. Abbildung 4.17 zeigt die Definition einer einfachen Musikpräsentation.

```
1     <!-- Medienobjekte definieren -->
2
3     <!ENTITY paradise-lost SYSTEM "Gothic.mp3" NDATA "MPEG3">
4     <!ENTITY tiamat        SYSTEM "Gaia.mp3" NDATA "MPEG3">
5     <!ENTITY moonspell     SYSTEM "RavenClaws.mp3" NDATA "MPEG3">
6
7     <!-- Zeitachse definieren -->
8
9     <!ELEMENT TIME EMPTY>
10    <!ATTLIST TIME
11              HyTime      NAME  #FIXED "axis"
12              axismeas    CDATA #FIXED "SISECOND"
13              axisdim     CDATA #FIXED "1000">
14
15    <!-- Eindimensionalen FCS definieren -->
16
17    <!ELEMENT MUSICFCS (EVSCHED+)>
18    <!ATTLIST MUSICFCS
19              HyTime      NAME  #FIXED "fcs"
20              id          ID    #IMPLIED
21              axisdefs    NAMES #FIXED "TIME">
22
23    <!-- Definition entsprechender Elemente fuer folgende AF:
24         event, evsched, extlist -->
25
26    ...
27
28    <!-- Event-Schedule definieren -->
29
30    <MUSICFCS>
31      <EVSCHED id="Musikpraesentation" axisord="TIME">
32        <EVENT data="paradise-lost" exspec="song1"></EVENT>
33        <EVENT data="tiamat" exspec="song2"></EVENT>
34        <EVENT data="moonspell" exspec="song3"></EVENT>
35      </EVSCHED>
36    </MUSICFCS>
37
38    <!-- Es fehlen noch die genauen Positionen und Ausdehnungen
39         der Lieder im FCS -->
40
41    <EXTLIST id="song1"><DIMSPEC>1 283</DIMSPEC></EXTLIST>
42    <EXTLIST id="song2"><DIMSPEC>300 688</DIMSPEC></EXTLIST>
43    <EXTLIST id="song3"><DIMSPEC>700 897</DIMSPEC></EXTLIST>
```

Abbildung 4.17: Beispiel für einen Event Schedule

#### Die Moduln Event-Projection und Object-Modification

Das Event-Projection-Modul definiert Mittel, um Events aus einem Event Schedule in einen anderen Event Schedule einzublenden. Dieses nennt man Projektion. Mit Hilfe von Projektionen ist es beispielsweise möglich, eine veränderte Abspielgeschwindigkeit eines Videos zu spezifizieren.

Das Object-Modification-Modul bietet Mittel und Wege, Events in einem FCS zu beeinflussen. Beispiele hierfür sind die Definition von Überblendeffekten, von Farbpaletten oder der Lautstärke von Musikstücken.

### 4.3.2 Bewertung

Es stellt sich nun die Frage, ob HyTime nach den Anforderungen aus Kapitel 3 ein flexibles multimediales Dokumentmodell darstellt. Dabei werden in diesem Abschnitt die einzelnen Forderungen betrachtet und HyTime bezüglich dieser bewertet. Das Ergebnis der Bewertung ist in Tabelle 4.2 auf Seite 37 zusammengefaßt.

#### Wiederverwendbarkeit

Zunächst ist zu untersuchen, ob und mit welcher Granularität HyTime Wiederverwendbarkeit unterstützt. Wie schon in der obigen Beschreibung des HyTime-Standards ausführlich erläutert, bietet das Location-Address-Modul den Mechanismus der Locators an. Mit diesen können einerseits externe Entitäten referenziert werden. Als externe Entitäten sind sowohl Medienobjekte als auch vollständige SGML- oder HyTime Dokumente zu sehen. Andererseits ist es aber auch möglich, über Location Ladders Teile von externen Entitäten zu referenzieren, wie zum Beispiel einzelne SGML-Elemente eines anderen Dokuments. Ist dieses Dokument ein HyTime-Dokument, so entspricht dies der Referenzierung von Bestandteilen eines multimedialen Dokumentes. Somit kann man Fragmente adressieren. HyTime erlaubt damit die Wiederverwendbarkeit auf allen in Kapitel 3 geforderten Abstraktionsebenen.

Die nächste Frage ist, ob die Wiederverwendung in HyTime identisch oder semantisch erfolgt. Da HyTime auf SGML aufsetzt und wie SGML die Struktur eines multimedialen Dokumentes beschreibt, ist die Art der Wiederverwendung als semantisch zu charakterisieren.

Es bleibt noch zu klären, inwieweit HyTime die softwaregestützte Selektion von wiederverwendbaren Komponenten unterstützt. Dadurch, daß HyTime keine eigene DTD definiert, sondern das Konzept der Architectural Forms verfolgt, ist es möglich, eine anwendungspezifische DTD zu definieren. Im Rahmen dieser DTD kann man Attribute definieren, die zur Klassifikation der wiederverwendbaren Komponenten und zur Unterstützung einer softwaregestützten Suche dienen.

#### Interaktion

Eine weitere Forderung, die ein flexibles Dokumentmodell erfüllen muß, ist die Unterstützung der Modellierung navigierender und gestalterischer Interaktion. Bei der Beschreibung einer Präsentation in HyTime müssen vorab alle räumlichen und zeitlichen Positionen der Auftretenden Medienobjekte in einem Event Schedule defi-

niert werden. Dieses schließt navigierende Interaktion aus, da das Verhalten eines Benutzers nicht vorausgesehen werden kann und man nicht weiß, wie der Benutzer durch das Dokument navigiert. Es gibt in der Literatur Ansätze, dieses Problem zu lösen. So erweitert Wirag [WRW94] HyTime um einen neuen Link-Typ. Dieser verbindet mehrere alternative Event Schedules, zwischen denen abhängig von Benutzerinteraktionen hin- und hergeschaltet wird. Jedoch können sich bei einem komplexen Dokument sehr viele Interaktionsmöglichkeiten ergeben, so daß die Zahl der zu definierenden Schedules sehr groß werden kann.

HyTime bietet zwar die Mittel der Moduln Object-Modification und Event-Projection an, um die Präsentation von Medienobjekten gestalterisch zu beeinflussen. Jedoch haben diese Mittel keine interaktive Semantik. Folglich unterstützt HyTime keine gestalterische Interaktion. Wirag [WRW94] integriert zur Lösung dieses Problems in HyTime eine Skriptsprache.

#### Adaption

HyTime schreibt einer Anwendung keine DTD zur Beschreibung der Dokumente vor. So ist es durch die Definition einer geeigneten DTD ohne weiteres möglich, die wiederverwendbaren Komponenten eines Dokuments so zu attributieren, daß Informationen über deren Inhalt oder physikalischen Eigenschaften (benötigte Bandbreite usw.) zur Verfügung stehen. Gleichzeitig bietet HyTime die Query Locators, so daß man Elemente mit Anfragen in einer Query-Sprache indirekt referenzieren kann. Diese Anfragen können natürlich auch zur Adressierung von Elementen über deren, in Attributen beschriebenen, inhaltliche oder physikalische Eigenschaften dienen. Das Problem ist jedoch, daß die Ergebnisse solcher Abfragen vollständig durch den Dokumentinhalt bestimmt sind. Es ist nicht möglich, durch äußere Einfluße, wie zum Beispiel ein Benutzerprofil, das Ergebnis einer Query zu zu verändern. Damit kann ein HyTime-Dokument nicht an äußere Parameter wie ein Benutzerprofil adaptiert werden.

#### Präsentationsneutralität

Da eine HyTime-Spezifikation die Struktur und Semantik eines multimedialen Dokumentes beschreibt und weniger konkrete Präsentationsanweisungen gibt, ist das semantische Niveau einer HyTime-Dokumentbeschreibung recht hoch. Deshalb ist HyTime gut zur präsentationsneutralen Ablage von multimedialen Dokumenten geeignet.

## 4.4 SMIL

Die Synchronized Multimedia Integration Language (SMIL) [PBD+98] ist ein auf XML basierender Standard des WWW-Konsortiums. Er behandelt die Integration von unabhängigen Medienobjekten zu multimedialen Dokumenten. Bei der Definition des Standards[4] werden mehrere Ziele verfolgt:

[4] Zum Zeitpunkt der Erstellung dieser Arbeit lag der SMIL-Standard lediglich als zweiter Working Draft vor. Es können sich also noch wesentliche Änderungen zu den Ausführungen dieses Abschnittes ergeben.

| | | |
|---|---|---|
| Wiederverwendbarkeit | Granularität | Wiederverwendbarkeit von Medienobjekten, vollständigen Dokumenten und von Fragmenten. |
| | Art | Semantisch. |
| | Selektion | Im Rahmen einer anwendungspezifischen DTD können Elemente mit Attributen zur Klassifikation versehen werden. |
| Interaktion | | HyTime unterstützt keine Modellierung von Interaktion. |
| Adaption | | Nicht möglich. |
| Präsentationsneutralität | | Spezifikation von Dokumenten auf hoher semantischer Ebene. Dadurch gut geeignet zur präsentationsneutralen Verwaltung von Dokumenten. |

Tabelle 4.2: Zusammenfassung der Bewertung von HyTime

- SMIL-Dokumente sollen internetfähig sein. Das meint, daß sowohl die im Dokument vorkommenden Medienobjekte, als auch die Dokumente selbst auf Web-Servern gehalten werden. Somit erfolgt die Adressierung von Medienobjekten und Dokumenten über die vom World Wide Web bekannten Uniform Resource Locators (URL). Außerdem geschieht die Übertragung von SMIL-Dokumenten über das Hypertext Transfer Protocol (HTTP).
- SMIL-Dokumente sollen leicht erstellbar sein. Mit einem einfachen Texteditor sollen Autoren in der Lage sein, ansprechende Dokumente zu erstellen.

### 4.4.1 Beschreibung

SMIL wird durch eine XML-DTD definiert. Diese DTD sieht vor, daß ein SMIL-Dokument aus zwei Teilen besteht: dem *Document Head* und dem *Document Body*.

#### Head

Im Document Head (siehe Abbildung 4.18) können einerseits Metainformationen über das Dokument angegeben werden. Dazu dient das *Meta-Element*, mit dem Attribute definiert und mit Werten belegt werden können. Diese Attribute können beispielsweise zur Unterstützung einer softwaregestützten Suche nach Dokumenten dienen.

Andererseits wird im Document Head das räumliche Layout des Dokumentes definiert. Prinzipiell kann eine beliebige Layout-Spezifikationssprache hierfür verwendet werden, jedoch wird bereits ein einfacher Spezifikationsmechanismus im SMIL-Standard vorgeschlagen. Dieser unterteilt den Bereich, in dem das multimediale Dokumente präsentiert wird (in der Regel ein Bildschirmfenster) in *Channels*. Ein Channel ist eine rechteckige Region. Der Channel wird definiert über die Position seiner linken oberen Ecke, sowie seiner Höhe und Breite. Zusätzlich kann einem Channel eine Priorität, genannt z-Index, zugeordnet werden, die dazu dient, überlappende Channels richtig anzuordnen. Jeder Channel verfügt über eine eindeutige

ID. Jedes Medienobjekt referenziert eine solche ID, um festzulegen, in welchem Channel es präsentiert werden soll.

```
1     <HEAD>
2
3     <!-- Zuerst ein paar Metainformationen ueber das Dokument -->
4
5     <META name="Autor" content="Prof. Mueller"></META>
6     <META name="Art" content="Vorlesung"></META>
7     <META name="Thema" content="Der Einfluss von Kaffee auf den
8           Verlauf einer Herztransplantation"></META>
9
10    <!-- Definition des raeumlichen Layouts -->
11
12    <LAYOUT type="text/smil-basic">
13    <CHANNEL id="linke-praesentation" left="20" top="20" width="300"
14             height="300"></CHANNEL>
15    <CHANNEL id="rechte-praesentation" left="400" top="20" width="300"
16             height="300"></CHANNEL>
17    <CHANNEL id="untertitel" left="20" top="400" width="700"
18             height="200"></CHANNEL>
19    </LAYOUT>
20
21    </HEAD>
```

Abbildung 4.18: Beispiel für einen Document Head in SMIL

### Body

Der Document Body beschreibt den Ablauf der Präsentation des Dokumentes. Dazu steht eine Anzahl an XML-Elementtypen zur Verfügung, namentlich die *Schedule-Elemente*, das *Switch-Element* und die *Link-Elemente*.

- **Schedule-Elemente**

  Ein Schedule-Element (SE) beschreibt den temporalen Ablauf der Präsentation der von ihm umschlossenen Elemente. Jedes Schedule-Element kann über eine Reihe von Attributen verfügen:

  - `id`: Dieses Attribut dient zur eindeutigen Kennzeichnung eines SE. Andere SE können im Rahmen der zeitlichen Synchronisation ein SE über dieses Attribut referenzieren.
  - `repeat`: Mit diesem Attribut wird angegeben, wie oft die Präsentation des SE wiederholt wird.
  - `dur`: Diese Attribut spezifiziert die Dauer der Präsentation eines Elementes.
  - `fill`: Falls die Präsentation eines SE beendet ist, es aufgrund von Synchronisationsanweisungen aber noch auf die Beendigung der Präsentation anderer Schedule-Elemente warten muß, kann mit dem `fill`-Attribut das weitere Verhalten spezifiert werden. Ein SE kann beispielsweise seine Präsentation wiederholen, bis die anderen SE ihre Präsentation beendet haben oder aber seine Darstellung komplett beenden, d.h. vom Präsentationsbereich des Dokumentes verschwinden.

  SMIL definiert drei unterschiedliche Schedule-Elemente: das *Parallel-Element*, das *Sequential-Element* und das *Medienobjekt-Element*.

  Das Parallel-Element `<PAR>` definiert die parallele, synchronisierte Darstellung der von ihm umschlossenen Schedule-Elemente. Die Darstellung kann lippensynchron, aber auch lose synchronisiert erfolgen. Der Beginn der Präsentation

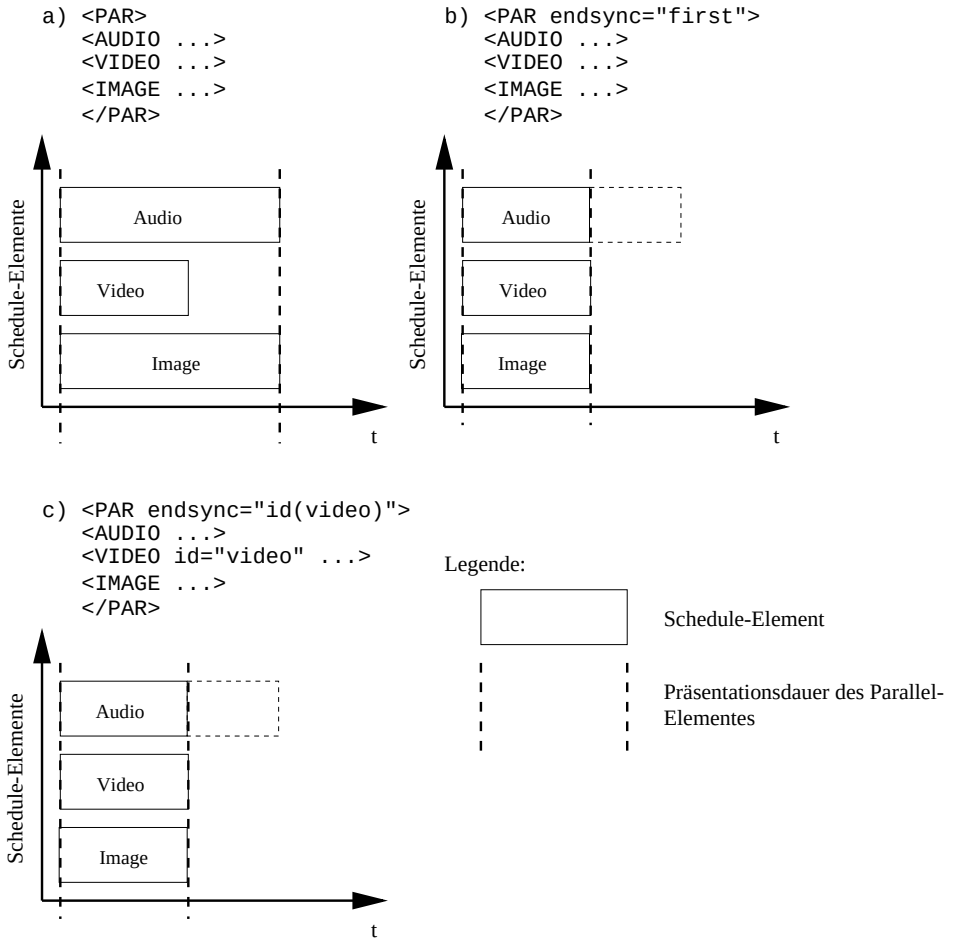


Abbildung 4.19: Präsentationsdauer des Parallel-Elementes von SMIL

eines Parallel-Elementes startet die von ihm umschlossenen SE. Die Darstellung eines Parallel-Elementes endet normalerweise, wenn die Präsentation des SE mit der längsten Dauer beendet ist (siehe Abbildung 4.19a), wobei SE mit nicht definierter Dauer (z.B. Bilder oder Text) nicht betrachtet werden. Dieses vordefinierte Verhalten läßt über das `endsync`-Attribut verändern. Zum Beispiel kann die Beendigung des Parallel-Elementes auf das Ende eines bestimmten SE (siehe Abbildung 4.19b) oder auf das Ende des SE mit der kürzesten (siehe Abbildung 4.19c) Präsentationsdauer festgelegt werden.

Das Sequential-Element `<SEQ>` definiert die sequentielle Präsentation der von ihm umschlossene Schedule-Elemente. Die Präsentation des Sequential-Elementes endet mit der Präsentation des letzten von ihm umschlossenen SE.

Das dritte von SMIL definiert Schedule-Element ist das Medienobjekt-Element. Dieses SE ist atomar und spezifiziert die Darstellung eines externen Medienobjektes. Es wird über den XML-Elementtyp `<REF>` realisiert, welcher zusätzlich folgende Attribute definiert:

- `src`: Dieses Attribut enthält die URL des externen Medienobjektes.
- `type`: Mit diesem Attribut wird die Art und das Format des Medienobjektes definiert, zum Beispiel „video/mpeg".
- `range`: Bei kontinuierlichen Medienobjekten kann hier der gewünschte

Zeitausschnitt angegeben werden, und zwar in Form der SMPTE- oder NPT-Formate [PBD+98].

- `channel`: Dieses Attribut bekommt die ID des Channels zugewiesen, im dem das Objekt präsentiert werden soll.

Das Beispiel in Abbildung 4.20 zeigt ein paar Medienobjekt-Elemente. Die XML-Elementtypen `<AUDIO>` und `<VIDEO>` sind lediglich Synonyme für `<REF>`.

```
1     <!-- Video "mueller.mpg" im Channel "rechte-praesentation"
2          darstellen -->
3
4     <REF id="vid1" src="mueller.mpg" type="video/mpeg"
5          channel="rechte-praesentation"></REF>
6
7     <!-- Video "mueller.mpg" im Channel "rechte-praesentation"
8          darstellen, jedoch nur ab Sekunde 10 bis Sekunde 100 -->
9
10    <VIDEO id="vid2" src="mueller.mpg" type="video/mpeg"
11           channel="rechte-praesentation"
12           range="npt:10s-100s"></VIDEO>
13
14    <!-- Audio "mueller.wav" abspielen, jedoch erst ab
15         Sekunde 100  -->
16
17    <AUDIO id="audio1" src="mueller.wav" type="audio/wav"
18           range="npt:100s-"></AUDIO>
```

Abbildung 4.20: Beispiele für Medienobjekte in SMIL

Um eine etwas feinere Synchronisation der Schedule-Elemente untereinander zu ermöglichen, definiert SMIL für SE zusätzlich das Synchronistationsattribut `begin`. Es kann mit einem Zeitoffset belegt werden. Mit diesem Attribut kann man eine Verzögerung vom Start eines Parallel-Elementes (siehe Abbildung 4.21a), eine Verzögerung vom Ende des vorhergehenden SE in einem Sequential-Element (siehe Abbildung 4.21b) sowie eine Verzögerung vom Start eines beliebigen SE innerhalb eines Parallel-Elementes (siehe Abbildung 4.21c) erreichen. Analog gibt es das Synchronisationsattribut `end`, das einen Zeitoffset bezüglich des Endes eines Parallel-Elementes darstellt.

- **Switch-Elemente**

  Neben den Schedule-Elementen kann man im Document Body eines SMIL-Dokumentes auch *Switch-Elemente* angeben. Switch-Elemente erlauben die Spezifikation von verschiedenen Varianten eines Dokumentes, von denen eine zur Präsentationszeit ausgewählt wird. Dazu sind definiert der SMIL-Standard spezielle, von außen setzbare Parameter, wie z.B. `bitrate` oder `language`. Die von einem Switch-Element umschlossenen Elemente werden vor der Präsentation des Dokumentes der Reihe nach durchgegangen. Haben diese Elemente ein Attribut mit demselben Namen wie ein Parameter, werden der spezifizierte Attributwert und der zugehörige Parameterwert miteinander verglichen. Stimmen sie überein, so wird diese Variante ausgewählt. In Abbildung 4.22 werden verschiedene Varianten eines Audioclips definiert, die sich lediglich in der benötigten Bitrate des Netzwerks unterscheiden. Je nach Bandbreite wird zur Präsentationszeit der passende Audioclip ausgewählt.

- **Link-Elemente**

  Schließlich erlaubt SMIL noch die Spezifikation von *Links*. Im Gegensatz zu HyTime ist ein Link mit einer konkreten Präsentationssemantik versehen. Ein Link verfügt über einen Anker, der die Elemente umschließt, welche die Interaktionsfläche des Links darstellen. Interagiert ein Benutzer während der

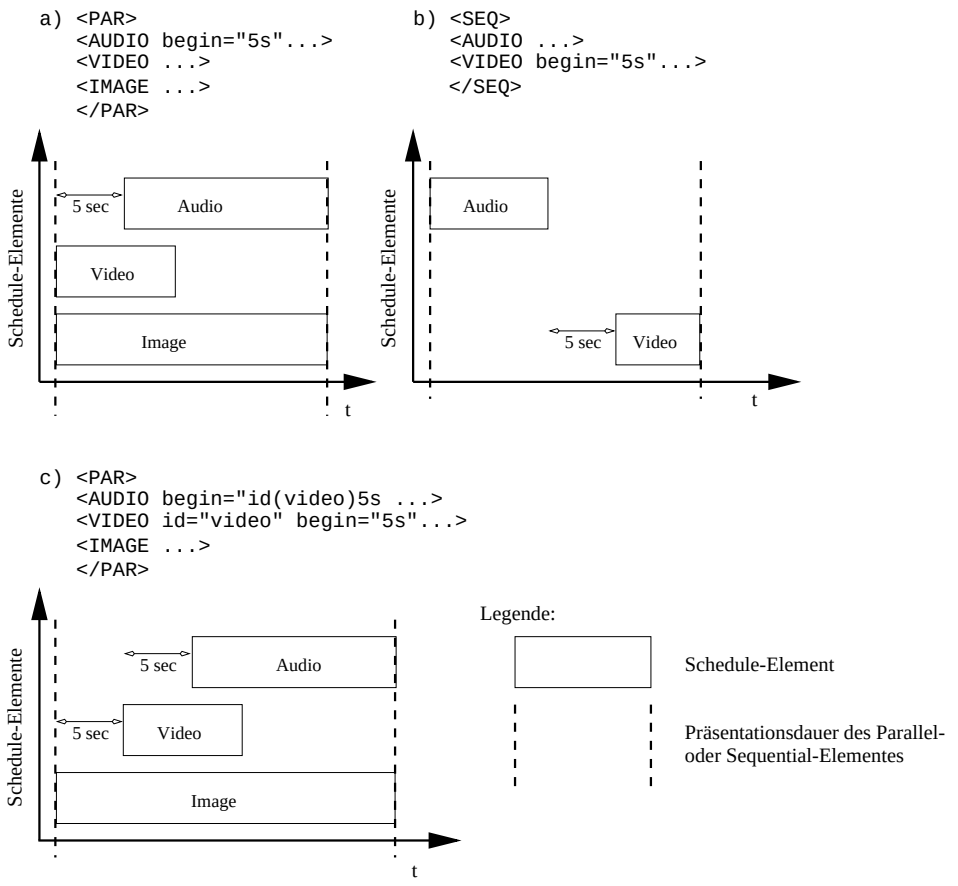


Abbildung 4.21: Synchronisationsattribute in SMIL

Präsentation eines SMIL-Dokumentes mit dieser Fläche (i.d.R. durch einen Mausklick), so wird der Link zum Link-Ziel hin traversiert.

Dabei werden drei Semantiken unterschieden. Die Traversierung kann zum einen die Präsentation des aktuellen Dokumentes beenden und die Präsentation des Link-Ziels einleiten. Zum anderen kann die Präsentation des aktuellen Dokumentes pausieren und die Präsentation des Link-Ziels gestartet werden. Nach Beendigung der Präsentation des Link-Ziels wird mit der Präsentation des ursprünglichen Dokumentes fortgefahren. Die dritte Semantik startet die Präsentation des Link-Ziels in einem neuen Präsentationskontext (zum Beispiel ein neues Bildschirmfenster), ohne jedoch die Darstellung des aktuellen Dokumentes zu beenden. Die Präsentationen der beiden Dokumente laufen vollkommen unabhängig voneinander weiter.

```
1  <PAR>
2  <IMG src="...>
3  <SWITCH>
4  <AUDIO src="mueller1.wav" bitrate="19200bps"></AUDIO>
5  <AUDIO src="mueller2.wav" bitrate="9600bps"></AUDIO>
6  <AUDIO src="mueller3.wav" bitrate="4800bps"></AUDIO>
7  <TEXT  src="mueller4.wav" bitrate="2400bps"></TEXT>
8  </SWITCH>
9  </PAR>
```

Abbildung 4.22: Beispiel für ein Switch-Element

Als Link-Ziele können sowohl SMIL-Dokumente als auch Schedule-Elemente innerhalb von SMIL-Dokumenten dienen. Ist ein Schedule-Element das Link-Ziel, so bewirkt dies die Präsentation des Dokumentes, in dem sich das SE befindet. Jedoch wird diese Präsentation bis zum SE ,,vorgespult". Abbildung 4.23 zeigt einige Beispiele für SMIL-Links.

```
1     <!-- Link, der aktuelle Pr"asentation ersetzt -->
2
3     <A href="http://xyz.org/irgendwas.smi" show="replace">
4     <IMG src="button1.gif"></IMG>
5     </A>
6
7     <!-- Link, der neue Pr"asentation zus"atzlich startet -->
8
9     <A href="http://xyz.org/nochwas.smi" show="new">
10    <IMG src="button2.gif"></IMG>
11    </A>
12
13    <!-- Link, der aktuelle Pr"asentation lediglich pausiert -->
14
15    <A href="http://xyz.org/sonstwas.smi" show="pause">
16    <IMG src="button3.gif"></IMG>
17    </A>
```

Abbildung 4.23: Beispiele für SMIL-Links

Es gibt außerdem eine weitere Link-Art, das `<ANCHOR>`-Element, die es erlaubt, die Interaktionsfläche eines Links sowohl räumlich, als auch zeitlich einzuschränken. Dieses erlaubt die Definition von Imagemaps.

### 4.4.2 Bewertung

Wie bei den vorangegangenen multimedialen Dokumentstandards wird in diesem Abschnitt SMIL auf die Erfüllung des in Kapitel 3 aufgestellten Anforderungskatalogs überprüft. Die Tabelle 4.3 auf Seite 43 zeigt eine Zusammenfassung der Bewertung.

#### Wiederverwendbarkeit

SMIL erlaubt die Wiederverwendung auf zwei Abstraktionsebenen. Zum einen kann man über das Tag `<REF>`, wie oben beschrieben, externe Medienobjekte über deren URL einbinden. Handelt es sich hierbei um kontinuierliche Medienobjekte, so ist außerdem der einzubindene Zeitausschnitt spezifizierbar. Zum anderen können SMIL-Dokumente als Ziel von Links angegeben werden, so daß auch die Wiederverwendbarkeit von vollständigen Dokumenten gegeben ist. Jedoch ist es nicht möglich, einzelne Bestandteile eines Dokumentes wiederzuverwenden. Man kann zwar Schedule-Elemente innerhalb eines Dokumentes als Link-Ziel angeben, jedoch hat dies lediglich eine Vorspul-Semantik und bewirkt damit ebenso die Präsentation des ganzen Dokumentes, wenn auch erst ab dem durch das Link-Ziel adressierten Schedule-Element. Somit wird die Wiederverwendbarkeit von Fragmenten nicht unterstützt.

SMIL trennt zwar die Layoutspezifikation im Document Head von der Spezifikation des Präsentationsablauf im Document Body, jedoch sind diese über die Referenzierung der Channels durch die Medienobjekte eng miteinander verzahnt. Somit ist nur identische Wiederverwendung möglich.

| | | |
|---|---|---|
| Wiederverwendbarkeit | Granularität | Wiederverwendbarkeit von Medienobjekten und vollständigen Dokumenten. |
| | Art | Identisch. |
| | Selektion | Dokumente können Meta-Attibute definieren. |
| Interaktion | | Navigierende Interaktion. |
| Adaption | | Clientbasiert. |
| Präsentationsneutralität | | Spezifikation von Dokumenten auf mittlerer semantischer Ebene. Eignung zur präsentationsneutralen Ablage von Dokumenten zwischen MHEG-5 und HyTime einzuordnen. |

Tabelle 4.3: Zusammenfassung der Bewertung von SMIL

Es bleibt noch zu untersuchen, inwieweit SMIL die Selektion von wiederverwendbaren Komponenten unterstützt. SMIL erlaubt die Definition von Meta-Attributen im Document-Head. Diese können zur Klassfizierung des Dokumentes verwendet werden, zum Beispiel über Facetten, was die softwaregestützte Selektion von SMIL-Dokumente erlaubt.

### Interaktion

SMIL erlaubt die Modellierung von navigierender Interaktion durch die Verwendung von Links. Jedoch gibt es keine Elemente, über die gestalterische Interaktionen spezifizierbar sind.

### Adaption

SMIL verfügt über Switch-Elemente, mit denen verschiedene Präsentationsvarianten eines Dokuments definierbar sind. Die Auswahl der passenden Variante wird durch von außen belegbare Parameter beeinflußt. SMIL-Dokument ist somit in der Lage, sich an ein von außen setzbares Profil zu adaptieren. Da die Auswahl der zu präsentierenden Variante zur Präsentationszeit in der Präsentationssoftware erfolgt, handelt es sich um clientbasierte Adaption.

### Präsentationneutralität

Im Gegensatz zu HyTime beschreibt SMIL weniger die Struktur eines Dokumentes, als vielmehr dessen Präsentation. Im Unterschied zu MHEG-5 geschieht dieses jedoch auf einer semantisch anderen Ebene. Während bei MHEG-5 auf dem niedrigen Niveau der Ereignis-Bedingung-Aktion-Regeln Präsentationsabläufe definiert werden, geschieht dies bei SMIL über Schedule-Elemente mit komplexerer Semantik, wie zum Beispiel die synchrone Paralleldarstellung von Medienobjekten. SMIL ist deswegen bei seiner Tauglichkeit zur präsentationsneutralen Ablage von Dokumenten zwischen HyTime und MHEG-5 anzusiedeln.

## 4.5 Fazit

Abschließend läßt sich feststellen, daß keiner der vorgestellten multimedialen Dokumentstandards die Forderungen an ein flexibles multimediales Dokumentmodell zufriedenstellend erfüllt. Der MHEG-5-Standard bietet zwar viel multimediale Funktionalität, was sich insbesonders an den angebotenen Interaktionsformen vor Augen führen läßt. Nachteilig ist jedoch das niedrige semantische Niveau der Beschreibungen des Dokumentinhalts und die stark eingeschränkte Wiederverwendbarkeit. HyTime hingegen zeigt bei der Wiederverwendbarkeit durch den Mechanismus der Locators seine Stärken. Auch ist das semantische Niveau einer Dokumentbeschreibung recht hoch, weswegen sich der Standard gut zur präsentationsneutralen Beschreibung von multimedialen Dokumenten eignet. Jedoch fehlt die Modellierbarkeit von Interaktion und Adaption. SMIL liegt bezüglich des semantischen Niveaus der Dokumentbeschreibungen zwischen MHEG-5 und HyTime. Auch erlaubt SMIL die Modellierung clientbasierter Adaption. Jedoch weist SMIL Mängel bei der Wiederverwendbarkeit und bei der Modellierung gestalterischer Interaktion auf.

# Kapitel 5

# Das ZYX-Dokumentmodell

Im vorigen Kapitel wurde festgestellt, daß keines der untersuchten Dokumentmodelle die an ein flexibles multimediales Dokumentmodell gestellten Anforderungen voll erfüllt. In diesem Kapitel soll deshalb ein multimediales Dokumentmodell namens ZYX entwickelt werden, welches diesen Anforderungen gerecht wird. Dazu werden zunächst die Überlegungen zum Entwurf des Dokumentmodells erläutert. Danach wird eine informelle Einführung in das ZYX-Dokumentmodell gegeben und dann dieses multimediale Dokumentmodell formal definiert. Schließlich wird ZYX gemäß den Forderungen aus Kapitel 3 bewertet.

## 5.1 Überlegungen zum Entwurf

Die zentralen Forderungen an ein flexibles multimediales Dokumentmodell sind bereits in Kapitel 3 ausführlich vorgestellt worden. Das Hauptziel bei der Definition des ZYX-Dokumentmodells ist natürlich die Erfüllung dieser Anforderungen. Deswegen werden diese im folgenden einzeln durchgegangen und zur jeder Forderung erläutert, inwiefern sie beim Entwurf von ZYX berücksichtigt wurde.

### 5.1.1 Präsentationsneutralität

Die Eignung eines Dokumentmodells zur präsentationsneutralen Ablage von multimedialen Dokumenten ist ein zentraler Punkt. Wie in Kapitel 3 ausführlich dargelegt, ist diese Eignung im wesentlichen durch die semantische Ebene des Dokumentmodells bestimmt. Eine hohe semantische Ebene ermöglicht eine einfache automatische Konvertierung in andere multimediale Dokumentmodelle, sofern diese auf einer semantisch niedrigeren Ebene liegen. Die semantische Ebene ist somit früh im Entwurfsprozeß zu berücksichtigen.

Orientiert man sich am ereignisbasierten Modell der Spezifikation von multimedialen Dokumenten von MHEG-5, so stellt man fest, daß dieses viel zu niedrig ist, um die präsentationsneutrale Ablage von Dokumenten zu gewährleisten, obwohl ein solches Modell sich wegen seiner Nähe zu Programmiersprachen sehr gut zur Realisierung von viel multimedialer Funktionalität eignet. Betrachtet man hingegen das multimediale Dokumentmodell HyTime, so entdeckt man, daß das hohe semantische Niveau dieses Standards zu Lasten der multimedialen Funktionalität geht, was sich

im Fehlen jeglicher Beschreibungsmöglichkeiten für Interaktion bemerkbar macht.

SMIL stellt in dieser Hinsicht einen recht guten Kompromiß bezüglich semantischer Ebene und multimedialer Funktionalität dar. Deshalb orientiert sich ZYX an SMIL in der Weise, daß von SMIL die hierarchische Spezifikation eines multimedialen Dokuments übernommen wird. Jedoch geht ZYX in den Bereichen Wiederverwendbarkeit, Adaption und Interaktion weit über SMIL hinaus.

### 5.1.2 Wiederverwendbarkeit

Das ZYX-Dokumentmodell soll die Wiederverwendbarkeit von multimedialen Dokumenten bzw. von deren Bestandteile auf allen in Kapitel 3 vorgestellten Ebenen ermöglichen. Deswegen werden im Gegensatz zu SMIL Abstraktionsmechanismen angeboten. Man kann in ZYX Teile einer hierarchischen Spezifikation kapseln und in anderen Spezifikationen wiederverwenden, und zwar gleichwertig mit den von ZYX angebotenen Basiskonstrukten. Diese wiederverwendbaren Teile einer Spezifikation können sowohl einzelne Medienobjekte, Fragmente einer Dokumentspezifikation als auch die Spezifikation eines ganzen Dokuments sein. Außerdem ist es möglich, Schablonen von Fragmenten zu spezifizieren, die bei Benutzung an die Bedürfnisse eines Autors angepaßt werden können.

Bei der Art der Wiederverwendbarkeit wird bei ZYX ein pragmatischer Weg gegangen. Einerseits werden die Vorteile semantischer Wiederverwendbarkeit erkannt und deswegen versucht, eine möglichst starke Trennung von Struktur und räumlichen und gestalterischen Beziehungen angestrebt. Andererseits sollen Autoren multimedialer Dokumente dennoch die Möglichkeit haben, bei Bedarf die gestalterischen und räumlichen Beziehungen zwischen den Elementen eines Dokuments festzulegen.

Zur softwaregestützten Selektion wird der facettenorientierte Ansatz verfolgt.

### 5.1.3 Interaktion

Es sollen sowohl Navigationsinteraktionen als auch Gestaltungsinteraktionen mit dem ZYX-Dokumentmodell beschreibbar sein. Hierbei soll die Mächtigkeit der Modellierungsmittel von SMIL deutlich übertroffen und ungefähr die Mächtigkeit von MHEG-5 erreicht werden.

### 5.1.4 Adaption

In Kapitel 3 wurden zwei Adaptionsarten unterschieden, die clientbasierte Adaption und die serverbasierte Adaption. Man muß anerkennen, daß beide Adaptionsarten ihre prädestinierten Anwendungsgebiete haben. Wenn beispielsweise ein Autor eines multimedialen Dokuments anhand der zur Verfügung stehende Bandbreite des Netzwerks zwischen einer Zeichnung und einem Video entscheiden möchte, ist die clientbasierte Adaption hierfür ein einfaches Beschreibungsmittel. Kennt der Autor hingegen nicht alle Alternativen (beispielsweise möchte er irgendein multimediales Dokument über die Herzchirugie auswählen), so ist die serverbasierte Adaption eine geeignete Beschreibungsform.

ZYX unterstützt somit beide Adaptionsarten, was es deutlich von SMIL abhebt.

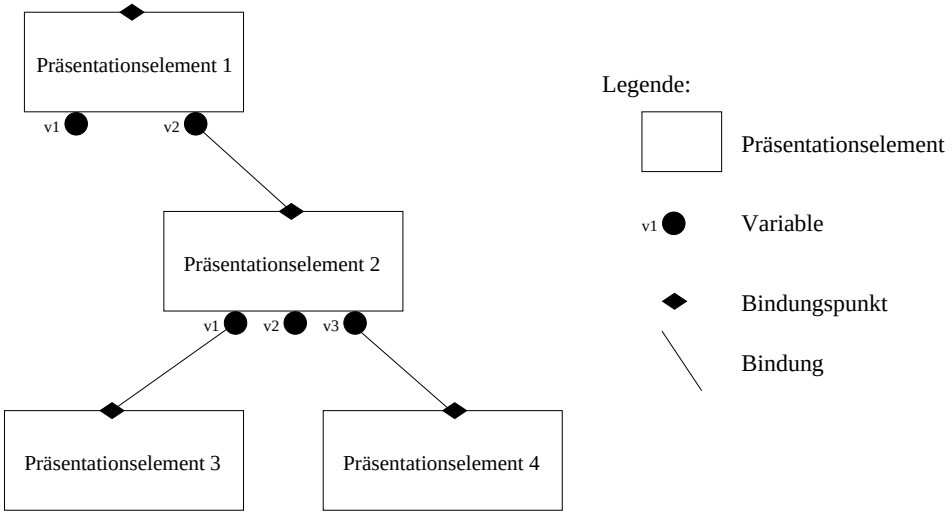


Abbildung 5.1: Eine hierarchische Spezifikation

## 5.2 Informelle Einführung in ZYX

Die grundlegende Idee des ZYX-Dokumentmodells ist die hierarchische Spezifikation eines multimedialen Dokumentes mit Hilfe eines Baumes (siehe Bild 5.1). Die Knoten des Baumes repräsentieren dabei die sogenannten *Präsentationselemente*, die Kanten des Baumes verbinden die Präsentationselemente untereinander. Sie werden deswegen *Bindungen* genannt. Jedem Präsentationselement ist ein *Bindungspunkt* zugeordnet, über den es an ein anderes Präsentationselement gebunden werden kann. Desweiteren können Präsentationselemente über *Variablen* verfügen. Variablen eines Präsentationselements werden über Bindungen mit Bindungspunkten verbunden. Ist eine Variable eines Präsentationselementes $p_1$ an den Bindungspunkt eines anderen Präsentationselementes $p_2$ gebunden, bedeutet dies, daß $p_1$ die Präsentation von $p_2$ in einer gewissen Art und Weise beeinflußt. Das Wurzel-Präsentationselement eines Baumes bildet so den Einstiegspunkt der Spezifikation.

Ein Präsentationselement ist entweder ein *Operatorelement* oder ein *Fragment*. Ein Fragment kapselt wiederverwendbare Dokumentbestandteile, deren Granularität von einem Medienobjekt, wie z. B. ein Video, über eine logisch zusammenhängende Gruppe von Dokumentbestandteilen bis hin zu einem kompletten multimedialen Dokument reichen kann. Operatorelemente dienen dazu, Beziehungen zwischen den Präsenationselementen eines Dokumentes zu beschreiben. Hierzu betrachte man Abbildung 5.2, die eine Slide-Show darstellt. Das Präsentationselement *seq* stellt ein Operatorelement dar. Es spezifiziert, daß die an seine Variablen $v1, \ldots, v5$ gebunden Präsenationselemente in sequentieller Abfolge präsentiert werden sollen. Die Präsentationselemente $Folie\ 1, \ldots, Folie\ 3$ sind Fragmente. Sie kapseln wiederverwendbare Dokumentbestandteile. In der Abbildung ist nicht näher angegeben, was genau diese Fragmente kapseln. Es kann sich um Medienobjekte wie zum Beispiel Images handeln. In diesem Fall würden die Fragmente $Folie\ 1, \ldots, Folie\ 3$ als *atomare Medienobjekte* bezeichnet. Jedoch können die Fragmente $Folie\ 1, \ldots, Folie\ 3$ auch weitere hierarchische Spezifikationen im ZYX-Dokumentmodell kapseln. In diesem Fall würden sie als *komplexe Medienobjekte* bezeichnet.

Ein interessanter Aspekt des ZYX-Dokumentmodells ist, daß es erlaubt, in einer Spezifikation Variablen zum Erstellungszeitpunkt nicht zu binden. Stattdessen ist es

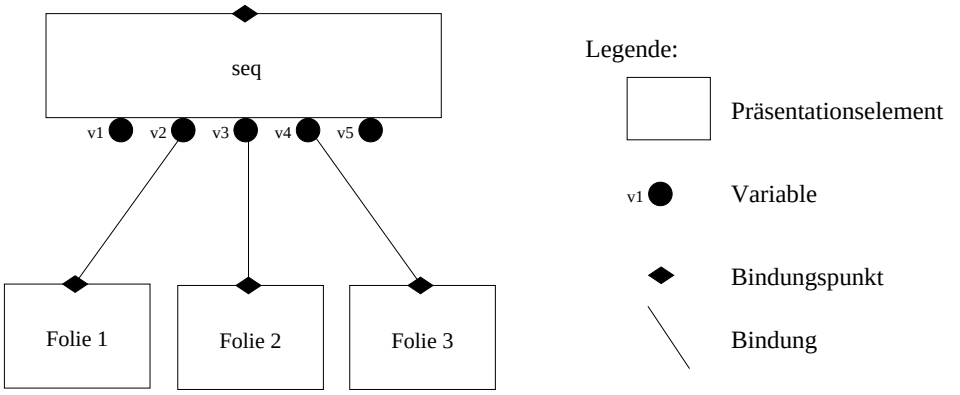


Abbildung 5.2: Beispiel einer Dokumentschablone

möglich, dies zu einem späteren Zeitpunkt nachzuholen. Dieses schafft die Möglichkeit von Dokumentschablonen, deren nicht-gebundene Variablen erst später gebunden und somit an einen konkreten Anwendungsfall angepaßt werden.

Als Beispiel hierfür soll erneut die Spezifikation der Slide-Show in Abbildung 5.2 dienen. Der Autor der Slide-Show hat auf die Bindungen der Variablen $v1$ und $v5$ des *seq*-Operatorelements (Titel und Abschlußfolie) verzichtet. Er läßt sich so die Möglichkeit offen, dies zu einem späteren Zeitpunkt nachzuholen und konkret eine Titel und Abschlußfolie anzugeben. Dies kann zum Beispiel Sinn machen, wenn derselbe Vortrag auf verschiedenen Konferenzen gehalten werden soll.

Das oben vorgestellte *seq*-Operatorelement dient zur Beschreibung von zeitlichen Beziehungen zwischen Präsentationselementen. Es gehört deswegen zur Gruppe der *temporalen Operatorelemente*. Es gibt jedoch auch Operatorelemente zur Beschreibung von räumlichen bzw. gestalterischen Beziehungen zwischen Präsenationselementen, die sogenannten *Projektoren*. Es gibt beispielsweise einen räumlichen Projektor, genannt $spatial_p$, der den Bildschirmbereich festlegt, in dem Präsentationselemente bei der Präsentation dargestellt werden. Es gibt auch einen $acoustic_p$-Projektor, der die Lautstärke festlegt, mit der Medien mit akustischem Anteil präsentiert werden.

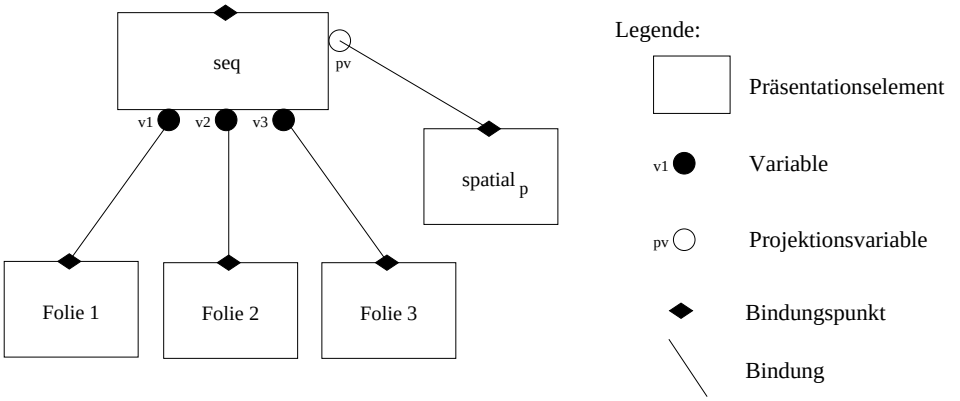


Abbildung 5.3: Beispiel Projektoren

Projektoren haben die Besonderheit, daß sie nur an eine spezielle Sorte von Variablen gebunden werden dürfen, die *Projektionsvariablen*. Dieses dient zur klaren

Trennung von Layout und Struktur eines ZYX-Dokumentes. Jedes Präsentationselement kann beliebig viele Projektionsvariablen definieren. Abbildung 5.3 soll dies illustrieren. An ein *seq*-Operatorelement ist ein $spatial_p$-Projektor über eine Projektionsvariable *pv* gebunden. Das bedeutet, daß dieser Projektor die räumliche Position festlegt, in der nicht nur das *seq*-Operatorelement selbst, sondern auch die an das *seq*-Operatorelement gebundenen Präsentationselemente zum Präsentationszeitpunkt dargestellt werden. Da ein *seq*-Operatorelement über keine räumliche Komponente verfügt (es ist nur ein temporales Operatorelement), wirkt in diesem Fall der $spatial_p$-Projektor nur auf die Präsentionselemente $Folie\ 1, \ldots, Folie\ 3$. Die Semantik der ganzen Spezifikation ist, daß die Präsentationselemente $Folie\ 1$, $\ldots, Folie\ 3$ nacheinander im gleichen rechteckigen Auschnitt der Präsentationsfläche dargestellt werden sollen.

Es gibt noch ein große Zahl weiterer Projektoren und Operatorelemente, die beispielsweise zur Modellierung von Interaktion und Adaption dienen. Sie werden in der formalen Definition des ZYX-Dokumentmodells näher vorgestellt.

## 5.3 Formale Definition von ZYX

Nach der obigen kurzen Einführung wird im folgenden das ZYX-Dokumentmodell formal definiert. Dabei werden zunächst die Grundlagen von ZYX definiert. Danach werden die einzelnen Operatorelemente vorgestellt, und zwar unterteilt in die Kategorien *temporale Operatorelmente*, *gestalterische und räumliche Operatorelemente*, *Interaktionsoperatorelemente* und *Adaptionsoperatorelemente*.

### 5.3.1 Grundlegende Definitionen

**Definition 1 (Präsentationselement, Fragment, Operatorelement).**
Die *Präsentationselemente* sind die grundlegenden Bestandteile des ZYX-Dokumentmodells . Jedem Präsentationselement $e$ ist neben einem Bindungspunkt $bp_e$ eine Menge $V_e$ von *Variablen* und eine Menge $PV_e$ von *Projektionsvariablen* zugeordnet.

□

Die Menge der Projektionsvariablen eines Präsentationselementes ist in der Regel leer. Erst zum Erstellungszeitpunkt einer Spezifikation kann ein Autor Projektionsvariablen definieren und sie der Menge $PV_e$ eines Präsentationselementes $e$ hinzufügen. Deswegen wird in den nachstehenden Definitionen von Präsentationselementen auf die explizite Angabe $PV_e = \emptyset$ verzichtet.

**Definition 2 (Spezifikationsbaum).**
Ein Spezifikationsbaum $S = (E, B)$ ist ein azyklischer, ungerichteter Graph, der zur Beschreibung von ganzen oder von Teilen eines multimedialen Dokumentes dient, wobei:

- $E$ die Menge der im Spezifikationsbaum auftretenden Präsentationselemente,
- $B$ die Menge der Bindungen von Variablen der Präsentationselemente aus $E$ an Bindungspunkte anderer Elemente aus $E$. Eine Bindung $b \in B$ ist definiert als: $b = (v, bp_{e'})$, mit $v \in \bigcup_{e \in E} (V_e \cup PV_e), e' \in E$.

Für $S = (E, B)$ müssen folgende Randbedingungen eingehalten werden:

1. Seien $b_1, b_2 \in B$, $b_1 = (v_1, bp_e), b_2 = (v_2, bp_e), e \in E$.
   Dann muß gelten: $v_1 = v_2$. Das bedeutet, daß in $S$ ein Bindungspunkt nur an eine Variable gebunden sein darf.
2. $|\{e \in E | \forall v \in \bigcup_{e' \in E} (V_{e'} \cup PV_{e'}) : (v, bp_e) \notin B\}| = 1$
   Hiermit wird ausgesagt, daß es genau eine Wurzel in $S$ geben muß. Eine Wurzel ist ein Präsentationselement, welches in $S$ an keine Variable gebunden ist. Die Wurzel bildet den Einstiegspunkt der Spezifikation.
3. Es gibt keine Sequenz von Bindungen $b_1, \ldots, b_n$, mit $b_i = (v_i, bp_{e_i}), i = 1 \ldots n$, so daß gilt: $v_{i+1} \in (V_{e_i} \cup PV_{e_i})$ und $v_1 \in (V_{e_n} \cup PV_{e_n})$. Dies meint, daß in $S$ keine Zyklen existieren dürfen.
4. $\forall (v, bp_e) \in B : v \in PV_{e'} \Longleftrightarrow e$ ist Projektor.
   Damit wird ausgesagt, daß an Projektionsvariablen nur *Projektoren* gebunden werden dürfen. Projektoren sind spezielle Präsentationselemente, die weiter unten definiert werden.

□

Nachdem jetzt die Möglichkeit besteht, Präsentationselemente zu definieren und in Spezifikationsbäumen miteinander zu verbinden, sollen nun die einzelnen Arten von Präsentationselementen vorgestellt werden.

**Definition 3 (Fragment).**
Ein *Fragment* ist ein Präsentationselement, das einen wiederverwendbaren Bestandteil eines multimedialen Dokuments kapselt.

□

Fragmente sollen mittels facettenorientierter Klassifikation klassifizierbar sein. Dieses erlaubt die softwaregestützte Selektion von Fragmenten.

**Definition 4 (Metainformation).**
Eine Metainformation ist ein Tupel $(k, d)$, wobei $k$ den Schlüssel (also die Bezeichnung) für eine Metainformation und $d$ den Wert einer Metainformation darstellen.

Jedem Fragment $f$ ist eine Menge von Tupeln $M_f \in \mathcal{P}(dom(k) \times dom(d))$ an Metainformationen zugeordnet, wobei $dom(k)$ und $dom(d)$ den jeweiligen Wertebereich vom Schlüssel und vom Wert einer Metainformation repräsentieren.

□

Metainformationen beschreiben den Inhalt eines Fragments. Nützlich sind Metainformation aber auch zur Beschreibung technischer Eigenschaften eines Fragments, wie zum Beispiel die benötigte Netzwerkbandbreite oder Rechnerleistung.

**Definition 5 (Atomares Medienobjekt).**
Ein *atomares Medienobjekt a* ist ein Fragment. Es kapselt ein Medienobjekt eines Basismedientyps $Type_a, Type_a \in \{Video, Audio, Image, Text\}$, innerhalb von Spezifikationsbäumen.. Das Objekt ist im Format $Format_a, Format_a \in \{MPEG, JPEG, WAV, \ldots\}$ abgelegt.

Es gilt immer folgende Bedingung: $V_a = \emptyset$.

□

Mit Hilfe von atomaren Medienobjekten ist das ZYX-Dokumentmodell in der Lage die Daten einzelner Medienobjekte zu kapseln. Somit ist die Wiederverwendbarkeit auf der Ebene der Medienobjekte gewährleistet. Wie aber schon erwähnt, soll es auch möglich sein, Fragmente von multimedialen Dokumenten und ganze multimediale Dokumente wiederzuverwenden. Hierzu dienen die komplexen Medienobjekte.

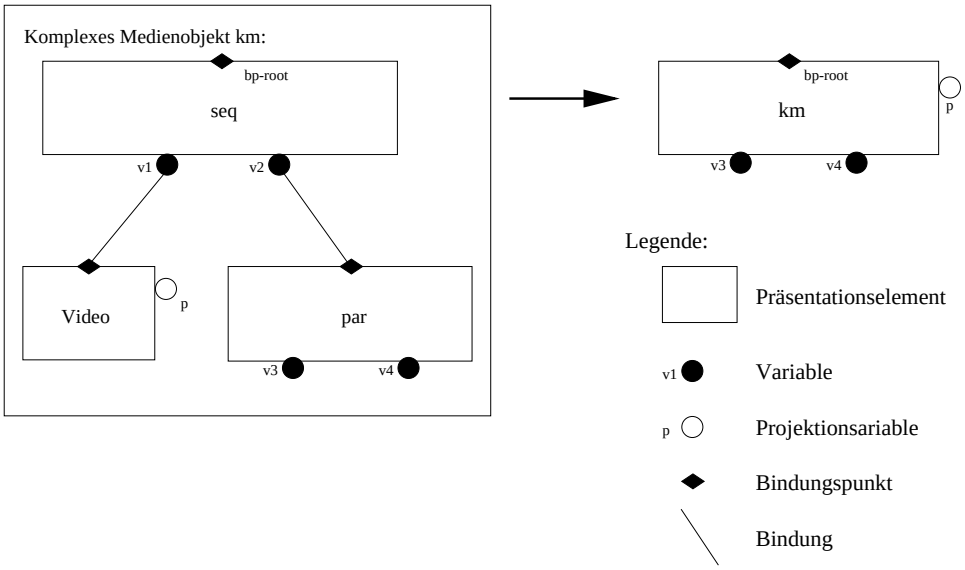


Abbildung 5.4: „Export" von Variablen und Bindungspunkten durch ein komplexes Medienobjekt

**Definition 6 (Komplexes Medienobjekt).**
Ein *komplexes Medienobjekt* $km$ ist ein Fragment, welches einen Spezifikationsbaum $S = (E, B)$ kapselt. $km$ kann als Platzhalter von $S$ in anderen Spezifikationsbäumen wiederverwendet werden. Dabei gelten folgende Bedingungen:

1. $\forall v \in \bigcup_{e \in E} V_e$ :
   $v \in V_{km} \Leftrightarrow \forall e \in E : (v, bp_e) \notin B$
   Dieses bedeutet, daß die in $S$ nicht gebundenen Variablen der gekapselten Präsentationselemente die Menge der Variablen von $km$ bilden. Diese ungebundenen Variablen der gekapselten Elemente werden sozusagen von $km$ „nach außen exportiert" (siehe Abbildung 5.4).

2. $\forall v \in \bigcup_{e \in E} PV_e$ :
   $v \in PV_{km} \Leftrightarrow \forall e \in E : (v, bp_e) \notin B$
   Dieses bedeutet, daß die in $S$ nicht gebundenen Projektionsvariablen der gekapselten Präsentationselemente die Menge der Projektionsvariablen von $km$ bilden (siehe Abbildung 5.4).

3. $bp_{km} = bp_{Root(S)}$. Der Bindungspunkt der Wurzel $Root(S)$ von $S$ ist auch der Bindungspunkt von $km$ (siehe Abbildung 5.4).

□

Weil komplexe Medienobjekte Spezifikationen von Dokumentbestandteilen kapseln, stellen sie somit ein Abstraktionsmittel zur Verfügung. Präsentationselemente kön-

nen beliebig kompliziert zu neuen Präsentationselementen gruppiert werden, welche wiederum in anderen Spezifikationsbäumen wiederverwendbar sind. Somit ist die Wiederverwendbarkeit auf allen in Kapitel 3 identifizierten Abstraktionsebenen möglich.

Da ungebundene Variablen und Projektionsvariablen von komplexen Medienobjekten exportiert werden, ist es möglich, Schablonen in komplexen Medienobjekten zu kapseln. Diese Schablonen können später in anderen Spezifikationsbäumen an die konkreten Bedürfnisse eines Autors angepaßt werden.

Während komplexe Medienobjekte es den Autoren erlauben, ihre multimedialen Dokumente logisch zu strukturieren, ist die sich daraus ergebende Verschachtelung von Spezifikationsbäumen für die nachfolgenden Definitionen unbequem zu handhaben. Deswegen wird ein Hilfsmittel definiert.

**Definition 7 ($flatten$).**
Sei $S$ ein Spezifikationsbaum. Dann ist $flatten(S)$ der zu $S$ semantisch äquivalente Spezifikationsbaum, in dem keine komplexen Medienobjekt mehr vorkommen. Dazu ist rekursiv das Vorkommen von komplexen Medienobjekten in $S$ durch die im jeweiligen komplexen Medienobjekt gekapselte Spezifikation zu ersetzen.

□

Wie in Kapitel 3 im Abschnitt über Präsentationsneutralität beleuchtet, kann es problematisch sein, vollautomatisch zwischen verschiedenen Dokumentmodellen zu konvertieren. Es ist nun unrealistisch anzunehmen, daß Dokumente nur im ZYX-Dokumentmodell spezifiziert werden. Es muß stattdessen davon ausgegangen werden, daß Dokumente anderer multimedialer Dokumentmodelle vorliegen, die nach ZYX konvertiert werden sollen. Um das Überführen solcher Dokumente nach ZYX zu erleichtern und das Problem der Konvertierbarkeit zu umgehen, wird eine weitere Fragmentart definiert, die *externen Medienobjekte*. Anstelle Bestandteile von Dokumenten anderer Dokumentmodelle nach ZYX zu konvertieren, werden diese in ihrem ursprünglichen Modell belassen und lediglich in externen Medienobjekten gekapselt. Externe Medienobjekte können dann ähnlich zu komplexen Medienobjekten in Spezifikationsbäumen verwendet werden.

**Definition 8 (Externes Medienobjekt).**
Ein *externes Medienobjekt* $e$ ist ein Fragment. Es kapselt die Spezifikation eines ganzen Dokuments oder eines Dokumentfragments in einem fremden Dokumentmodell $Format_e, Format_e \in \{SMIL, HyTime, MHEG-5, \ldots\}$. Einem externen Medienobjekt ist eine Menge von Variablen $V_e$ zugeordnet. Was genau gekapselt wird und die Semantik einer Bindung eines Präsentationselements an eine Variablen $v \in V_e$ hängt vom fremden Dokumentmodell ab.

□

Im Rahmen dieser Arbeit wird lediglich die Kapselung von Dokumenten behandelt, die in SMIL spezifiziert wurden (siehe Abbildung 5.5). Ein externes Medienobjekt kapselt dabei ein vollständiges SMIL-Dokument, inklusive Document Head und Document Body. Die Verwendung eines externen Medienobjektes $e$ mit gekapselten SMIL-Dokument $s$ in einem Spezifikationsbaum bewirkt zum Präsentationszeitpunkt die Präsentation von $s$. Um $s$ flexibler wiederverwenden zu können, ist es möglich, Link-Ziele in $s$ an die Variablen $V_e$ des externen Medienobjektes zu koppeln, wie in Abbildung 5.5 illustriert. An diese Variablen dürfen nur komplexe Medienobjekte gebunden werden, da die Semantik von SMIL-Links navigierender Interaktion zwischen Dokumenten entspricht. Die genaue Bedeutung der Bindung eines komplexen Medienobjektes an eine Variable von $e$ ist dabei folgende: Aktiviert

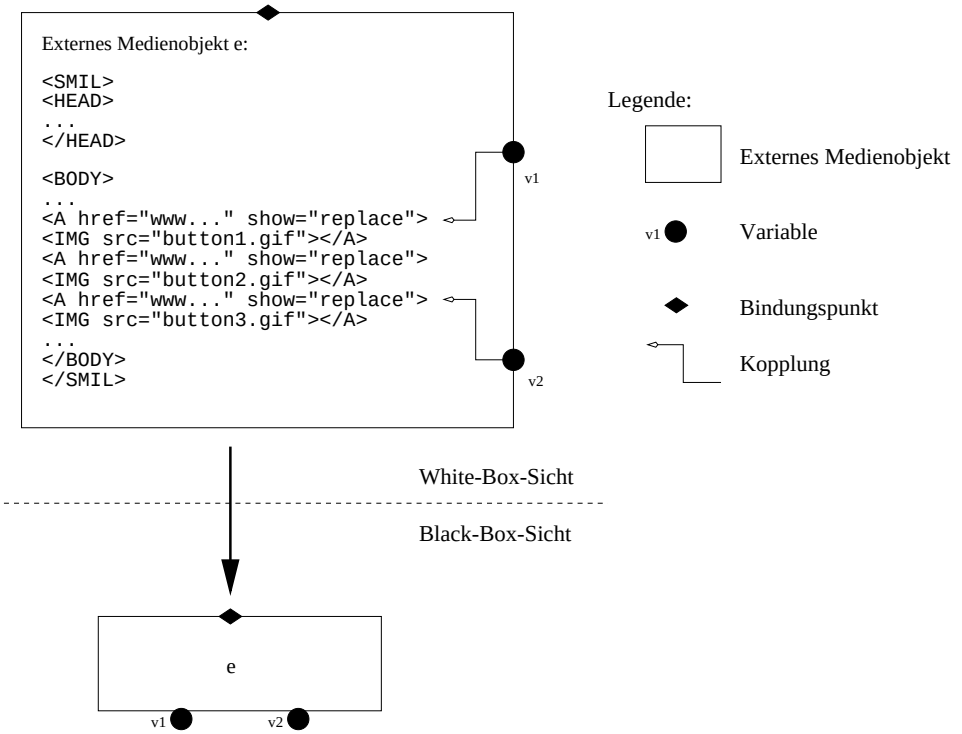


Abbildung 5.5: Kapselung von SMIL-Dokumenten in externen Medienobjekten

der Benutzer während der Präsentation des in $e$ gekapselten SMIL-Dokumentes $s$ einen Link, der an eine Variable $v \in V_e$ gekoppelt ist, so wird nicht das im SMIL-Dokument spezifizierte Link-Ziel aufgerufen. Stattdessen wird die Präsentation des an $v$ gebundenen komplexen Medienobjektes eingeleitet. Die Präsentation des aktuellen Dokumentes wird bei Aktivierung eines SMIL-Links beendet.

Mit den bisherigen Definitionen ist es möglich, Präsentationselemente über Variablenbindungen zueinander in Beziehung zu setzen und sie in komplexen Medienobjekten zu neuen, komplexeren Präsentationsfragmenten zusammenzustellen. Während Variablenbindungen zwar festlegen, daß Präsentationselemente zueinander in Beziehung stehen, so ist die Semantik dieser Beziehungen nicht definiert. Dazu dienen *Operatorelemente*, die Beziehungen zwischen Präsentationselementen festlegen.

**Definition 9 (Operatorelement).**
Ein *Operatorelement* $o(v_1, \ldots, v_n, p_1, \ldots, p_m)$ ist ein Präsentationselement, welches andere Präsenationselemente zueinander mit einer gewissen Semantik in Beziehung setzt. Dabei sind

- $V_o = \{v_1, \ldots, v_n\}$ die Menge der Variablen von $o$ und
- $p_1, \ldots, p_m$ vom Operatorelement abhängige Parameter.

□

Nachdem nun das Gerüst des ZYX-Dokumentmodells vorhanden ist, müssen in den nächsten Abschnitten die verschiedenen Operatorelemente des Modells definiert werden.

### 5.3.2 Temporale Operatorelemente

Temporale Operatorelemente dienen dazu, zeitliche Beziehungen zwischen Präsentationselementen festzulegen. Das ZYX-Dokumentmodell orientiert sich dabei im wesentlichen an den Operatoren des temporalen Modells der Interval Expressions [DK95, KD96], wobei jedoch verschiedene Varianten von Operatoren zu mächtigeren, parametrisierbaren ZYX-Operatorelementen zusammengefaßt werden. Das Modell der Interval Expressions hat den Vorteil, daß es in der Lage ist, zeitliche Beziehungen zwischen Elementen mit unbekannter Präsentationsdauer zu beschreiben. Dies ist für ein multimediales Dokumentmodell wichtig, in dem Interaktion modellierbar sein soll, weil das Verhalten des Betrachters nicht zum Spezifikationszeitpunkt eines Dokuments vorhergesehen werden kann.

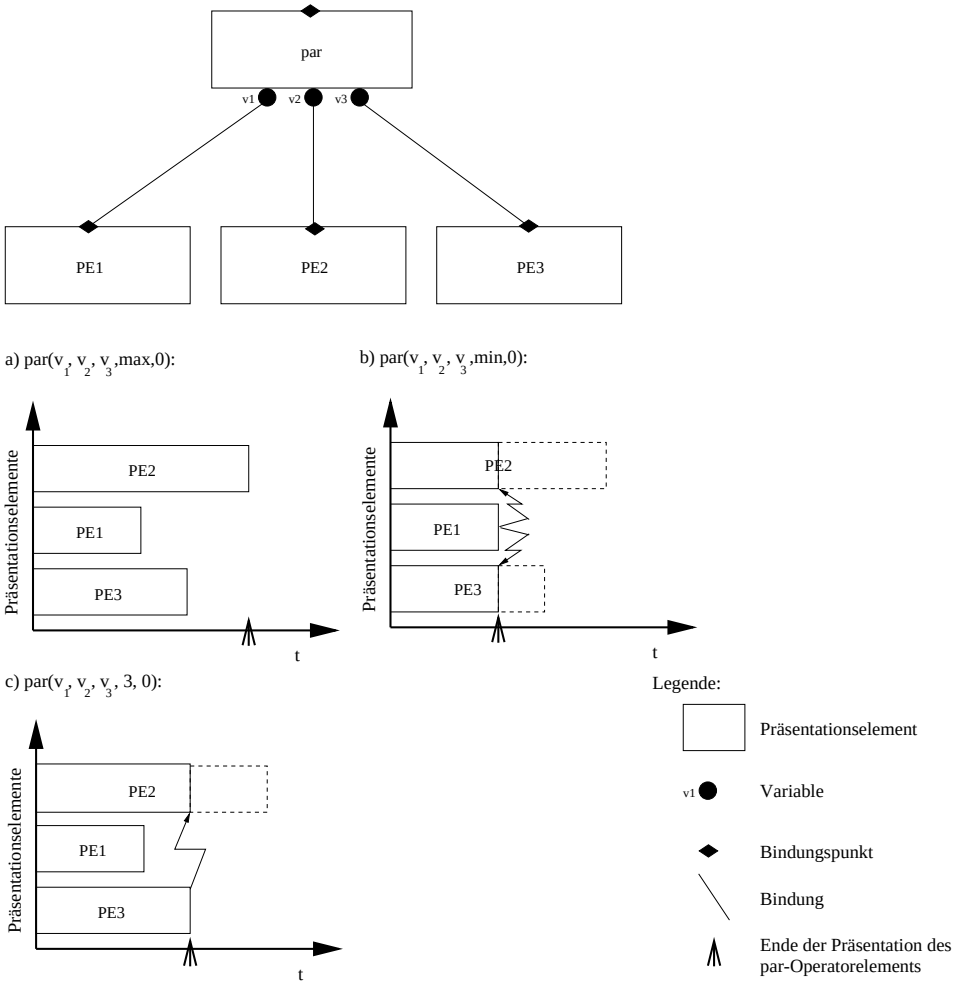


Abbildung 5.6: Verschiedene Varianten des *par*-Operatorelements

**Definition 10 (*par*-Operatorelement).**
Der Beginn der Präsentation des Operatorelements $par(v_1, \ldots, v_n, finish, lipsync)$ startet die parallele Präsentation der mit $v_i, i = 1 \ldots n$ verbundenen Präsentationselemente[1], wobei:

[1] Im folgenden werden diese Elemente vereinfachend direkt mit den Variablen, an die sie gebunden sind, bezeichnet, obwohl Variablen selbst natürlich keine Präsentationselemente darstellen.

- $V_{par} = \{v_1, \ldots, v_n\}$
- $finish \in \{1, \ldots, n, min, max\}$
- $lipsync \in \mathcal{N}_0$

Dabei sind drei Fälle bzgl. $finish$ zu unterscheiden:

1. $finish \in \{1, \ldots, n\}$: Sobald die Präsentation des Elementes $v_{finish}$ beendet ist, wird die Darstellung des *par*-Operatorelements und damit auch der Elemente $v_i, i \neq finish$ beendet (siehe Abbildung 5.6c).
2. $finish = min$: Sobald die Präsentation eines Elementes $v_{min}$ endet, wird die Darstellung des *par*-Operatorelements beendet (siehe Abbildung 5.6b).
3. $finish = max$: Erst wenn alle Präsentationselemente $v_i, i = 1 \ldots n$ ihre Präsentation beendet haben, endet die Darstellung des *par*-Operatorelements (siehe Abbildung 5.6a).

$lipsync \neq 0$ bedeutet, daß die Präsentation von $v_1, \ldots, v_n$ lippensynchron mit $v_{lipsync}$ als Master erfolgt.

□

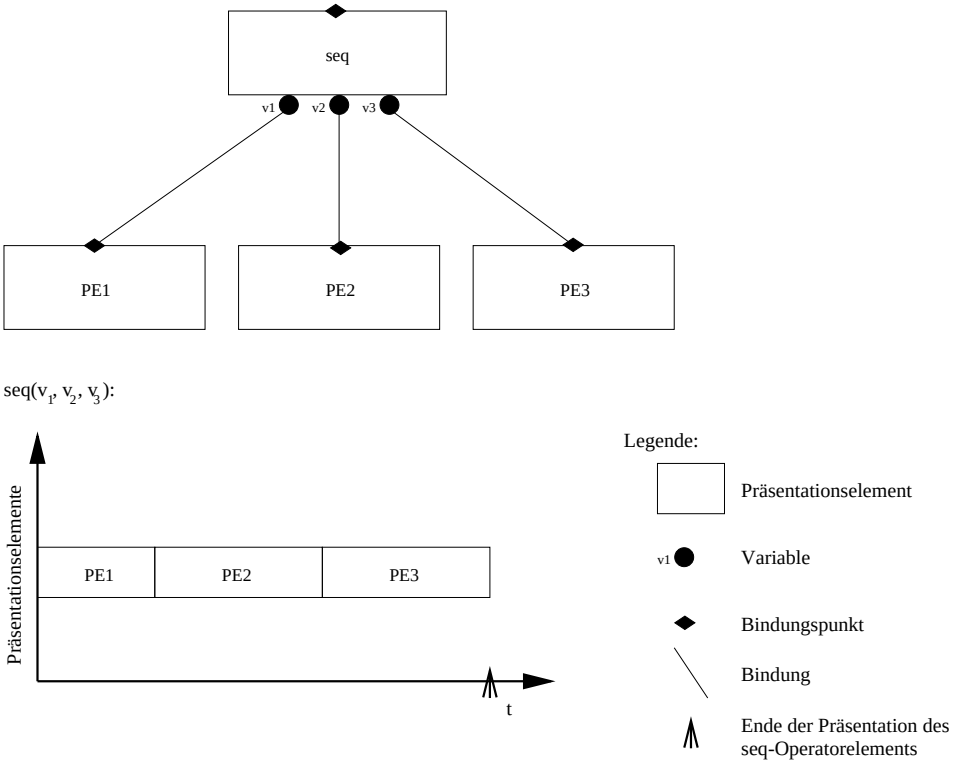


Abbildung 5.7: Das *seq*-Operatorelement

**Definition 11 (*seq*-Operatorelement).**
Der Beginn der Präsentation des Operatorelements $seq(v_1, \ldots, v_n)$, $\{v_1, \ldots, v_n\} = V_{seq}$, startet die Präsentation des Elementes $v_1$. Die Beendingung der Präsentation eines Elementes $v_i, i = 1 \ldots n-1$ startet die Präsentation des Elementes $v_{i+1}$. Das

Ende der Darstellung des letzten Elementes $v_n$ beendet die Präsentation des *seq*-Operatorelements (siehe Abbildung 5.7).

□

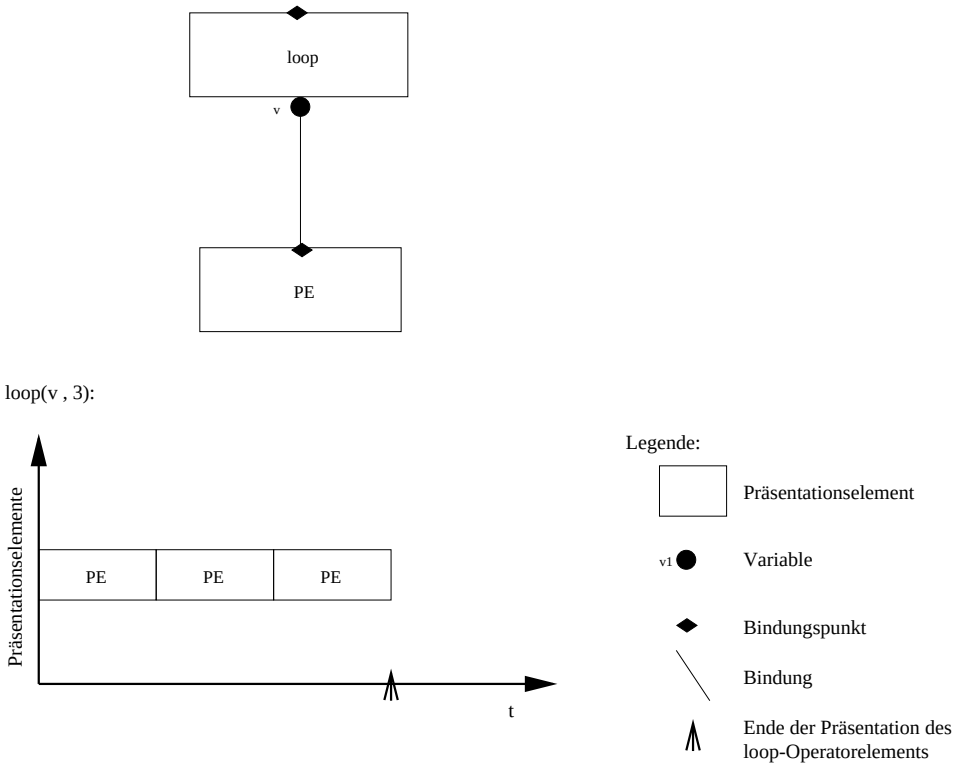


Abbildung 5.8: Das *loop*-Operatorelement

**Definition 12 (*loop*-Operatorelement).**
Der Beginn der Präsentation des Operatorelements $loop(v, n)$, $n \in \mathcal{N}_0$ und $\{v\} = V_{loop}$, startet die wiederholte Darstellung des Präsentationselementes $v$. Ist $n \neq 0$, so wird $v$ $n$-mal wiederholt. Nach der letzten Wiederholung wird der *loop*-Operatorelements beendet. Ist $n = 0$, so wird $v$ unendlich oft wiederholt (siehe Abbildung 5.8).

□

**Definition 13 (*delay*-Operatorelement).**
Das Operatorelement $delay(t)$, $t \in \Re_0^+$ und $V_{delay} = \emptyset$, modelliert eine Verzögerung von $t$ msec (siehe Abbildung 5.9).

□

### 5.3.3 Gestalterische und räumliche Operatorelemente

Während im vorigen Abschnitt Operatorelemente definiert wurden, welche die zeitlichen Beziehungen zwischen Präsentationselementen eines multimedialen Dokuments modellieren, werden in diesem Abschnitt Operatorelemente für räumliche und gestalterische Beziehungen vorgestellt. Diese Operatorelemente werden durch die nachstehende Definition in zwei Kategorien eingeteilt.

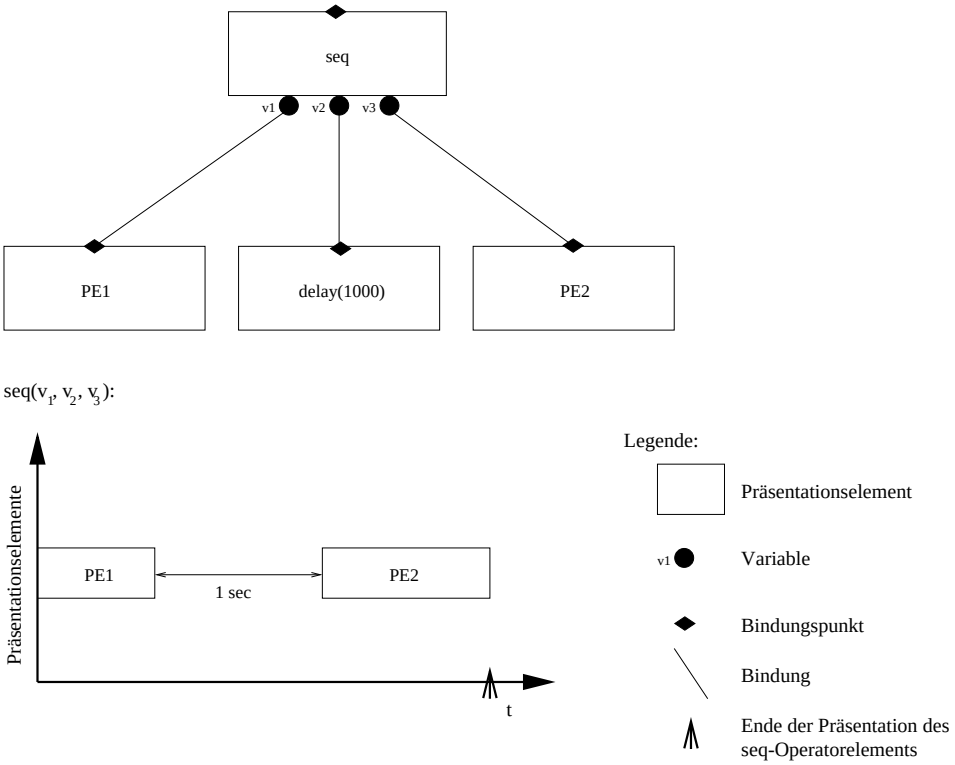


Abbildung 5.9: Das *delay*-Operatorelement

**Definition 14 (Projektoren, Selektoren).**
Ein *Projektor* ist ein spezielles Operatorelement, das eine räumliche oder gestalterische Beziehung festlegt. Projektoren sind die einzigen Präsentationselemente, die an Projektionsvariablen gebunden werden dürfen. Für einen Projektor $p$ muß immer gelten: $V_p = \emptyset \wedge PV_p = \emptyset$. Projektoren werden durch den Index $p$ im Operatorelementnamen kenntlich gemacht.

Ein *Selektor* ist ein spezielles Operatorelement, das einen räumlichen, zeitlichen oder gestalterischen Ausschnitt eines Fragments zur Präsentation auswählt. Selektoren werden durch den Index $s$ im Operatorelementnamen markiert.

□

Der Unterschied zwischen Projektoren und Selektoren ist, daß ein Projektor bestimmt, *wie* etwas präsentiert wird (z.B. die Position und Abspielgeschwindigkeit eines Videos), während ein Selektor festlegt, *was* präsentiert wird (z.B. einen zeitlichen Ausschnitt aus einem Video). Im weiteren Verlauf erweist sich nachstehende Definition als nützlich:

**Definition 15 (Nachfolger).**
Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$ und $e$ ein Präsentationselement aus $E$.

$e'$ ist *direkter Nachfolger* von $e$, gdw. $\exists (v, bp_{e'}) \in B : v \in V_e$.

$e'$ ist *indirekter Nachfolger* von $e$, gdw. $e'$ kein direkter Nachfolger von $e$ ist und es eine Folge $(succ_1, \ldots, succ_n), n \in \mathcal{N}$ gibt, mit $succ_1$ ist direkter Nachfolger von $e$, $succ_i, i = 2, \ldots, n$ ist direkter Nachfolger von $succ_{i-1}$ und $e'$ ist direkter Nachfolger

von $succ_n$.

$e'$ ist *Nachfolger* von $e$, gdw. $e'$ direkter oder indirekter Nachfolger von $e$ ist.

□

Abbildung 5.10 stellt einen Spezifikationsbaum dar. In diesem sind Präsentationselement 5 und Präsentationselement 2 direkte (nicht indirekte) Nachfolger von Präsentationselement 1. Präsentationselemente 3 und 4 sind indirekte Nachfolger von Präsentationselement 1 und direkte Nachfolger von Präsentationselement 2. Die Präsentationselemente 5 und 2 stehen in keiner Nachfolgerbeziehung zueinander.

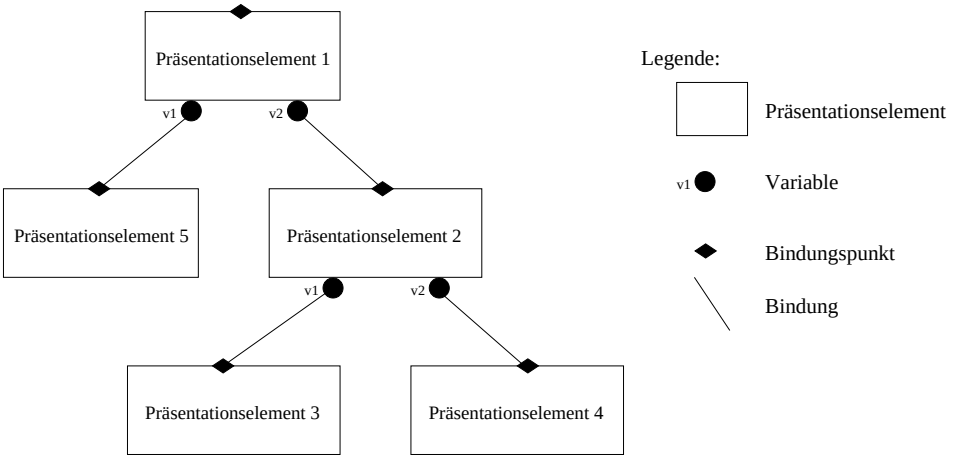


Abbildung 5.10: Nachfolger in einem Spezifikationsbaum

**Definition 16 ($spatial_p$-Projektor).**
Der Projektor $spatial_p(x, y, width, height, priority, type)$, $V_{spatial_p} = \emptyset$, $x$, $y$, $width$, $height$ und $priority \in \mathcal{N}_0$, $type \in \{pixel, percent\}$ definiert den rechteckigen Ausschnitt auf der Präsentationsfläche, auf dem dasjenige Präsentationselement und dessen Nachfolger präsentiert werden, an das der Projektor über eine Projektionsvariable gebunden ist. Diese werden so skaliert, daß sie den durch $x$, $y$, $width$ und $height$ spezifierten Bereich exakt ausfüllen. Hierbei sind zwei Fälle zu unterscheiden.

Gilt $type = pixel$, definieren $x$, $y$ die linke obere Ecke, $width$ und $height$ die Höhe und Breite des Ausschnittes in Pixeln. Gilt jedoch $type = percent$, so sind $x$, $y$, $width$ und $height$ prozentuale Angaben bezüglich der Präsentationsfläche.

Der Parameter $priority$ dient dazu, sich überlappende Bereiche zu ordnen. Diese werden so nach dem Parameter $priority$ sortiert, daß ein Bereich mit größerer Priorität Bereiche mit geringerer Priorität überdeckt.

$spatial_p$-Projektoren können hierarchisch verschachtelt werden. Wenn ein Präsentationselement $e$ einen $spatial_p$-Projektor $s$ an sich bindet, so beziehen sich die Koordinatenangaben der an die Nachfolger von $e$ gebundenen $spatial_p$-Projektoren auf den durch $s$ definierten Präsentationsbereich und nicht auf die gesamte Präsentationsfläche. Passen diese Koordinatenangaben nicht in den von $s$ definierten Bereich, werden sie durch Clipping angepaßt (siehe Abbildung 5.11). Formal bedeutet dies:

Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$. Jedem Präsentationselement $e \in E$ ist die räumliche Projektionsumgebung $spe_e = (x_e, y_e, width_e, height_e, priority_e)$ zugeordnet. Sei nun $e'$ ein direkter Nachfolger von $e$:

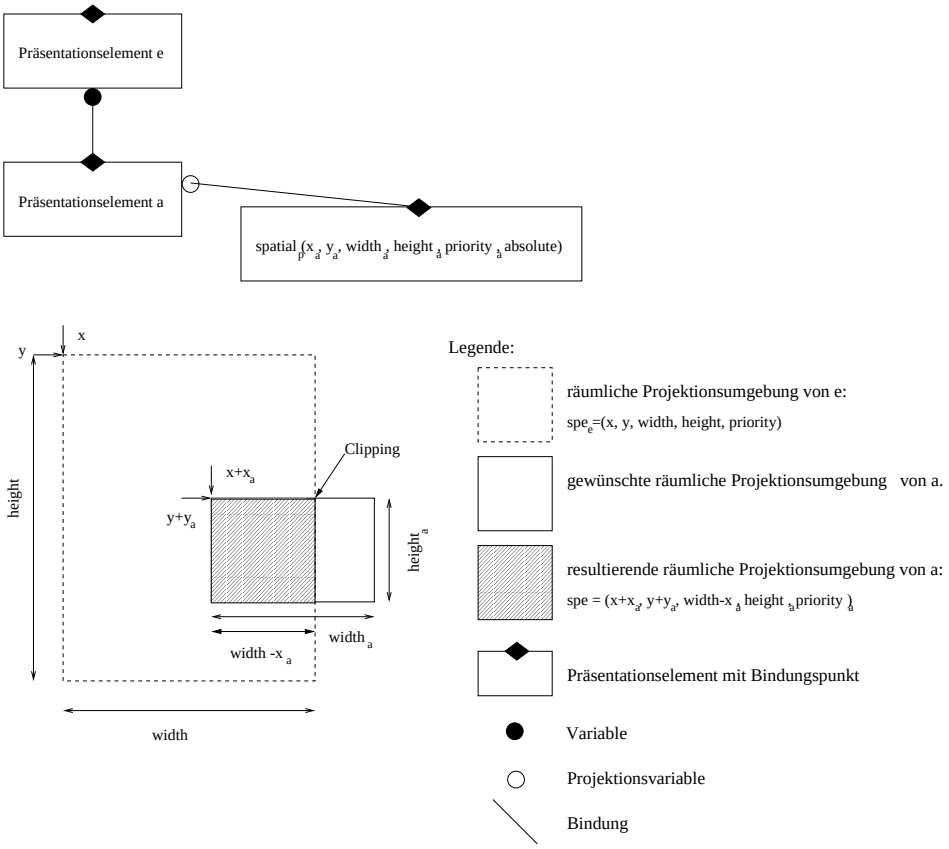


Abbildung 5.11: Verschachtelung von $spatial_p$-Operatorelementen

- Ist an $e'$ kein $spatial_p$-Projektor gebunden, so gilt: $spe_{e'} = spe_e$.

- Ist an $e'$ der Projektor $spatial_p(x, y, width, height, priority, type)$ gebunden und $type = pixel$, so gilt für $spe_{e'} = (x_{e'}, y_{e'}, width_{e'}, height_{e'}, priority_{e'})$:

$$
\begin{aligned}
x_{e'} &= x + x_e \\
y_{e'} &= y + y_e \\
width_{e'} &= \begin{cases} max(0, width_e - x), & \text{falls } x + width \geq width_e \\ width, & \text{sonst} \end{cases} \\
height_{e'} &= \begin{cases} max(0, height_e - y), & \text{falls } y + height \geq height_e \\ height, & \text{sonst} \end{cases} \\
priority_{e'} &= priority
\end{aligned}
$$

- Ist an $e'$ der Projektor $spatial_p(x, y, width, height, priority, type)$ gebunden und $type = percent$, so gilt für $spe_{e'} = (x_{e'}, y_{e'}, width_{e'}, height_{e'}, priority_{e'})$:

$$
\begin{aligned}
x_{e'} &= x_e + round(width_e \times x/100) \\
y_{e'} &= y_e + round(width_e \times y/100) \\
width_{e'} &= round(width_e \times width/100) \\
height_{e'} &= round(height_e \times height/100) \\
priority_{e'} &= priority
\end{aligned}
$$

Ist nun $a$ ein atomares Medienobjekt mit visueller Komponente oder ein externes Medienobjekt, so wird es bei der Präsentation exakt auf den durch $spe_a$ spezifizierten Präsentationsbereich skaliert und dort dargestellt.

□

Die verschachtelte Spezifikation von $spatial_p$-Projektoren ist besonders bei der Definition von Präsentationschablonen sinnvoll. In Schablonen können Präsentationsbereiche mit $spatial_p$-Projektoren vordefiniert werden. Ein Benutzer der Schablone kann später Spezifikationsbäume, die evtl. weitere $spatial_p$-Projektoren enthalten, an die von der Schablone nicht gebundenen Variablen binden und sicher sein, daß die spezifierten Dokumentbestandteile innerhalb der vom Schablonenersteller vorgesehenen Präsentationsbereichen dargestellt werden.

**Definition 17 ($temporal_p$-Projektor).**
Der Projektor $temporal_p(direction, speed)$, $V_{temporal_p} = \emptyset$, $direction \in \{-1, 1\}$ und $speed \in \Re_0^+$, legt die Richtung und Geschwindigkeit der Wiedergabe von demjenigen Präsentationselement und dessen Nachfolgern (sofern es sich um kontinuierliche Medienobjekte handelt) fest, an das der Projektor über eine Projektionsvariable gebunden ist. $direction = -1$ bedeutet Wiedergabe rückwärts, $direction = 1$ bedeutet Wiedergabe vorwärts. Die Wiedergabegeschwindigkeit wird mit dem Faktor $speed$ multipliziert.

Analog zum $spatial_p$-Projektor können auch $temporal_p$-Projektoren hierarchisch verschachtelt werden. Dieses wird formal wie folgt beschrieben:

Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$. Jedem Präsentationselement $e \in E$ ist eine temporale Projektionsumgebung $tpe_e = (direction_e, speed_e)$ zugeordnet. Sei nun $e'$ ein direkter Nachfolger von $e$:

- Ist an $e'$ kein $temporal_p$-Projektor gebunden, so gilt: $tpe_{e'} = tpe_e$.
- Ist an $e'$ der Projektor $temporal_p(direction, speed)$ gebunden, so gilt für $tpe_{e'} = (direction_{e'}, speed_{e'})$:

$$
\begin{aligned}
speed_{e'} &= speed \times speed_e \\
direction_{e'} &= direction \times direction_e
\end{aligned}
$$

Ist nun $a$ ein atomares, kontinuierliches Medienobjekt (Video, Audio), so wird es bei der Präsentation in der durch $tpe_a$ spezifizierten Geschwindigkeit und Richtung dargestellt.

□

Ein Beispiel soll erläutern, warum diese Definition der Berechnung der temporalen Projektionsumgebung sinnvoll ist. Ein Autor hat ein Video und einen $temporal_p$-Projektor $t_1$ so in einem komplexen Medienobjekt $km$ gekapselt, daß bei der Präsentation von $km$ das Video rückwärts in doppelter Geschwindigkeit abgespielt wird. Ein anderer Autor will nun $km$ in seinem Spezifikationsbaum benutzen. Er will das Video aber vorwärts in Normalgeschwindigkeit darstellen. Da er die von $km$ gekapselte Spezifikation nicht näher betrachten möchte, bindet er an $km$ an einen weiteren $temporal_p$-Projektor $t_2$, der $km$ in halber Geschwindigkeit rückwärts darstellt. $t_1$ und $t_2$ heben sich gegenseitig auf, so daß das Video bei der Präsentation wie gewünscht angezeigt wird.

**Definition 18 ($acoustic_p$-Projektor).**
Der Projektor $acoustic_p(volume, balance, base, treble)$, $V_{acoustic_p} = \emptyset$, $balance$, $base$ und $treble \in [-1, \ldots, 1]$, $volume \in [0, \ldots, 100]$, legt die Lautstärke, die Balance, die Stärken des Basses und der Höhen bei der Wiedergabe von demjenigen Präsentationselement sowie dessen Nachfolgern (sofern es sich um Medienobjekte mit akustischer Komponente wie Audio oder Video handelt) fest, an die der Projektor über eine Projektionsvariable gebunden ist.

Dabei bedeuten:

- $volume$ ist die Lautstärke, mit der das Medienobjekt bei seiner Wiedergabe präsentiert wird. $volume = 0$ bedeutet dabei, daß eine Wiedergabe ohne Ton stattfindet, $volume = 100$ hingegen entspricht einer Wiedergabe mit maximaler Lautstärke.
- $balance$ gibt das Verhältnis der Lautstärken des linken bzw. rechten Lautsprechers bei der Stereowiedergabe wieder. Bei $balance = -1$ wird mit maximaler Lautstärke über den linken Lautsprecher abgespielt, während der rechte Lautsprecher stumm bleibt. $balance = 1$ ist der inverse Fall. $balance = 0$ entspricht der gleichmäßigen Wiedergabe über beide Lautsprecher.
- $base$, $treble$ geben den Bass- bzw. Höhenpegel der jeweiligen Wiedergabe an.

Zwar können $acoustic_p$-Projektoren genau wie $spatial_p$-Projektoren hierarchisch verschachtelt werden, jedoch überschreiben die Einstellungen eines an ein Präsentationselement gebundenen $acoustic_p$-Projektors die Einstellungen von etwaigen Vorgängern. Dieses wird formal wie folgt beschrieben:

Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$. Jedem Präsentationselement $e \in E$ ist eine akustische Projektionsumgebung $ape_e = (volume_e, balance_e, base_e, treble_e)$ zugeordnet. Sei nun $e'$ ein direkter Nachfolger von $e$:

- Ist an $e'$ kein $acoustic_p$-Projektor gebunden, so gilt: $ape_{e'} = ape_e$.
- Ist an $e'$ der Projektor $acoustic_p(volume, balance, base, treble)$ gebunden, so gilt für $ape_{e'} = (volume_{e'}, balance_{e'}, base_{e'}, treble_{e'})$:

$$\begin{aligned} volume_{e'} &= volume \\ balance_{e'} &= balance \\ base_{e'} &= base \\ treble_{e'} &= treble \end{aligned}$$

Ist nun $a$ ein atomares Medienobjekt oder ein externes Medienobjekt mit akustischer Komponente, so wird es bei der Präsenation in der durch $ape_a$ spezifizierten Klangform wiedergegeben.

□

**Definition 19 ($typographic_p$-Projektor).**
Der Projektor $typographic_p(font, size, style)$, $V_{typographic_p} = \emptyset$, legt den Zeichensatz bei der Darstellung von demjenigen Präsentationselement sowie dessen Nachfolgern (sofern es sich um Medienobjekte mit textueller Komponente handelt) fest, an die der Projektor über eine Projektionsvariable gebunden ist.

Dabei sind:

- $font$ ist der Name des Zeichensatz.
- $size$ gibt die Schriftgröße in Punkten an.
- $style$ gibt die Stilvariante des Zeichensatzes an: $style \in \{normal, italic, bold, italic + bold\}$

Zwar können $typographic_p$-Projektoren wie $spatial_p$-Projektoren hierarchisch verschachtelt werden, jedoch überschreiben die Einstellungen eines an ein Präsentationselement gebundenen $typographic_p$-Projektors die Einstellungen von etwaigen Vorgängern. Dieses wird formal wie folgt beschrieben:

Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$. Jedem Präsentationselement $e \in E$ ist eine typographische Projektionsumgebung $type_e = (font_e, size_e, style_e)$ zugeordnet. Sei nun $e'$ ein direkter Nachfolger von $e$:

- Ist an $e'$ kein $typographic_p$-Projektor gebunden, so gilt: $type_{e'} = type_e$.
- Ist an $e'$ der Projektor $typographic_p(font, size, style)$ gebunden, so gilt für $type_{e'} = (font_{e'}, size_{e'}, style_{e'})$ :

$$\begin{aligned} font_{e'} &= font \\ size_{e'} &= size \\ style_{e'} &= style \end{aligned}$$

Ist nun $a$ ein atomares Medienobjekt oder ein externes Medienobjekt mit Textinhalt, so wird dieser bei der Präsentation in dem durch $type_a$ spezifizierten Zeichensatz wiedergegeben.

□

**Definition 20 ($temporal_s$-Selektor).**
Der Selektor $temporal_s(v, start, duration)$, $V_{temporal_s} = \{v\}$, $duration \in \Re_0^+$ und $start \in \Re_0^+$, legt fest, daß die Präsentation der direkten oder indirekten Nachfolger des Selektors, falls es sich um atomare, kontinuierliche Medienobjekte handelt, bei der Position $start$ msec nach dem originalen Startzeitpunkt beginnt und $duration$ msec andauert.

$temporal_s$-Selektoren sind hierarchisch verschachtelbar. Dabei sind die Angaben des durch einen $temporal_s$-Selektors $t$ spezifizierten Präsentationsintervalles relativ

auf evtl. schon spezifizierte Intervalle der direkten oder indirekten Nachfolger von $t$ bezogen. Formal:

Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$. Jedem Präsentationselement $e \in E$ ist eine temporale Selektionsumgebung $tse_e = (start_e, duration_e)$ zugeordnet. Sei nun $e'$ ein direkter Nachfolger von $e$:

- Ist $e'$ kein $temporal_s$-Selektor, so gilt: $tse_{e'} = tse_e$.
- Ist $e' = temporal_s(v, start, duration)$, so gilt für $tse_{e'} = (start_{e'}, duration_{e'})$ :

$$start_{e'} = start_e + start$$

$$duration_{e'} = \begin{cases} max(0, duration - start_e), & \text{falls } start_e + duration_e \geq duration \\ duration_e, & \text{sonst} \end{cases}$$

Ist nun $a$ ein kontinuierliches, atomares Medienobjekt, so wird von diesem bei der Präsentation nur der durch $tse_a$ spezifizierte Zeitausschnitt dargestellt.

□

Abbildung 5.12 zeigt ein Beispiel für die Verschachtelung von $temporal_s$-Selektoren. Der Selektor $s2 = temporal_s(v_{s2}, 10000, 40000)$ ist direkter Nachfolger von $s1 = temporal_s(v_{s2}, 10000, 25000)$. $s1$ bezieht sich damit auf das von $s2$ spezifierte Intervall. Folglich wird vom Video der Zeitausschnitt präsentiert, der bei $t = 20sec$ beginnt und bei $t = 45sec$ endet.

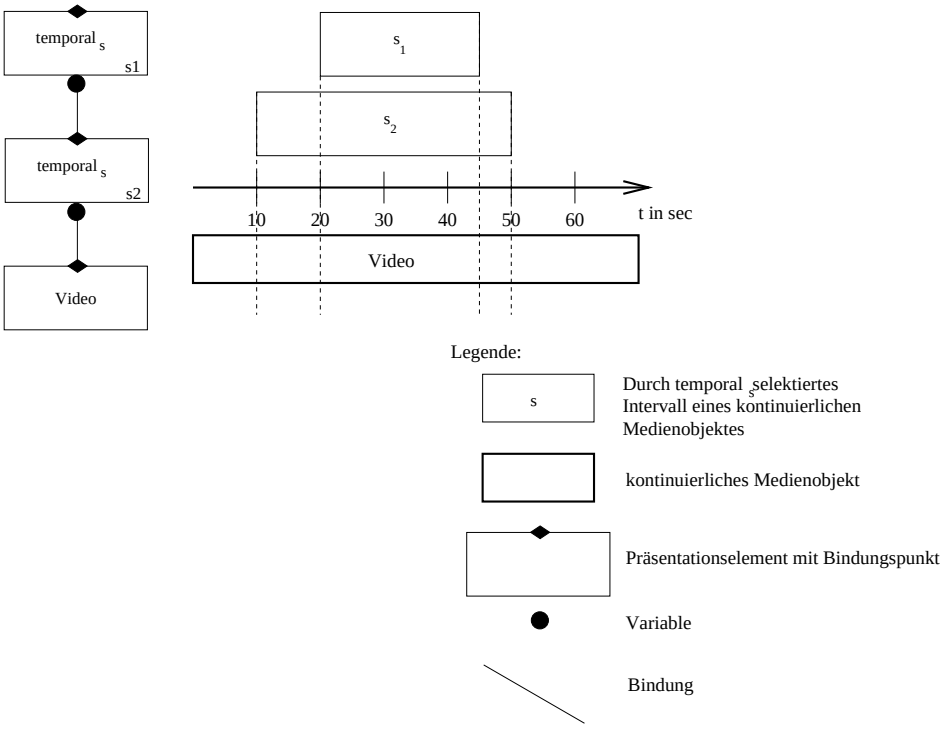


Abbildung 5.12: Selektion eines Zeitintervalls

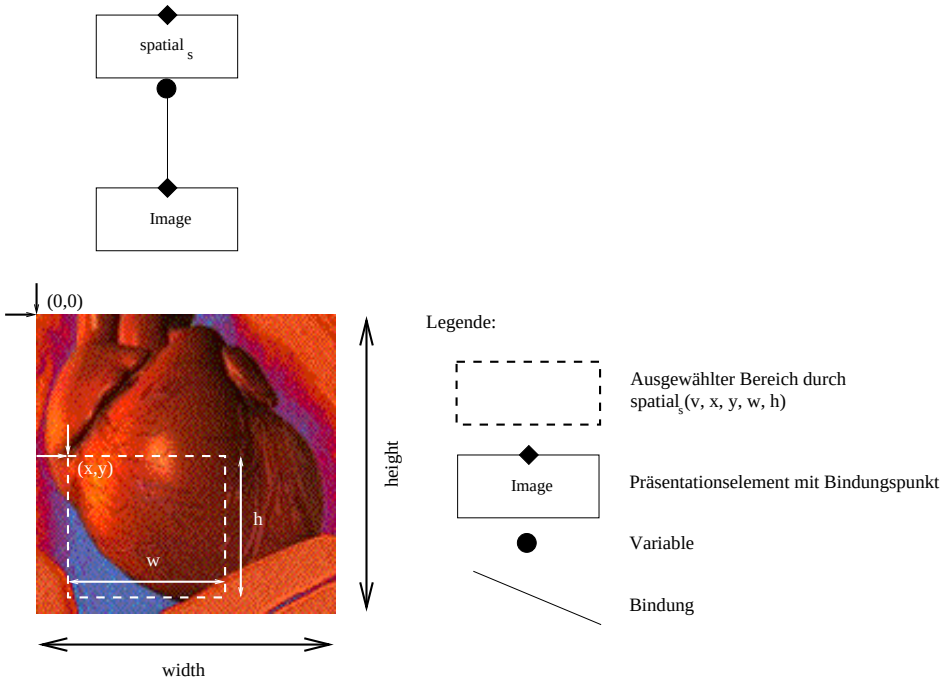


Abbildung 5.13: Selektion eines Bildauschnittes

**Definition 21 ($spatial_s$-Selektor).**
Der Selektor $spatial_s(v, x, y, width, height)$, $V_{temporal_s} = \{v\}$, $x$, $y$, $width$ und $height \in \mathcal{N}_I$, legt fest, daß die Präsentation der direkten oder indirekten Nachfolger des Selektors, falls es sich um atomare Medienobjekte mit visueller Komponente handelt, nur den durch $x$,$y$,$width$ und $height$ spezifizierten Bildausschnitt dieser Objekte darstellt (siehe Abbildung 5.13). Dabei sind $x$,$y$,$width$ und $height$ absolute Angaben in Pixeln.

$spatial_s$-Selektoren sind hierarchisch verschachtelbar. Dabei beziehen sich die Angaben des durch einen $spatial_s$-Selektor $s$ spezifizierten Bildausschnittes relativ auf evtl. schon spezifizierte Bildauschnitte von dessen direkten oder indirekten Nachfolgern. Formal:

Sei $S$ ein Spezifikationsbaum, $flatten(S) = (E, B)$. Jedem Präsentationselement $e \in E$ ist eine räumliche Selektionsumgebung $sse_e = (x_e, y_e, width_e, height_e)$ zugeordnet. Sei nun $e'$ ein direkter Nachfolger von $e$:

- Ist $e'$ kein $spatial_s$-Selektor, so gilt: $sse_{e'} = sse_e$.
- Ist $e' = spatial_s(v, x, y, width, height$,
  so gilt für $sse_{e'} = (x_{e'}, y_{e'}, width_{e'}, height_{e'})$ :

$$
\begin{aligned}
x_{e'} &= x + x_e \\
y_{e'} &= y + y_e \\
width_{e'} &= \begin{cases} max(0, width - x_e), & \text{falls } x_e + width_e \geq width \\ width_e, & \text{sonst} \end{cases} \\
height_{e'} &= \begin{cases} max(0, height - y_e), & \text{falls } y_e + height_e \geq height \\ height_e, & \text{sonst} \end{cases}
\end{aligned}
$$

Ist nun $a$ ein atomares Medienobjekt mit visueller Komponente, so wird bei der Präsentation nur der durch $sse_a$ spezifizierte Bildausschnitt von $a$ dargestellt.

□

### 5.3.4 Interaktionsoperatorelemente

Wie in Kapitel 3 erläutert wurde, werden in dieser Arbeit Navigations- und Gestaltungsinteraktionen als wesentlich betrachtet. Für die navigierende Interaktion werden ein Operatorelement zur Definition eines Links und ein Operatorelement zur Definition eines Menüs zur Verfügung gestellt.

**Definition 22 ($link$-Operatorelement).**
Das Operatorelement $link((v_1,t_1),\ldots,(v_n,t_n))$, $V_{link} = \{v_1,\ldots,v_n,t_1,\ldots,t_n\}$, definiert eine Menge von Links zwischen den Präsentationselementen $v_i, i = 1\ldots n$ und den in den komplexen Medienobjekten $t_j, j = 1\ldots n$ gekapselten Dokumenten. $v_i$ sind dabei Anker und werden beim Start der Präsentation des $link$-Operatorelements direkt präsentiert. Findet eine Interaktion mit einem Anker $v_x$ statt, so wird die Präsentation des aktuellen Dokumentes komplett beendet und die Darstellung des in $t_x$ gekapselten Dokumentes eingeleitet. Die Präsentation eines Links ist beendet, wenn entweder die Präsentation der Anker beendet oder mit einem dieser Anker interagiert wurde.

□

Das $link$-Operatorelement erlaubt navigierende Interaktion von einem Dokument zum anderen. Damit sind Hypertextstrukturen modellierbar. Jedoch ist auch wünschenswert, innerhalb eines Dokumentes eine Auswahl zwischen verschiedenen Präsentationsabläufen treffen zu können. Hierzu dient das $menu$-Operatorelement.

**Definition 23 ($menu$-Operatorelement).**
Das Operatorelement $menu((v_1,t_1),\ldots,(v_n,t_n),default,mode)$, $V_{menu} = \{v_1,\ldots,v_n,t_1,\ldots,t_n,default\}$, $mode \in \{vanish,prevail\}$ definiert eine Auswahlmöglichkeit. Startet die Präsentation des $menu$-Operatorelements, werden die mit $v_1,\ldots,v_n$ und $default$ verbundenen Präsentationselemente parallel gestartet. Danach wird abhängig vom Parameter $mode$ verfahren:

- $mode = vanish$: Tritt eine Interaktion mit einem $v_i, 1 \leq i \leq n$ auf, so werden die Präsentationen aller $v \in \{v_1,\ldots,v_n\}$ und von $default$ abgebrochen und die Präsentation des zu $v_i$ gehörigen $t_i$ eingeleitet. Die Präsentation des $menu$-Operatorelements endet in diesem Fall, sobald die Präsentation von $t_i$ endet. Tritt keine Interaktion mit einem $v_i$ auf, so endet die Präsentation des $menu$-Operatorelements, sobald die Präsentationen aller $v \in \{v_1,\ldots,v_n\}$ beendet sind.

- $mode = prevail$: Tritt eine Interaktion mit einem $v_i, 1 \leq i \leq n$ auf, so wird zusätzlich die Präsentation des zu $v_i$ gehörenden $t_i$ eingeleitet. Jede weitere Interaktion mit einem $v_j, 1 \leq j \leq n$ beendet die Darstellung aller präsentierten $t \in \{t_1,\ldots,t_n\}$ und zeigt das zu $v_j$ gehörige $t_j$ an. Die Präsentation des $menu$-Operatorelements endet, sobald die Präsentationen aller $v \in \{v_1,\ldots,v_n\}$ beendet sind.

□

Neben den obigen Operatorelementen zur navigierenden Interaktion müssen noch Operatorelemente zur gestalterischen Interaktion eingeführt werden. Abstrahiert man von der Sicht, daß gestalterische Interaktionen durch Interaktionsformen implementiert werden, deren Manipulation Präsentationseigenschaften von Medienobjekten ändert, so stellt man fest, daß gestalterische Interaktionen eine spezielle Form der gestalterischen Beziehungen darstellen. Diese speziellen gestalterischen Beziehungen können durch den Betrachter eines Dokumentes beeinflußt werden.

Die gestalterischen Interaktionsoperatorelemente des ZYX-Dokumentmodells orientieren sich deshalb an dessen gestalterischen und räumlichen Operatorelementen. In der Tat sind sie semantisch mit diesen bis auf die Interaktionsmöglichkeit identisch. Die konkrete Interaktionsform wird nicht festgelegt und bleibt implementierungsabhängig. So wird z.B. nicht definiert, ob die Änderung der Lautstärke eines Audios mit einem Slider, mit einem Drehknopf oder über ein Menü erfolgt. Wegen der Ähnlichkeit zu den Projektoren bzw. Selektoren werden die gestalterischen Interaktionsoperatorelemente *Interaktoren* genannt.

**Definition 24 (Interaktor).**
Ein *Interaktor* ist ein Operatorelement, welches zur Beschreibung von gestalterischer Interaktion genutzt wird. Der Bezeichner eines Interaktors wird durch ein zusätzliches $i$ indiziert.

□

**Definition 25 ($spatial_{pi}$-Interaktor).**
Der Interaktor $spatial_{pi}(x, y, width, height, priority, type)$, $V_{spatial_{pi}} = \emptyset$, $x$ und $y \in \mathcal{N}_0$, $width$, $height$ und $priority \in \mathcal{N}_0$, $type \in \{percent, pixel\}$, ist ein spezieller $spatial_p$-Projektor, bei dem zusätzlich die Möglichkeit besteht, über Benutzerinteraktionen die Parameter $x$, $y$, $width$, $height$ und $priority$ innerhalb der erlaubten Bereiche zu ändern. Dieses impliziert, daß dynamisch zur Laufzeit die räumlichen Projektionsumgebungen von demjenigen Präsentationselement und dessen Nachfolgern aktualisiert werden müssen, an das der Interaktor gebunden ist.

□

**Definition 26 ($temporal_{pi}$-Interaktor).**
Der Interaktor $temporal_{pi}(direction, speed)$, $speed \in \Re_0^+$, $direction \in \{-1, 1\}$ und $V_{temporal_{pi}} = \emptyset$, ist ein spezieller $temporal_p$-Projektor, bei dem zusätzlich die Möglichkeit besteht, über Benutzerinteraktionen die Parameter $direction$ und $speed$ innerhalb der erlaubten Bereiche zu ändern. Dieses impliziert, daß dynamisch zur Laufzeit die temporalen Projektionsumgebungen von demjenigen Präsentationselement und dessen Nachfolgern aktualisiert werden müssen, an das der Interaktor gebunden ist.

□

**Definition 27 ($acoustic_{pi}$-Interaktor).**
Der Interaktor $acoustic_{pi}(volume, balance, base, treble)$, $V_{acoustic_{pi}=\emptyset}$, $base$, $balance$ und $treble \in [-1, \ldots, 1]$, $volume \in [1, \ldots, 100]$, ist ein spezieller $acoustic_p$-Projektor, bei dem zusätzlich die Möglichkeit besteht, über Benutzerinteraktionen die Parameter $volume$, $balance$, $base$ und $treble$ innerhalb der erlaubten Bereiche zu ändern. Dieses impliziert, daß dynamisch zur Laufzeit die akustischen Projektionsumgebungen von demjenigen Präsentationselement und dessen Nachfolgern aktualisiert werden müssen, an das der Interaktor gebunden ist.

□

**Definition 28 ($typographic_{pi}$-Interaktor).**
Der Interaktor $typographic_{pi}(font, size, style)$, $V_{typographic_{pi}} = \emptyset$, ist ein spezieller $typographic_p$-Projektor, bei dem zusätzlich die Möglichkeit besteht, über Benut-

zerinteraktionen die Parameter *font*, *size* und *style* innerhalb der erlaubten Bereiche zu ändern. Dieses impliziert, daß dynamisch zur Laufzeit die typographischen Projektionsumgebungen von demjenigen Präsentationselement und dessen Nachfolgern aktualisiert werden müssen, an das der Interaktor gebunden ist.

□

**Definition 29 ($temporal_{si}$-Interaktor).**
Der Interaktor $temporal_{si}(v, start, duration)$, $start \in \Re_0^+$, $duration \in \Re_0^+$ und $\{v\} = V_{temporal_{si}}$, ist ein spezieller $temporal_s$-Selektor, bei dem zusätzlich die Möglichkeit besteht, über Benutzerinteraktionen die Parameter *start* und *duration* innerhalb der erlaubten Bereiche zu ändern.

□

**Definition 30 ($spatial_{si}$-Interaktor).**
Der Interaktor $spatial_{si}(v, x, y, width, height)$, $\{v\} = V_{spatial_{si}}$, $x$, $y$, $width$, $height$ und $priority \in \mathcal{N}_0$, ist ein spezieller $spatial_s$-Selektor, bei dem zusätzlich die Möglichkeit besteht, über Benutzerinteraktionen die Parameter $x$, $y$, $width$, $height$ und $priority$ innerhalb der erlaubten Bereiche zu ändern.

□

### 5.3.5 Adaptionsoperatorelemente

In diesem Abschnitt werden insbesonders diejenigen Operatorelemente des ZYX-Dokumentmodells eingeführt, welche die Adaption eines ZYX-Dokumentes an das Profil eines Benutzers ermöglichen, weswegen in der weiteren Behandlung diese Operatorelemente Adaptionsoperatorelemente genannt werden. In der Tat steht vor der Präsentation eines adaptiven Dokumentes nicht fest, was dargestellt wird. Erst während der Präsentation wird über die genaue Darstellung entschieden.

Die Grundlage für adaptive Dokumente bilden die Metainformationen, die an Fragmente gebunden sein können. Fragmente sind aber nicht die einzigen Quellen von Metainformationen. Im folgenden wird davon ausgegangen, daß einem Betrachter eines Dokumentes ebenfalls eine Menge von Metainformationen zugeordnet ist (genannt globales Profil) und daß die Adaptionsoperatorelemente diese Menge kennen. Somit ist man in der Lage, das Benutzerprofil eines Betrachters mit Metainformationen zu modellieren und bei der Adaption zu berücksichtigen.

**Definition 31 (Globales Profil).**
Das globale Profil $P = \{(k_1, d_1), \ldots, (k_n, d_n)\}$ ist eine Menge von Metainformationen, die dem Betrachter eines Dokuments zugeordnet sind.

□

Wenn man anhand von Metainformationen entscheiden möchte, was dargestellt wird, so benötigt man ein Vergleichsprädikat, das zwei Mengen von Metainformationen miteinander vergleicht und bezüglich ihrer Übereinstimmung bewertet.

**Definition 32 ($matching(Meta_1, Meta_2)$).**
Seien $Meta_1$ und $Meta_2$ Mengen von Metainformationen. Dann ist $matching(Meta_1, Meta_2)$ wahr genau dann, wenn gilt: $\forall (k, d) \in Meta_1 : \exists (k, d') \in Meta_2 \Rightarrow d = d'$.

□

Dieses Prädikat gibt demnach an, ob alle Metainformationen, deren Schlüssel sowohl in $Meta_1$ als auch in $Meta_2$ vorkommen, in den zu den Schlüssel gehörenden Werten übereinstimmen. Seien beispielsweise folgende drei Mengen gegeben:

$M_1 = \{(Thema, Zinkoxid), (Uni, Ulm)\}$
$M_2 = \{(Art, Seminar), (Thema, Zinkoxid), (Uni, Ulm)\}$
$M_3 = \{(Art, Vorlesung), (Thema, Zinkoxid), (Leser, Bronkowitz)\}$

Es gilt $matching(M_1, M_2)$ und $matching(M_1, M_3)$, aber nicht $matching(M_2, M_3)$, weil die Tupel $(Art, Seminar)$ und $(Art, Vorlesung)$ nicht in den Werten übereinstimmen, obwohl sie denselben Schlüssel aufweisen.

Es ist nun aber in einem adaptierbaren Dokument nicht ausreichend, sich anhand eines Wahrheitswertes für ein Präsentationselement zu entscheiden. So gilt im obigen Beispiel sowohl $matching(M_1, M_2)$ als auch $matching(M_1, M_3)$. Es stellt sich hier die Frage, welche Menge ,,besser" zu $M_1$ paßt. Deswegen wird ein Maß für die Übereinstimmung zweier Mengen von Metainformationen benötigt.

**Definition 33 ($matches(Meta_1, Meta_2)$).**
Seien $Meta_1$ und $Meta_2$ Mengen von Metainformationen. Dann ist
$matches(Meta_1, Meta_2) = |\{i \in Meta_1 | i \in Meta_2\}|$.

□

Dieses Maß gibt die Zahl der in zwei Metainformationsmengen identischen Tupel wieder. Auf das obige Beispiel angewendet ergeben sich folgende Zahlen:

$matches(M_1, M_2) = 2$, $matches(M_1, M_3) = 1$ und $matches(M_2, M_3) = 1$. Nach diesem Maß entspricht also $M_1$ eher $M_2$ als $M_3$.

Das erste Adaptionsoperatorelement, das hier definiert wird, soll die Auswahl zwischen mehreren, vorgebenen Präsentationsalternativen anhand des globalen Benutzerprofils ermöglichen. Er ist dabei an das Switch-Element von SMIL angelehnt und soll wie dieses eine clientbasierte Adaption ermöglichen.

**Definition 34 (*switch*-Operatorelement).**
Das Operatorelement $switch(v_1, \ldots, v_n, default, Meta_1, \ldots, Meta_n)$, $\{v_1, \ldots, v_n, default\} = V_{switch}$, $Meta_i, i = 1 \ldots n$ sind Mengen von Metainformationen, erlaubt die Entscheidung zwischen $n + 1$ Präsentationsalternativen anhand des globalen Profils $P$. Die gewählte Alternative $a$ wird dargestellt und die Präsentation des *switch*-Operators endet, wenn die Präsentation von $a$ beendet ist.

Die Wahl der zu präsentierenden Alternative $a$ geschieht wie folgt:

1. Beginne mit $i = 1$.
2. Gilt $matching(P, Meta_i) \wedge matches(P, Meta_i) \geq 1$, so ist $a$ das mit $v_i$ verbundene Präsentationselement. Die Auswahl ist beendet und $a$ wird präsentiert.
3. Ist $i < n$, so setze $i = i + 1$ und gehe zu 2. Ansonsten gehe zu 4.
4. Das globale Profil paßt nicht zu den Präsentationsalternativen $v_1, \ldots, v_n$. Dann ist $a$ das mit $default$ verbundene Präsentationselement. $a$ wird präsentiert.

□

Beim *switch*-Operatorelement muß der Autor eines Dokumentes alle möglichen Alternativen vorgeben. Es kann jedoch Situationen geben, in denen ein Autor gar nicht

alle Alternativen kennt, sondern alle bekannten Fragmente (i.e. alle auf vorhandenen Fragmente) zur Auswahl heranziehen möchte. Dieses macht zum Beispiel Sinn, wenn ein Autor im Anschluß an die Präsenation seines Dokumentes, ein Dokument über das Thema ,,Herzchirurgie" präsentieren möchte, er aber nicht weiß, welche Dokumente von welchen Autoren über das Thema vorhanden sind. Außerdem hat die Auswahl unter allen Dokumenten den angenehmen Nebeneffekt, daß immer der aktuelle Bestand an Dokumenten berücksichtigt wird. Ein Autor muß somit sein Dokument bei Änderung des Dokumentbestandes nicht anpassen.

**Definition 35 (*query*-Operatorelement).**
Das Operatorelement $query(Meta_1, \ldots, Meta_n)$, $V_{query} = \emptyset$, $Meta_i, i = 1 \ldots n$ sind Mengen von Metainformationen, sucht beim Start seiner Präsentation ein spezielles Fragment $f$ unter allen bekannten Fragmenten, welches dann dargestellt wird. Das Ende der Präsentation von $f$ beendet die Präsentation des *query*-Operatorelements.

Sei $P$ das globale Profil. Die Suche nach $f$ geschieht dann folgendermaßen:

1. Setze $i = 1$.
2. Bilde die Teilmenge $F$ aller Fragmente, für die gilt:
   $\forall f \in F : matching(M_f, Meta_i \cup P) \wedge V_f = \emptyset$.

   Wähle aus dieser Menge dasjenige $f_{max}$ mit:

   $matches(M_{f_{max}}, Meta_i \cup P)$ ist maximal in $F$ und $matches(M_{f_{max}}, Meta_i \cup P) \geq 1$.

   Existiert ein solches $f_{max}$, so wird die Suche beendet und $f = f_{max}$. Existiert kein solches $f_{max}$ oder ist $F = \emptyset$, geht es weiter weiter bei 3.
3. Wenn $i < n$, so setze $i = i + 1$ und gehe zu 2.
4. Es wurde für alle $Meta_i, i = 1 \ldots n$ kein passendes $f$, gefunden: Beende die Präsentation des *query*-Operatorelements.

□

Das *query*-Operatorelement wurde mit dem Hintergedanken eingeführt, eine serverbasierte Adaption zu ermöglichen.

## 5.4 Bewertung

Nachdem im vorigen Abschnitt das ZYX-Dokumentmodell definiert wurde, soll es nun auf die Erfüllung der in Kapitel 3 gestellten Anforderungen geprüft werden. Eine Zusammenfassung des Ergebnisses der Prüfung findet sich in Tabelle 5.1 auf Seite 71 wieder.

### 5.4.1 Wiederverwendbarkeit

Die Forderung nach Wiederverwendbarkeit von Dokumentbestandteilen wird in drei Teilforderungen zerlegt. Da gibt es einmal die Forderung nach Wiederverwendbarkeit auf möglichst vielen Abstraktionsebenen. Das ZYX-Dokumentmodell erlaubt

generell die Wiederverwendung von Fragmenten. Fragmente sind atomare Medienobjekte, externe Medienobjekte aber auch komplexe Medienobjekte. Diese wiederum kapseln Spezifikationsbäume, die Dokumentteile, Dokumentschablonen und ganze Dokumente beschreiben können. Ein Autor unterliegt bei der Strukturierung seines Dokumentes mit komplexen Medienobjekten in logisch zusammenhängende Einheiten keinen Einschränkungen. Damit bietet die ZYX-Dokumentmodell Wiederverwendbarkeit auf allen in Kapitel 3 definierten Ebenen: der Ebene der Medienobjekte, der Ebene der Fragmente und der Ebene vollständiger Dokumente. Dieses ermöglicht bei guter Strukturierung einen hohen Grad an Wiederverwendung.

Die zweite Forderung betrifft die Art der Wiederverwendung. Prinzipiell ist die Wiederverwendung bei ZYX identisch. Wird nämlich in einem komplexen Medienobjekt *km* ein Projektor verwendet, so nimmt dieser auch Einfluß auf das Layout der Spezifikationen, in denen *km* verwendet wird. Dennoch kann durch die gezielte Nichtbindung von Projektionsvariablen eine gewisse Trennung der Struktur eines Fragmentes von dessen Layout erreicht werden. Der Autor eines komplexen Medienobjektes kann nämlich an denjenigen Stellen im Spezifikationsbaum Projektionsvariablen anbringen, von denen er meint, daß dort das Layout beeinflußt werden muß. Wenn der Autor diese Projektionsvariablen nicht bindet, werden sie vom komplexen Medienobjekt exportiert. Verwendet nun ein anderer Autor dieses komplexe Medienobjekt, so kann er nachträglich Projektoren gemäß seinen Layoutvorstellungen an die exportierten Projektionsvariablen binden. Das konkrete Layout wird also nicht von dem Fragment selbst vorgeschrieben.

Die dritte Teilforderung zielt auf die Auffindbarkeit von wiederverwendbaren Komponenten ab. ZYX unterstützt die facettenorientierte Klassifikation von Fragmenten durch das Konzept der Metainformationen. Dadurch ist das softwaregestützte Auffinden von Fragmenten realisierbar.

### 5.4.2 Interaktion

Eine weitere Forderung ist es, die Modellierung von Interaktion in multimedialen Dokumenten zu unterstützen. Als wesentliche Interaktionsarten werden in Kapitel 3 navigierende und gestalterische Interaktion gesehen. Für beide Arten stellt das ZYX-Dokumentmodell Operatorelemente zur Verfügung.

### 5.4.3 Adaption

ZYX definiert zwei Adaptionsoperatorelemente. Das *switch*-Operatorelement ermöglicht die Beschreibung von clientbasierter Adaption, während das *query*-Operatorelement zur Beschreibung von serverbasierter Adaption bereitgestellt wird. Somit sind beide grundlegenden Adaptionsarten und auch Mischformen von beiden realisierbar.

### 5.4.4 Präsentationsneutralität

Das ZYX-Dokumentmodell ist auf einer höheren semantischen Ebene einzustufen als die Dokumentformate SMIL bzw. MHEG-5. SMIL verfügt zwar über ähnliche temporale Operatoren wie ZYX (`<PAR>` und `<SEQ>` entsprechen ungefähr den *par*- bzw. *seq*-Operatorelementen), jedoch weist SMIL nicht die Strukturierungsmöglich-

| | | |
|---|---|---|
| Wiederverwendbarkeit | Granularität | Wiederverwendbarkeit auf allen Ebenen. |
| | Art | Identisch, dennoch gewisse Trennung von Struktur und Layout realisierbar. |
| | Selektion | Facettenorientierte Klassifikation über Metainformationen |
| Interaktion | | Navigierende und gestalterische Interaktion möglich. |
| Adaption | | Sowohl client- als auch serverbasierte vorgesehen. |
| Präsentationsneutralität | | Spezifikation von Dokumenten auf relativ hoher semantischer Ebene. Gut geeignet zur präsentationsneutralen Verwaltung von Dokumenten, falls externe Medienobjekte selten verwendet werden. |

Tabelle 5.1: Zusammenfassung der Bewertung von ZYX

keiten, die das ZYX-Modell einem Autor durch die komplexen Medienobjekte bietet. Außerdem kann durch Verwendung von Projektionsvariablen eine gewisse Trennung von Struktur und Layout erreicht werden. HyTime hingegen ist semantisch höher einzustufen, da es strikt zwischen Struktur und Layout einer Präsentation trennt. ZYX stellt somit bezüglich der semantischen Ebene einen Kompromiß zwischen der multimedialen Funktionalität von SMIL und MHEG-5 sowie dem Abstraktionsniveau von HyTime dar.

Inwieweit ZYX zur präsentationsneutralen Verwaltung multimedialer Dokumente geeignet ist, hängt wesentlich von der Verwendung externer Medienobjekte ab. Werden diese intensiv verwendet, um beim Import vorhandener Dokumente anderer Modelle Aufwand zu sparen, wird man hier dafür bestraft. Möchte man beispielsweise ein Dokument nach SMIL exportieren, welches externe Medienobjekte enthält, die MHEG-5 Dokumentbestandteile kapseln, so muß man einen Konverter zwischen MHEG-5 und SMIL bereitstellen. Diese Konvertierung vollautomatisch zu realisieren, dürfte sich, wie in Kapitel 3 dargelegt, als schwierig erweisen. Werden jedoch keine externen Medienobjekte verwendet, so eignet sich ZYX aufgrund seines relativ hohen semantischen Niveaus gut zur präsentationsneutralen Ablage von Dokumenten.

### 5.4.5 Erweiterbarkeit

In Kapitel 3 wird die Erweiterbarkeit eines Dokumentmodells nicht explizit gefordert. Jedoch ist es vorteilhaft, ein Modell leicht an neue Anforderungen oder Gegebenheiten anpassen zu können. Das ZYX-Dokumentmodell trennt strikt zwischen den wiederverwendbaren Teilen eines Dokuments, den Fragmenten, und den die Beziehungen modellierenden Operatorelementen. Neue Operatorelemente können relativ einfach ohne Änderung des Grundmodells hinzugefügt werden. Somit sind auch in dieser Arbeit nicht realisierte Features (wie z.Bsp. Fading von visuellen Objekten oder Textformatierung) in Form von neuen Operatorelementen in das Modell integrierbar.

# Kapitel 6

# Modellierung von ZYX in einem DBMS

Im vorhergehenden Kapitel wurde das ZYX-Dokumentmodell definiert, welches die Beschreibung von wiederverwendbaren, interaktiven und adaptiven multimedialen Dokumenten und Fragmenten auf einer hohen semantischen Ebene ermöglicht und somit die Forderung des Projektes ,,Galerie der Herzchirurgie" (siehe Kapitel 3) nach einer flexiblen kontextabhängigen und modularen Kombinierbarkeit von Fragmenten erfüllt. Eine weitere Forderung des Projektes ist, daß das zu erstellende Informationssystem datenbankbasiert sein soll. Hier liegt die Verwendung eines Datenbank-Management-Systems (DBMS) zur Verwaltung der Fragmente nahe, da ein solches System den Mehrbenutzerbetrieb sowie eine transaktionsorientierte, konsistente Datenhaltung ermöglicht und außerdem eine Anfragesprache bereitstellt. Auf diesem DBMS ist eine Datenbank zu implementieren, die im ZYX-Modell beschriebene Dokumente und deren Fragmente verwalten kann.

Im Rahmen dieser Arbeit wurde die Implementierung der Datenbank auf dem objektrelationalen DBMS Informix Dynamic Server/ Universal Data Option (IDS/UD) durchgeführt. Bei der Implementierung von Datenbanken können im wesentlichen drei Entwurfsschritte identifiziert werden [STS97].

Der erste Schritt ist der *konzeptionelle Entwurf*. In diesem Schritt soll die Datenbank unabhängig vom später verwendeten DBMS entworfen werden. Dieses geschieht in der Regel durch ER-Diagramme oder ähnliche Modellierungsmittel. In dieser Arbeit wird in diesem Schritt ein Klassendiagramm erstellt. Der zweite Entwurfsschritt ist der *logische Entwurf*. Hier werden der konzeptionelle Entwurf auf das Datenmodell des Ziel-DBMS abgebildet und eventuell Optimierungen vorgenommen. Am Ende dieses Schrittes steht das Datenbankschema. Der dritte Schritt ist die *Datendefinition*. In diesem Schritt wird das Datenbankschema in die Datendefinitionssprache des Ziel-DBMS übersetzt. Saake [STS97] nennt noch zwei weitere Schritte, den physischen Entwurf sowie die Installation und Wartung, die in dieser Arbeit jedoch nicht näher behandelt werden.

Jedem Entwurfsschritt ist in diesem Kapitel ein Abschnitt gewidmet. Zusätzlich wird zwischen dem konzeptionellen und logischen Entwurf ein Abschnitt über das Datenmodell des IDS/UD eingeschoben. Schließlich wird in einem weiteren Abschnitt die implementierte Funktionalität auf dem IDS/UD behandelt.

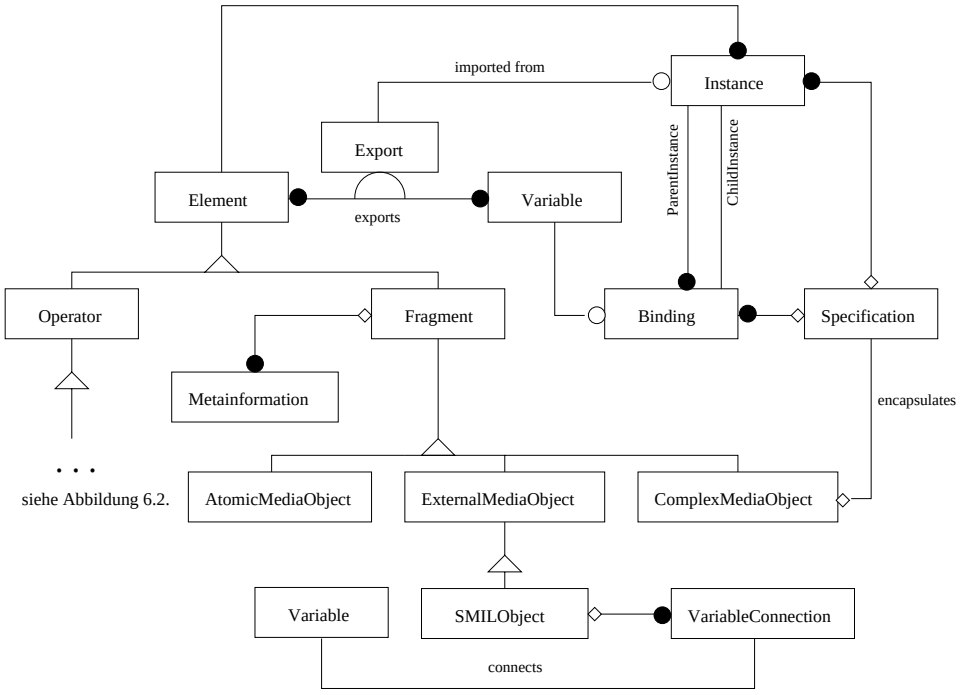


Abbildung 6.1: Zentrale Klassen des konzeptionellen Entwurfs in OMT-Notation

## 6.1 Konzeptioneller Entwurf

In diesem Entwurfsschritt wird ein objektorientierter Entwurf des ZYX-Dokumentmodells unabhängig vom verwendeten Ziel-DBMS durchgeführt. Dieser Entwurf liefert ein Klassendiagramm, welches Dokumente im ZYX-Modell beschreiben kann. Dabei wird im wesentlichen entlang den Definitionen aus Kapitel 5 vorgegangen. In der Tat läßt sich diese Hierarchie nahezu identisch auf ein Klassendiagramm abbilden. Dennoch sind einige Entwurfsentscheidungen zu treffen, die in diesem Abschnitt näher erläutert werden sollen.

Abbildung 6.1 zeigt die wesentlichen Klassen des objektorientierten Entwurfs des ZYX-Dokumentmodells . Diese Klassen werden im folgenden beschrieben, ohne jedoch auf jedes einzelne Attribut einzugehen. Sämtliche Attribute sind aber in Anhang A aufgeführt.

### 6.1.1 Element

Die Klasse `Element` ist die zentrale Klasse des konzeptionellen Entwurfs. Sie bildet eine abstrakte Basisklasse für alle Präsentationselemente des ZYX-Modells. Sie definiert im wesentlichen ein Attribut namens `ID` zur eindeutigen Identifizierung eines solchen Präsentationselementes.

Wie aus Kapitel 5 bekannt, ist einem Präsentationselement $e$ eine Menge an Variablen $V_e$ bzw. ein Menge an Projektionsvariablen $PV_e$ zugeordnet. Dies wird im Entwurf über die Beziehung `exports` modelliert. Es ist möglich, daß eine Variable mehreren Präsentationselementen zugeordnet ist. Das ist beispielsweise dann der Fall, wenn ein komplexes Medienobjekt einen Spezifikationsbaum kapselt, in dem

eine Variable eines Operatorelementes ungebunden ist. Das komplexe Medienobjekt exportiert diese Variable und somit ist dieselbe Variable sowohl dem Operatorelement als auch dem komplexen Medienobjekt zugeordnet. Das hat Auswirkungen auf den Entwurf. Zum einen rechtfertigt es die Modellierung von Variablen in einer eigenständigen Klasse `Variable`. Zum andern ist die Kardinalität der Assoziation `exports` als $(m : n)$ zu definieren.

Die Assoziationsklasse `Export` hat die Aufgabe, die einem Präsentationselement zugeordnenten Variablen zu ordnen, um sie einfach adressieren zu können. Dazu wird in der Klasse das Attribut `No` definiert. Betrachtet man zum Beispiel das Operatorelement $seq(v_1, v_2, v_3, v_4)$, so definiert dieses vier Variablen. Der dieses Operatorelement repräsentierenden Instanz der Klasse `Element` sind somit über die Assoziation `exports` vier Instanzen der Klasse `Variable` zugeordnet. Um die Variablen des Präsentationselements zu ordnen, erfolgt jede Zuordnung über eine Instanz der Klasse `Export`. Der Wert des Attributs `No` der Zuordnung von $v_1$ ist 1, der Wert des Attributs `No` der Zuordnung von $v_2$ ist 2, etc.

Die Klasse `Variable` bietet neben einem Attribut, das zwischen einer normalen und einer Projektionsvariable unterscheidet, das Attribut `ID` zur eindeutigen Identifikation von Variablen.

### 6.1.2 Specification

Die Klasse `Specification` repräsentiert die Spezifikationsbäume des ZYX-Dokumentmodells . Die von einem Spezifikationsbaum verwalteten Mengen von Präsentationselementen und Bindungen wird durch die Aggregation der Klassen `Instance` und `Binding` modelliert.

Betrachtet man zunächst die Menge der in einem Spezifikationsbaum enthaltenen Präsentationselemente, so stellt man fest, daß eine direkte Aggregation der Klasse `Element` von `Specification` nicht ausreichend ist. Es muß möglich sein, dasselbe Präsentationselement (beispielsweise ein Video) mehrmals innerhalb eines Spezifikationsbaumes zu verwenden und dessen Verwendungen eindeutig voneinander zu unterscheiden. Diese Überlegung führt zur Definition der Klasse `Instance`. Ein Objekt der Klasse `Instance` stellt ein tatsächliches Auftreten eines Präsentationselements innerhalb eines Spezifikationsbaumes dar. Von einem Präsentationselement können beliebig viele Instanzen definiert werden. Die Assoziation `imported from` zeigt bei komplexen Medienobjekten an, von welcher Instanz eines Präsentationselementes des gekapselten Spezifikationsbaumes eine exportierte Variable des komplexen Medienobjekts stammt.

Eine Bindung wird durch ein Objekt der Klasse `Binding` repräsentiert. Es stellt über die an der Klasse beteiligten Assoziationen eine Verbindung von einer Variable einer Instanz eines Präsentationselementes zu der Instanz eines anderen Präsentationselementes her und definiert damit eine Kante des Spezifikationsbaumes.

### 6.1.3 Operator

Das ZYX-Dokumentmodell unterteilt Präsentationselemente in Operatorelemente und Fragmente. Dieses spiegelt sich auch in der Klassenhierarchie des konzeptionellen Entwurfs wieder. Für jede der zwei Gruppen von Präsentationselementen wird eine weitere abstrakte Basisklasse zusaätzlich zu `Element` definiert. Die Klas-

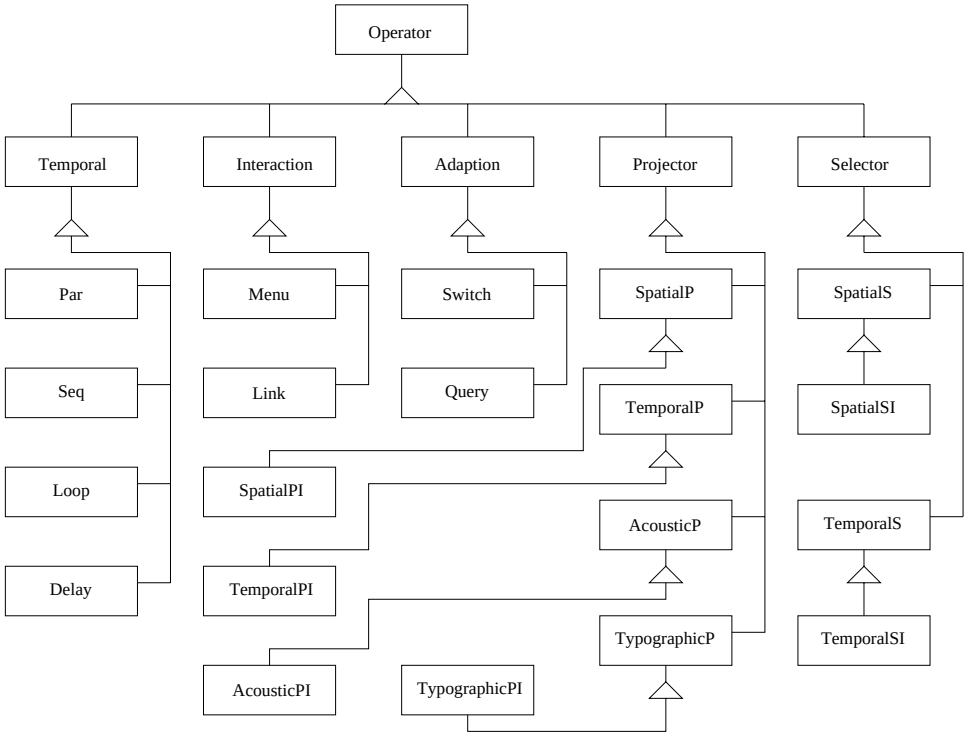


Abbildung 6.2: Subklassen von `Operator` in OMT-Notation

se `Operator` bildet die abstrakte Basisklasse für die Operatorelemente des ZYX-Modells.

Die Definitionshierarchie der Operatorelemente aus Kapitel 5 läßt sich direkt in eine Klassenhierachie überführen (siehe Abbildung 6.2). Eine Klasse, die ein Operatorelement modellieren soll, muß lediglich direkt oder indirekt von `Operator` erben und Attribute definieren, welche die Parameter des Operatorelements repräsentieren. Zum Beispiel definiert die das *par*-Operatorelement modellierende Klasse `Par` die Attribute `LipSync` und `Finish`.

Um trotz der relativ großen Anzahl an Operatorelementen diesen Teil der Klassenhierarchie übersichtlich zu gestalten, wurden die abstrakten Klassen `Temporal`, `Interaction`, `Adaption`, `Projector` und `Selector` eingeführt. Diese spalten die Klassenhierarchie weiter auf, ohne jedoch neue Attribute gegenüber der Basisklasse `Operator` zu definieren.

Aufgrund der Einfachheit der Abbildung der Operatorelemente auf Klassen des konzeptionellen Entwurfs soll hier nicht weiter auf diesen Teilbaum der Klassenhierarchie eingegangen werden. Anhang A zeigt alle Subklassen von `Operator` zusammen mit ihren Attributen.

### 6.1.4 Fragment

So wie die Klasse `Operator` die abstrakte Basisklasse für die Operatorelemente des ZYX-Dokumentmodells darstellt, bildet `Fragment` die abstrakte Basisklasse für die Fragmente des Dokumentmodells. Einem Fragment kann eine Menge von Metainformationen zugeordnet sein, die zur seiner Klassifizierung und der Beschreibung

seines Inhalts dient. Metainformationen werden über die Klasse `Metainformation` definiert. Eine Instanz dieser Klasse kapselt ein (*key*, *value*)-Tupel. Eine Instanz von `Fragment` aggregiert beliebig viele Instanzen von `Metainformation`. Außerdem definiert die Klasse `Fragment` ein Attribut, welches zur Aufnahme einer textuellen Beschreibung des Fragmentes dient.

Die erste Art von Fragmenten, die atomaren Medienobjekte, wird durch die Klasse `AtomicMediaObject` repräsentiert. Diese Klasse definiert ein Attribut `Url` zur Aufnahme einer textuellen Referenz auf den Speicherungsort der von dem atomaren Medienobjekt gekapselten Mediendaten. Weil diese Referenz als String kodiert ist, macht es für die Implementierung der Datenbank keinen Unterschied, ob die Mediendaten in einem Filesystem, in einer Datenbank, auf einem Webserver oder woanders gehalten werden. Die Attribute `Type` und `Format` spezifizieren die Art und das Ablageformat der Mediendaten.

Die zweite Art von Fragmenten sind die komplexen Medienobjekte, repräsentiert durch die Klasse `ComplexMediaObject`. Ein Objekt dieser Klasse kapselt exakt einen Spezifikationsbaum, modelliert durch die Aggregation von `Specification`.

Die letze Art von Fragmenten bilden die externen Medienobjekte, die dazu dienen, Spezifikationen von multimedialen Dokumenten externer Dokumentstandards zu kapseln. Hierzu definiert die Klasse `ExternalMediaObject` eine abstrakte Basisklasse. Sie verfügt über die Attribute `Format` und `Url`, um Dokumentstandard und Speicherungsort der gekapselten Spezifikation zu beschreiben. Für die Unterstützung eines fremden Dokumentmodells muß lediglich eine Klasse definiert werden, die von `ExternalMediaObject` abgeleitet ist. In dieser Arbeit ist dies für den SMIL-Dokumentstandard geschehen. Die Klasse `SMILMediaObject` kapselt die Spezifikation eines SMIL-Dokuments. Wie in Kapitel 5 erläutert, ist es möglich, Link-Ziele eines SMIL-Links an eine Variable des ZYX-Dokumentmodells zu koppeln. Dieses wird über die Aggregation der Klasse `VariableConnection` realisiert. Diese Klasse verfügt über ein Attribut, welches das zu verbindende SMIL-Link-Ziel beschreibt. Weiterhin existiert eine Assoziation zwischen `Variable` und `VariableConnection`, welche die Kopplung des Link-Ziels an eine Variable modelliert.

## 6.2 Der Informix Dynamic Server/ Universal Data Option

Der Informix Dynamic Server/ Universal Data Option ist ein sogenanntes *objektrelationales Datenbank-Management-System*. Objektrelationale DBMS erweitern das Datenmodell herkömmlicher relationaler Systeme um objektorientierte Konzepte [STS97]. Allerdings gibt es sowohl in der Forschung als auch in der Wirtschaft keine Einigkeit darüber, welche Konzepte der Objektorientierung unterstützt werden müssen, so daß ein relationales DBMS als objektrelational zu bezeichnen ist. Deshalb soll in diesem Abschnitt beschrieben werden, wie der Begriff des objektrelationalen DBMS im Sinne des IDS/UD verstanden wird.

Der Informix Dynamic Server/ Universal Data Option ist ein relationales DBMS, welches im wesentlichen um die Konzepte der *erweiterten Datentypen*, der *Tabellenhierarchien*, der *serverbasierten Routinen* und der *DataBlade-Moduln* ergänzt wurde [Inf97b]. Auf diese Konzepte wird im folgenden näher eingegangen.

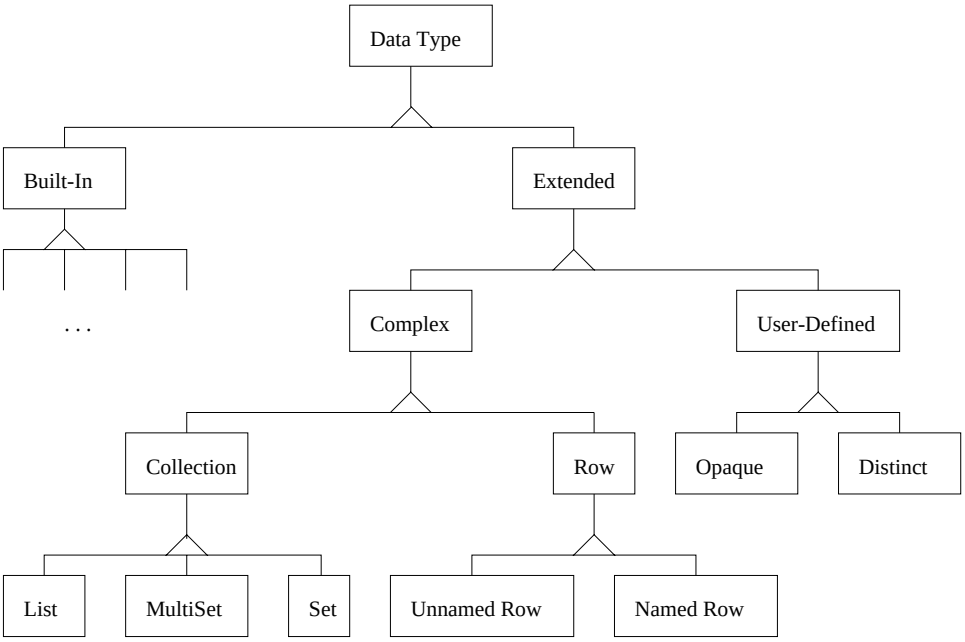


Abbildung 6.3: Typhierarchie des IDS/UD

### 6.2.1 Erweiterte Datentypen

Die Typhierarchie des IDS/UD (siehe Abbildung 6.3) unterscheidet zwischen den her-kömmlichen, *eingebauten Datentypen* – wie zum Beispiel `INTEGER`, `BOOLEAN` und `VARCHAR` – und den sogenannten *erweiterten Datentypen*. Die erweiterten Datentypen sind grundsätzlich als gleichwertig zu den eingebauten Datentypen anzusehen: sie können ebenso zur Typisierung von Spalten einer Datenbanktabelle dienen und Teil der Signatur einer serverbasierten Routine sein. Die verschiedenen Arten der erweiterten Datentypen sollen nun kurz vorgestellt werden.

#### Collection Types

*Collections* sammeln mehrere Elemente eines Typs in einer Gruppe. Der Typ der Elemente unterliegt hierbei keinen Einschränkungen: sowohl eingebaute Typen als auch erweiterte Datentypen sind gültige Typen für Elemente.

Der IDS/UD definiert drei Collection Types, die sich darin unterscheiden, ob ihre Elemente geordnet sind und ob Duplikate eines Elementes erlaubt sind. Tabelle 6.1 zeigt die verschiedenen Arten der Collection Types.

| Collection Type | Duplikate | Ordnung |
|---|---|---|
| Set | nein | nein |
| Multiset | ja | nein |
| List | ja | ja |

Tabelle 6.1: Collection Types des IDS/UD

```
1     create row type Person
2     (
3             ID        integer not null,
4             Name      varchar(20) not null,
5             Anschrift Adresse  -- Adresse ist ein weiterer Named Row Type.
6     );
```

Abbildung 6.4: Beispiel zur Erzeugung eines Named Row Types

### Row Types

Ein *Row Type* faßt eine Menge von Datenfeldern zu einem Tupel zusammen. Ein solches Tupel entspricht dem Konzept eines Records in einer Programmiersprache. Ein Datenfeld eines Row Type kann von einem beliebigen Datentyp sein, auch von einem weiteren Row Type. Jedoch ist eine rekursive Verschachtelung von Row Types nicht möglich, da der IDS/UD keine Referenzen kennt.

Es gibt zwei Arten von Row Types, nämlich die *Named Row Types* und die *Unnamed Row Types*. Ein Named Row Type verfügt über einen eindeutigen Namen und ist unter diesem systemweit bekannt, während eine Unnamed Row ein namenloses, ad-hoc erzeugtes Tupel ist. Ein Resultattupel einer Datenbank-Query kann man beispielsweise als Unnamed Row betrachten. Abbildung 6.4 zeigt ein einfaches Beispiel einer SQL-Anweisung, die einen Named Row Type `Person` mit den Datenfeldern `ID`, `Name` und `Anschrift` erzeugt.

Named Row Types können zueinander in eine Vererbungsbeziehung gesetzt werden. Dabei erbt der Subtyp sämtliche Attribute des Supertyps. Außerdem kann der Subtyp zusätzliche Attribute definieren. Eine Instanz des Subtyps kann überall dort verwendet werden, wo eine Instanz seines Supertyps erwartet wird. Vererbungsbeziehungen dieser Art nennt man auch *intensionale Spezialisierung* [STS97].

### Opaque Types

Ein *Opaque Type* ist ein fundamentaler Datentyp, der einen atomaren Wert aufnehmen kann. Jedoch kann dieser Wert vom DBMS weder eingesehen, in Komponenten zerlegt, noch interpretiert werden.

Ein Opaque Type wird über eine C-Struktur implementiert. Damit ein sinnvoller Umgang mit den Instanzen einer solchen C-Struktur möglich ist, müssen serverbasierte Zugriffsroutinen implementiert werden. Typische Einsatzgebiete für solche Routinen sind Erzeugung von Instanzen eines Opaque Type, Zugriff auf einzelne Felder der C-Struktur sowie der Vergleich zwischen Instanzen eines Opaque Type.

Die definierten Routinen können in Datenbankabfragen verwendet werden. Das Beispiel in Abbildung 6.5 ermittelt aus einer Tabelle, die eine Spalte `Konto` vom Opaque Type `Bankkonto` enthält, die Liste der Kontonummern und Kontostände.

### Distinct Types

Ein *Distinct Type* ist prinzipiell ein Alias für einen Quelldatentyp und hat dieselbe Speicherrepräsentation wie dieser. Der Quelldatentyp darf sowohl ein eingebauter Datentyp, ein Collection Type, ein Row Type, ein Opaque Type als auch ein anderer Distinct Type sein.

```
1     create table Kontos
2     (
3        Konto Bankkonto not null
4     );
5
6     select nummer(Konto), kontostand(Konto)
7     from Kontos;
```

Abbildung 6.5: Beispiel zur Verwendung eines Opaque Types

Ein Distinct Type ist jedoch von seinem Quelltyp eindeutig zu unterscheiden. Er kann nicht durch den Quelldatentyp substituiert werden und umgekehrt ebenfalls nicht als Substitut für den Quelldatentyp dienen. Möchte man Funktionen des Quelldatentyps für den Distinct Type nutzen, so ist ein expliziter Type Cast des Distinct Type auf den Quelldatentyp nötig.

Als Beispiel für die Definition eines Distinct Type möge eine Liste des Opaque Type `Bankkonto` aus dem obigen Beispiel dienen (s. Abbildung 6.6).

```
1     create distinct type KontoList as List(Bankkonto not null);
```

Abbildung 6.6: Beispiel zur Erzeugung eines Distinct Types

### 6.2.2 Tabellenhierarchien

Der IDS/UD erlaubt es, Named Row Types direkt auf Tabellen abzubilden. Eine solche Tabelle, auch *Typed Table* genannt, enthält exakt dieselben Datenfelder wie der zugehörige Named Row Type. So erzeugt die SQL-Anweisung `create table Persons of type Person` bezogen auf das Beispiel aus Abbildung 6.4 einen Typed Table mit Namen `Persons` und den Spalten `ID`, `Name` und `Anschrift`.

Wie schon erwähnt, können Named Row Types zueinander in einer Vererbungsbeziehnung stehen. Parallel zu einer solchen intensionalen Spezialisierungshierarchie ist es möglich, eine Hierarchie von Typed Tables aufzubauen (siehe Abbildung 6.7).

Die Auswirkungen einer Tabellenhierarchie sind folgende: Eine Select-Anfrage auf einen Typed Table $t$ innerhalb dieser Hierarchie untersucht automatisch auch alle Tupel derjenigen Typed Tables (auf die Spalten des $t$ zugeordneten Named Row Type reduziert), die unterhalb von $t$ in der Tabellenhierarchie stehen. Ebenso wirken sich Update- und Delete-Anweisungen auf die unter $t$ in der Hierarchie stehenden Typed Tables aus. Eine Insert-Anweisungen hingegen bezieht sich nur auf die angegebene Tabelle.

Eine Tabellenhierarchie entspricht somit einer *extensionalen Spezialisierungshierarchie* [STS97]. Da diese parallel zu der intensionalen Hierarchie der zugehörigen Named Row Types aufgebaut ist, unterstützt der IDS/UD den Aufbau von kombinierten intensionalen und extensionalen Spezialisierungshierarchien.

### 6.2.3 Serverbasierte Routinen

IDS/UD erlaubt die Ausführung von benutzerdefinierten Routinen auf dem Datenbankserver. Zum Zeitpunkt der Erstellung dieser Arbeit gibt es zwei Möglichkeiten,

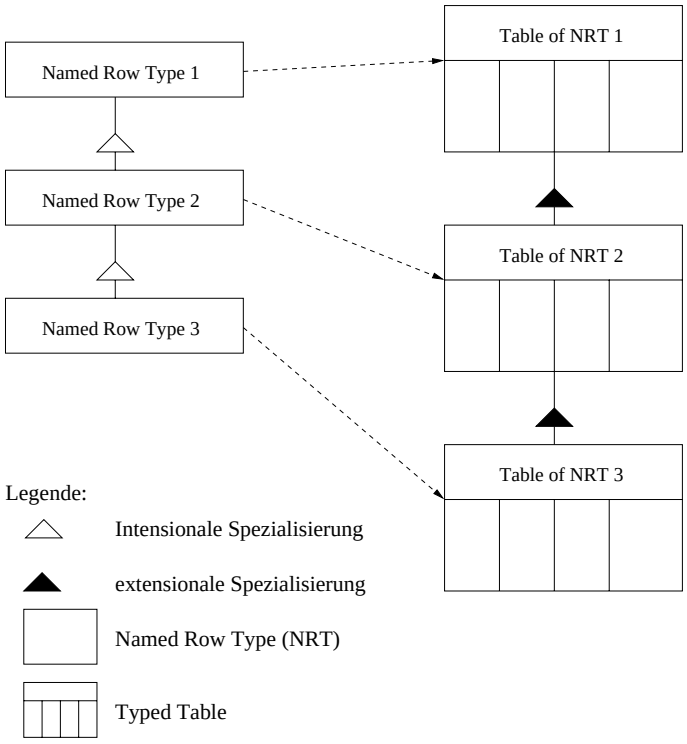


Abbildung 6.7: Hierarchie von Typed Tables

solche *serverbasierte Routinen* zu entwickeln.

Die erste Möglichkeit ist die Verwendung der Informix-spezifischen *Stored Procedure Language* (SPL), die eine Erweiterung von SQL zu einer imperativen Programmiersprache darstellt. Eine in SPL geschriebene Routine wird in ein Zwischenformat übersetzt und auf dem Datenbankserver abgelegt. Wird eine in SPL geschriebene Routine aufgerufen, so sucht der Server die Routine und führt sie aus, indem deren Anweisungen im Zwischenformat interpretiert werden. Die Verwendung von SPL hat den Vorteil, daß IDS/UD Optimierungen am Programmcode und den darin enthaltenen SQL-Anfragen bereits zur Übersetzungszeit vornehmen kann. Nachteilig ist jedoch die Tatsache, daß SPL eine recht primitive Sprache ist, so daß sie sich schlecht zur Entwicklung komplexer Routinen eignet.

Die andere Möglichkeit ist die Implementierung der Routinen in C mit Hilfe des *DataBlade API* [Inf97a]. Die Entwicklung der Routinen in C stellt die einzige Möglichkeit dar, Zugriffsroutinen für Opaque Types zu erstellen. Der C-Code wird mit einem gewöhnlichen C-Compiler übersetzt und zu einer Shared Library gebunden. Der Speicherungsort der Shared Library wird dem IDS/UD bekanntgegeben. Wird eine in C implementierte Routine aufgerufen, so wird die Shared Library, in der die Routine gespeichert ist, in den Adreßraum des Datenbankservers eingeblendet und die Routine ausgeführt. Weil die Routine im Adreßraum des Servers abläuft, wird eine hohe Ausführungsgeschwindigkeit der Routine erreicht. Allerdings besteht die Gefahr, durch fehlerhafte Programmierung den ganzen Datenbankserver lahmzulegen. Ein weiterer Nachteil der Entwicklung serverbasierter Routinen mit Hilfe des DataBlade API besteht darin, daß das Abstraktionsniveau dieser API relativ gering ist. Eine C-Routine ist deswegen um ein vielfaches länger als eine äquivalente SPL-Routine.

Informix plant ein C++-DataBlade-API und ein Java-DataBlade-API zur Entwicklung serverbasierter Routinen, um das Problem des geringen Abstraktionsniveaus des C-DataBlade-API zu beheben.

### 6.2.4 DataBlade-Moduln

Der Sinn und Zweck der sogenannten *DataBlade-Moduln* besteht darin, zusammengehörige Datenbankelemente zu Auslieferungseinheiten zu bündeln. Eine solche Auslieferungseinheit läßt sich einfach während des Betriebs des Datenbankservers in einer Datenbank installieren (registrieren) und auch wieder entfernen (deregistrieren). Zu den Datenbankelementen, die zu einem DataBlade-Modul zusammengefaßt werden können, zählen u.a. erweiterte Datentypen, serverbasierte Routinen (C und SPL), Tabellen, Indizes, Fehlercodes und Fehlermeldungen.

Zum Zweck der Erstellung, Registration und Deregistration von DataBlade-Moduln stellt Informix einige nützliche Werkzeuge zur Verfügung. Der *BladeSmith* erlaubt die einfache Definition der Bestandteile eines DataBlade-Moduls (wie zum Beispiel Row Types, Opaque Types und serverbasierte Routinen) mit Hilfe einer GUI. Das Werkzeug generiert aus diesen Definitionen automatisch die zum DataBlade gehörenden Installationskripte, sowie die Funktionsköpfe derjenigen serverbasierten Routinen, die in C implementiert werden. Der *BladeManager* erlaubt die einfache Registration und Deregistration von DataBlades in Datenbanken mit einer graphischen Benutzeroberfläche.

Aufgrund der Vorteile des DataBlade-Konzeptes sowie der vorhandenen Tool-Unterstützung erfolgt auch die Implementierung der Datenbank für das ZYX-Dokumentmodell in Form eines DataBlades.

## 6.3 Logischer Entwurf

Im bisherigen Verlauf dieses Kapitels wurde ein konzeptioneller Entwurf der Datenbank für ZYX-Dokumente durchgeführt und eine Einführung in diejenigen Datenmodellierungskonzepte des IDS/UD gegeben, die ihn von herkömmlichen relationalen DBMS unterscheiden. In diesem Abschnitt werden die Klassen und Assoziationen des konzeptionellen Entwurfs unter Verwendung der vorgestellten Datenmodellierungskonzepte in ein Datenbankschema überführt.

### 6.3.1 Abbildung der Klassen

Es sind Opaque Types und Named Row Types vorgestellt worden. Ein Opaque Type kapselt eine C-Struktur, während ein Named Row Type ein Tupel von Datenfeldern definiert. Beide Typen sind geeignet, eine Klasse des konzeptionellen Entwurfs zu modellieren. Es gilt nun, zwischen diesen beiden Modellierungsmitteln abzuwägen und eine Wahl zu treffen.

Ein Problem bei der Verwendung von Opaque Types ist die sehr strenge Kapselung der C-Struktur. Der IDS/UD kann die C-Struktur weder interpretieren noch direkt auf deren Felder zugreifen. Um den Zugriff zu ermöglichen, sind Zugriffsroutinen für die Felder der Struktur zu implementieren. Dieses bringt neben einem

hohen Implementierungsaufwand Probleme bei der Ausführung von Datenbankanfragen mit sich. Für jeden Zugriff auf ein Feld der C-Struktur muß der Server eine Routine aufrufen und ausführen. Im Gegensatz dazu kann der Server auf die Datenfelder eines Named Row Type direkt zugreifen. Sind Abfragen auf einzelne Attribute von Klassen häufig, so ist die Modellierung über Opaque Types ungünstig. Solche Abfragen sind aber für die zu implementierende Dokumentdatenbank zu erwarten, beispielsweise: „Suche alle *par*-Operatorelemente, die eine lippensynchrone Ausführung definieren.".

Problematisch ist außerdem die relativ komplexe Klassenhierarchie des konzeptionellen Entwurfs (siehe Abbildungen 6.1 und 6.2). Da Opaque Types nicht in eine Vererbungsbeziehung gesetzt werden können, erweist sich eine Abbildung dieser Klassenhierarchie auf Opaque Types als relativ aufwendig. Named Row Types hingegen können in intensionalen Spezialisierungsbeziehungen zueinander stehen, so daß die Klassenhierarchie eins zu eins auf Named Row Types abgebildet werden kann. Außerdem kann jedem Named Row Type ein Typed Table zugeordnet werden, der die Instanzen dieses Named Row Type verwaltet. Da die Typed Tables parallel zur intensionalen Spezialisierungshierarchie der Named Row Types eine Tabellenhierarchie bilden, sind sie zur Modellierung der extensionalen Spezialisierungssbeziehungen zwischen den Klassen des konzeptionellen Entwurfs geeignet.

Aus diesen Gründen ist die Verwendung von Named Row Types besser zur Abbildung der Klassen des konzeptionellen Entwurfs geeignet als die Verwendung von Opaque Types. Jede Klasse wird durch einen gleichnamigen Named Row Type modelliert[1]. Die Attribute der Klassen werden eins zu eins auf Datenfelder der Named Row Type projeziert. Anhang B zeigt sämtliche Named Row Types des logischen Entwurfs.

Aus der Verwendung von Named Row Types zur Klassenmodellierung ergibt sich die Frage, ob zu jedem Named Row Type ein Typed Table angelegt muß. In der Tat ist dieses nicht plausibel, da es im konzeptionellen Entwurf Klassen gibt, die nur in Verbindung mit anderen Klassen existieren können und für sich betrachtet wenig Sinn machen. Ein Beispiel hierfür ist die Klasse `Metainformation`. Ein $(key, value)$-Tupel für sich betrachtet ist vollkommen bedeutungslos. Ein solches Tupel wird erst aussagekräftig, wenn es einem Fragment zugeordnet wird. Weitere Klassen dieser Art sind `VariableConnection`, die Assoziationsklasse `Export` sowie die Klasse `Binding`. Den diese Klassen modellierenden Named Row Types werden keine Typed Tables zugeordnet. Stattdessen werden Instanzen dieser Named Row Types in Collections bei denjenigen Named Row Types gehalten, von denen sie abhängig sind. Im Fall von `Metainformation` ergibt sich für den Named Row Type `Fragment` ein zusätzliches Datenfeld des Typs `set(Metainformation not null)`.

Allen anderen Named Row Types wird jeweils ein Typed Table zugeordnet. Die Named Row Types `Element` und `Instance` definieren ein zusätzliches Datenfeld `Class`, welches den Namen des Typed Table enthält, in dem das Präsentationselement bzw. das zur Instanz gehörige Präsentationselement gespeichert ist.

### 6.3.2 Abbildung der Assoziationen

Während bisher die Klassen des konzeptionellen Entwurfs auf Named Row Types und Typed Tables abgebildet wurden, geht es in nun um die Modellierung der As-

[1] In ein paar Fällen ist eine leichte Abänderung der Namen nötig, da der IDS/UD die Länge eines Namens auf 14 Zeichen begrenzt.

soziationen zwischen den Klassen. Da der IDS/UD keine direkte Unterstützung von Assoziationen anbietet, müssen diese über Datenfelder in Named Row Types bzw. durch zusätzliche Tabellen simuliert werden. Abbildung 6.8 zeigt diejenigen Named Row Types mit ihren Datenfeldern, die den Kern des logischen Entwurfs ausmachen. Assoziationen, die über Datenfelder modelliert werden, sind als gestrichelte Pfeile markiert.

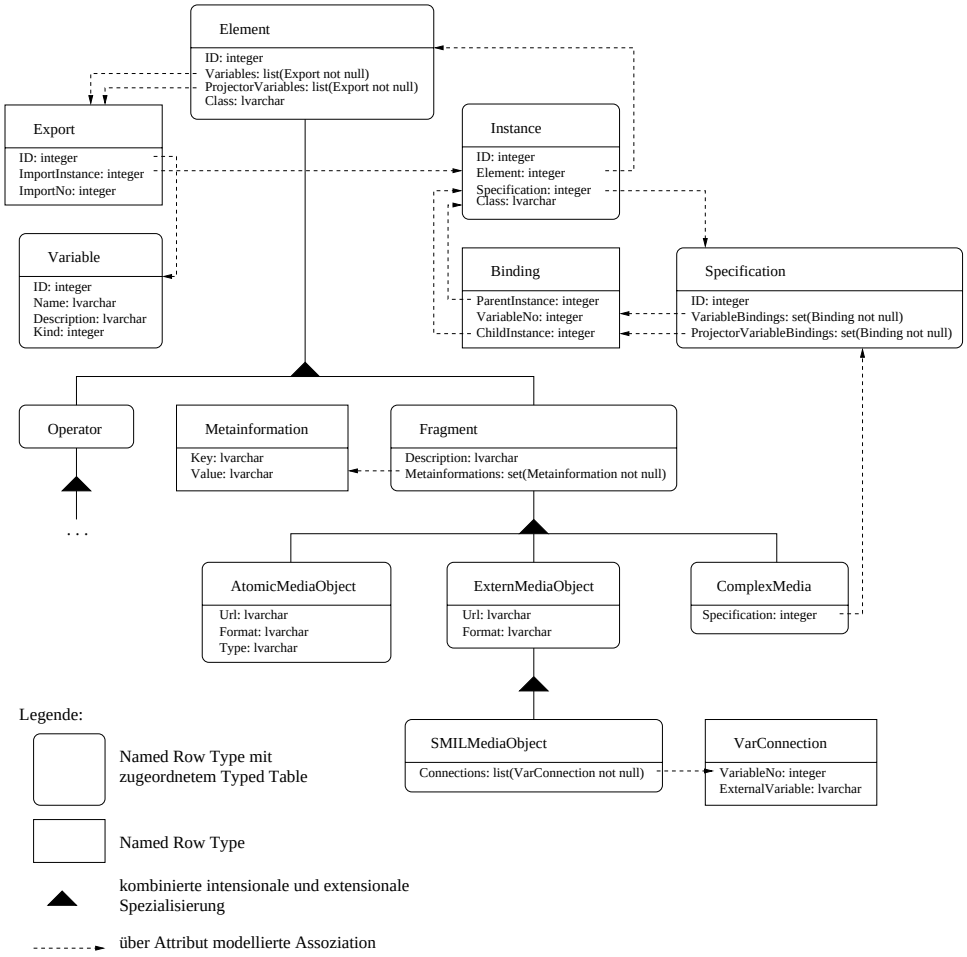


Abbildung 6.8: Kern des logischen Entwurfs

Die erste Assoziation des konzeptionellen Entwurfs (siehe Abbildung 6.1), die in den logischen Entwurf übernommen werden soll, ist die Assoziation `exports`. Sie verbindet die Klassen `Element` und `Variable` mit der Kardinalität $(m : n)$. In einem relationalen DBMS wäre eine gängige Vorgehensweise, `exports` über eine eigene Tabelle zu modellieren, die über zwei Fremdschlüssel verfügt, nämlich über die Werte des Attributs `ID` von `Element` und `Variable`. In diesem Fall wird aber anders vorgegangen. In der Regel dürfte die Assoziation nur in eine Richtung traversiert werden, nämlich von `Element` zu `Variable`. Eine Variable kann, wie bereits beschrieben, mehreren Präsentationselementen zugeordnet sein. Man kann somit nicht von einer Variable ausgehend ein eindeutiges, zugehöriges Präsentationselement bestimmen, was am Sinn einer solchen Traversierung zweifeln läßt. Umgekehrt ist es aber sehr sinnvoll, die zugeordneten Variablen eines Elementes zu bestimmen.

Deswegen wird wie folgt verfahren: Die Assoziationsklasse `Export` wird durch einen

gleichnamigen Named Row Type modelliert. Dieser verfügt über das Datenfeld `ID`, welches die ID der exportierten Variable aufnimmt. Die Assoziation `exports` wird über das Datenfeld `Variables` des Named Row Type `Element` modelliert, das den Typ `list(Export not null)` hat. Es wird eine Liste[2] gewählt, weil eine Ordnung auf der Assoziation zur eindeutigen Adressierung der Variablen eines Präsentationselementes gewünscht ist. Aus Effizienzgründen wird die Verwaltung der Projektionsvariablen eines Präsentationselements von der Verwaltung der normalen Variablen getrennt. Es wird also dem Row Type `Element` ein weiteres Datenfeld namens `ProjectorVariables` ebenfalls vom Typ `list(Export not null)` hinzugefügt.

Der Row Type `Export` definiert außerdem die Datenfelder `ImportInstance` und `ImportNo`, welche im Falle von komplexen Medienobjekten angeben, von welcher Instanz eines Präsentationselementes $e$ die exportierte Variable stammt und welche Position diese Variable in der Liste der exportierten Variablen von $e$ einnimmt. Dieses modelliert die Assoziation `imported from` zwischen den Klassen `Export` und `Instance`.

Die nächste Assoziation, die betrachtet werden soll, ist die $(1:n)$-Assoziation zwischen den Klassen `Element` und `Instance`. Diese wird, wie in relationalen DBMS üblich, über ein zusatzliches Datenfeld beim Row Type `Instance` realisiert. Dieses Datenfeld hat die Funktion eines Fremdschlüssels, trägt den Namen `Element` und nimmt den Wert des Attributs `ID` desjenigen Präsentationselements auf, zu dem eine Instanz des Row Types `Instance` gehört. Analog wird die $(1:1)$-Aggregation zwischen den Klassen `ComplexMediaObject` und `Specification` durch die Hinzunahme eines Fremdschlüssels namens `Specification` zum Named Row Type `ComplexMedia` modelliert.

Als nächstes sollen die Aggregation von `Instance` und die Aggregation von `Binding` der Klasse `Specification` untersucht werden. Wie schon erwähnt, ist `Binding` keine eigenständige Klasse sondern nur im Zusammenhang mit dem zugehörigen Spezifikationsbaum sinnvoll. Deswegen wird dem Named Row Type `Binding` kein eigener Typed Table zugeordnet. Stattdessen werden die Instanzen von `Binding` direkt mit Hilfe einer Collection beim Row Type `Specification` gehalten. Folglich definiert Binding die Datenfelder `VariableBindings` und `ProjectorBindings`[3], die vom Typ `set(Binding not null)` sind. Der Row Type `Binding` selbst verbindet eine Variable einer Elterninstanz mit einer Kindinstanz über die Datenfelder `ParentInstance`, `VariableNo` und `ChildInstance`, wobei `VariableNo` die Position der zu bindenden Variable in der Liste der exportierten Variablen des zur Elterninstanz gehörenden Präsentationselements angibt.

Die Aggregation der Klasse `Instance` hingegen wird anders modelliert. Es nämlich nicht nur interessant, festzustellen, welche Instanzen eines Präsentationselementes in einem Spezifikationsbaum vorkommen, sondern es ist auch von Bedeutung, über wieviele und welche Instanzen ein Präsentationselement verfügt. Das Löschen eines Präsentationselementes sollte beispielsweise nicht möglich sein, falls es noch in Spezifikationsbäumen verwendet wird. Dies ist genau dann der Fall, wenn Instanzen des Präsentationselements existieren. Somit kann `Instance` nicht eindeutig der Klasse `Specification` zugeordnet werden, sondern ist als eigenständige Klasse zu betrachten. Darum wird dem Row Type `Instance` ein Fremdschlüsseldatenfeld namens `Specification` zugeordnet, das angibt, in welchem Spezifikationsbaum die Instanz eines Präsentationselementes Verwendung findet.

[2] Diese Liste wird im folgenden als Liste der exportierten Variablen bezeichnet.

[3] Aus Effizienzgründen erfolgt wiederum die Trennung der Verwaltung der Projektionsvariablen von der Verwaltung der normalen Variablen

Schließlich verbleibt noch die Modellierung der Aggregationen der nicht eigenständigen Klassen `Metainformation` und `VariableConnections`. Auch sie werden direkt mit Hilfe von Collections bei den zugehörigen Row Types `Fragment` bzw. `SMILMediaObject` gehalten. Dazu definieren der Row Type `Fragment` ein Datenfeld namens `Metainformations` des Typs `set(Metainformation not null)` und der Row Type `SMILMediaObject` das Datenfeld `Connections`, welches den Typ `list(VarConnection not null)` zugewiesen bekommt.

## 6.4 Datendefinition

Als Ergebnis des logischen Entwurfs zeigt Anhang B das komplette Schema der Datenbank, welche die Dokumente des ZYX-Modell fassen soll. Es gilt nun, ein DataBlade-Modul zu erzeugen, welches dieses Schema bei der Registrierung in einer Datenbank anlegt bzw. bei der Deregistrierung wieder entfernt. Das DataBlade trägt den Namen *PresFragments*. Näheres zum Aufbau und zur Installation dieses DataBlades entnehme man Anhang D.

Zur Realisierung von DataBlades stellt Informix das schon erwähnte Tool BladeSmith bereit, welches u.a. die Definition von Named Row Types und den zugehörigen Typed Tables unter einer graphischen Benutzeroberfläche erlaubt. Abbildung 6.9 zeigt die Definition eines Named Row Type mit diesem Tool. Über den gezeigten Dialog können Datenfelder eines Named Row Type hinzugefügt, gelöscht und geändert werden.

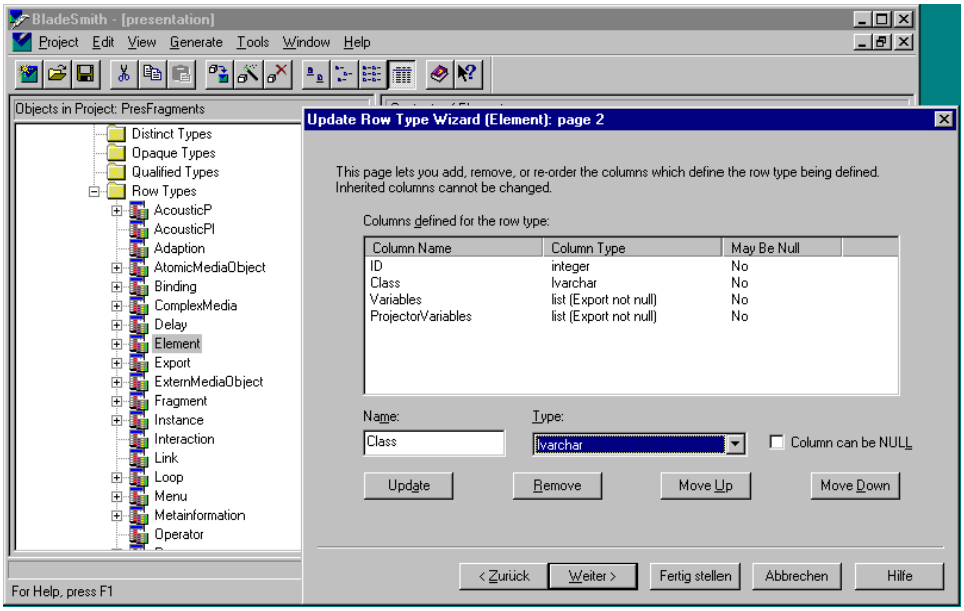


Abbildung 6.9: Definition eines Named Row Type im BladeSmith

Abbildung 6.10 zeigt die Definition eines Typed Table. Im Gegensatz zur Definition von Named Row Types ist die Unterstützung von Typed Tables nicht direkt im BladeSmith vorgesehen. BladeSmith erlaubt aber das Einbinden von beliebigen SQL-Anweisungen und damit auch von `create table`-Anweisungen.

BladeSmith generiert aus der erstellten Schemadefinition ein entsprechendes Instal-

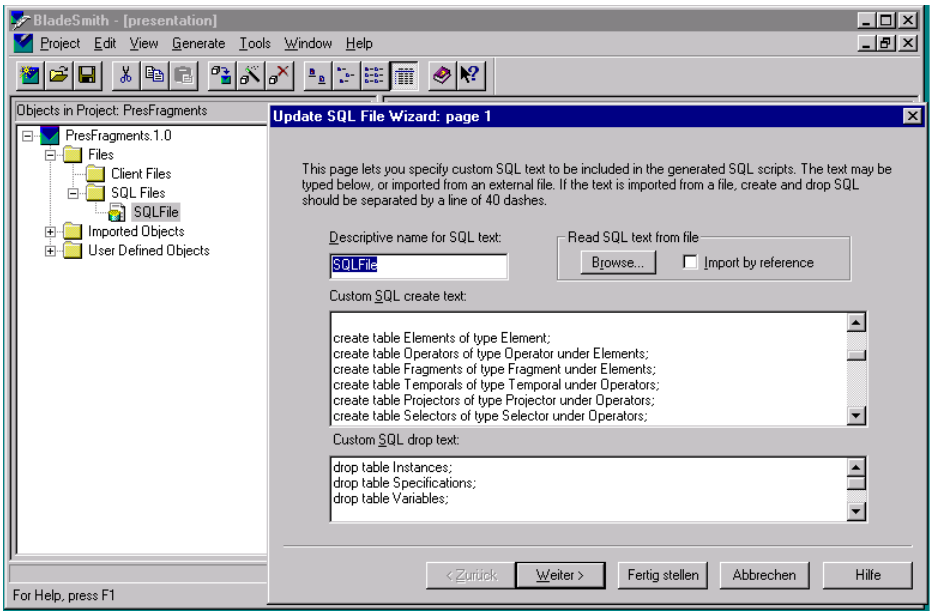


Abbildung 6.10: Definition von Typed Tables im BladeSmith

lationsskript in SQL, welches vom Werkzeug BladeManager zur Registration bzw. Deregistration eines Datablades verwendet wird. In der Praxis stößt man jedoch in der Version 3.40.TC1 von BladeSmith bei der Generierung des Installationsskripts auf einige Probleme. Zum einen kann es vorkommen, daß BladeSmith Deklarationen von Named Row Types in falscher Reihenfolge generiert, d.h. ein Named Row Type wird verwendet, bevor er deklariert ist. Zum anderen werden bei der Deklaration der Datenfelder eines Named Row Type keine `not null`-Anweisungen generiert. Außerdem gibt es Fälle, in denen SQL-Syntaxfehler generiert werden. Alles in allem bedeutet dies, daß das manuelle Nacheditieren des generierten Skriptes notwendig ist. Es ist aber zu erwarten, daß bei neueren Versionen diese Fehler nicht mehr auftreten.

Schließlich zeigt Abbildung 6.11 die Registration des DataBlades in einer Datenbank mit Hilfe des Werkzeugs BladeManager.

## 6.5 Serverbasierte Routinen

Wie schon im bisherigen Verlauf dieses Kapitels dargelegt wurde, bietet der Informix Dynamic Server/ Universal Data Option die Möglichkeit, serverbasierte Routinen in C oder in SPL zu definieren. In dieser Arbeit wird von hiervon aus mehreren Gründen Gebrauch gemacht.

Zum einen ist das Datenbankschema ziemlich komplex. Die Erzeugung eines Präsentationselementes ist beispielsweise mit relativ hohem Aufwand verbunden. So sind zuerst die zugehörigen Variablen des Präsentationselementes anzulegen. Dann gilt es, das Präsentationselement selbst zu erzeugen und diesem eine Liste mit Referenzen auf dessen Variablen zuzuweisen. Die Erzeugung eines komplexen Medienobjektes gestaltet sich aufgrund der notwendigen Berechnung der zu exportierenden Variablen noch aufwendiger. Um die Benutzung der Datenbank zu erleichtern, wird

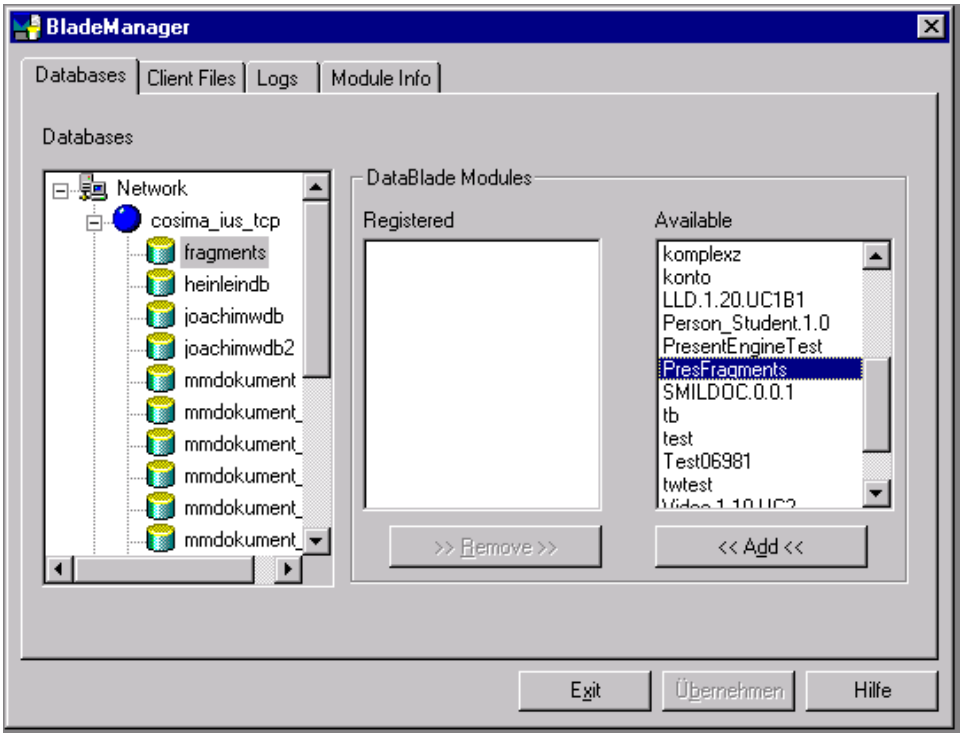


Abbildung 6.11: Registration eines DataBlades in einer Datenbank

deshalb eine Menge an Routinen implementiert, der das Erzeugen und Löschen von Präsentationselementen und Spezifikationsbäumen vereinfacht.

Zum anderen gibt es Funktionalität, bei der schon während der Definition des ZYX-Modells an eine serverseitige Implementierung gedacht wurde. So ist das *query*-Operatorelement mit dem Hintergedanken der serverbasierten Adaption eingeführt worden. Die Implementierung dieses Adaptionsmechanismus erfolgt somit ebenfalls über eine serverbasierte Routine.

Schließlich ist bei der Definition des ZYX-Modells viel Wert auf die Fähigkeit zur präsentationsneutralen Beschreibung eines multimedialen Dokuments gelegt worden. Zur präsentationsneutralen Ablage eines Dokumentes gehören Konvertierungsmöglichkeiten in andere multimediale Dokumentmodelle. Solche Konvertierungsroutinen können ebenfalls serverbasiert implementiert werden. In dieser Arbeit wird die Konvertierung in ein XML-basiertes Beschreibungsformat realisiert.

Die Implementierung der serverbasierten Routinen erfolgt in C, da sie zum Teil eine hohe Komplexität aufweisen und sich SPL schlecht zur Implementierung komplexerer Routinen eignet. Anhang C zeigt sämtliche implementierten serverbasierten Routinen. Im folgenden werden die serverbasierten Routinen kategorisiert und die typischen Vertreter der jeweiligen Kategorien näher vorgestellt.

### 6.5.1 Konstruktions-/Destruktionsroutinen

Diese Gruppe von Routinen dient u.a. dazu, Präsentationselemente und deren Instanzen zu erzeugen und zu löschen, Instanzen in Spezifikationsbäume einzufügen

und wieder zu entfernen und Variablen innerhalb von Spezifikationsbäumen an Instanzen von Präsentationselementen zu binden. Es sollen im folgenden die wesentlichen Routinen dieser Kategorie vorgestellt und ihr Zusammenspiel erläutert werden.

Zur Erzeugung eines Präsentationselements steht für jede Art von Präsentationselement eine Konstruktionsroutine zur Verfügung, deren Name sich aus dem Präfix `create` sowie dem Elementnamen herleitet. So erzeugt die Routine `createPar` ein *par*-Operatorelement samt zugehörigen Variablen. Die Routine `createComplex` erstellt hingegen ein komplexes Medienobjekt. Jeder `create`-Routine werden elementspezifische Parameter übergeben (z.B. `LipSync` und `Finish` bei `createPar`). Als Ergebnis einer solchen Routine wird die ID des erzeugten Präsentationselementes zurückgeliefert.

Zum Hinzufügen von Projektionsvariablen zu einem Präsentationselement dient die Routine `addProjVariable`. Sie fügt eine Projektionsvariable, die vorher über die Routine `createVariable` erstellt wurde, zur Liste der exportierten Projektionsvariablen eines gegebenen Präsentationselementes hinzu.

Wenn das erzeugte Präsentationselement in einem Spezifikationsbaum verwendet werden soll, muß zunächst eine Instanz des Präsentationselements erzeugt werden. Dieses geschieht über die Routine `createInstance`. Ihr wird die ID des Präsentationselementes übergeben. Sie erzeugt eine Instanz des Präsentationselements und liefert deren ID zurück. Die Instanz wird mit der Routine `addInstance` zu einem Spezifikationsbaum hinzugefügt.

Zur Bindung einer Instanz an eine Variable innerhalb eines Spezifikationsbaumes steht die Routine `bindVariable` für die Bindung an normale Variablen bzw. die Routine `bindProjVariable` für die Bindung an Projektionsvariablen zur Verfügung. Einer solchen Routine muß die ID des Spezifikationsbaumes, in der die Bindung stattfinden soll, die ID der Vaterinstanz, deren zugehöriges Präsentationselement die zu bindende Variable exportiert, die Nummer der Variable in der Exportliste des zur Vaterinstanz gehörigen Präsentationselementes und die ID der Instanz, die an die Variable gebunden werden soll, übergeben werden.

Es stehen zudem die Routinen `removeInstance`, `deleteInstance` und die Routine `deleteElement` zur Verfügung, die eine Instanz samt ihrer Bindungen aus einem Spezifikationsbaum entfernen, eine Instanz aus der Datenbank löschen und ein Präsentationselement samt seiner Variablen aus der Datenbank entfernen.

### 6.5.2 Navigationsroutinen

Während die Routinen der vorigen Kategorie dazu dienen, Präsentationselemente zu erzeugen und aus diesen Spezifikationsbäume zu konstruieren, erlauben die Navigationsroutinen die Navigation auf Spezifikationsbäumen.

Die Routine `getRoot` liefert zu einem angegebenen Spezifikationsbaum die Wurzel zurück. Mit der Routine `followVariable` können die Bindungen der Variablen des zu einer Instanz gehörenden Präsentationselementes verfolgt werden. Dazu werden die ID des Spezifikationsbaumes, die ID der Instanz sowie die Position der interessierenden Variable in der Liste der exportierten Variablen des der Instanz zugeordneten Präsentationselements der Routine `followVariable` als Argumente übergeben. Ist an die Variable eine Instanz gebunden, so wird deren ID zurückgeliefert, ansonsten $-1$. Die Routine `followProjVariable` leistet analoges für Projektionsvariablen.

Man kann im ZYX-Dokumentmodell Spezifikationsbäume durch Verwendung komplexer Medienobjekte verschachteln. Es kann vorkommen, daß in einem von einem komplexen Medienobjekt gekapselten Spezifikationsbaum eine Variable ungebunden ist. Diese wird exportiert. Es ist nun möglich, die Variable in einem anderen Spezifikationsbaum, der eine Instanz des komplexen Medienobjektes beinhaltet, zu binden. Um das Verfolgen von Bindungen durch die Kapselungsgrenzen komplexer Medienobjekte hindurch zu unterstützen, definieren die Routinen `followVariable` und `followProjVariable` einen zusätzlichen Parameter namens `Context`, der einen Stapel von Integer-Werten in Form einer Liste darstellt. Das erste Element der Liste ist das oberste Stapelelement. Auf den Stapel werden die IDs von Instanzen komplexer Medienobjekte abgelegt. Findet die Routine `followVariable` bzw. `followProjVariable` zu einer gesuchten Variable keine Bindung, so wird die oberste ID vom Stapel genommen. Es wird der Spezifikationsbaum ermittelt, in der eine Instanz eines komplexen Medienobjektes mit dieser ID verwendet wird und überprüft, ob die Variable in diesem Spezifikationsbaum gebunden ist. Wenn dies so ist, wird die ID der an die Variable gebundenen Instanz zurückgeliefert, andernfalls wird dieser Schritt solange wiederholt, bis entweder der Stapel leer ist oder eine an die Variable gebundene Instanz gefunden wurde.

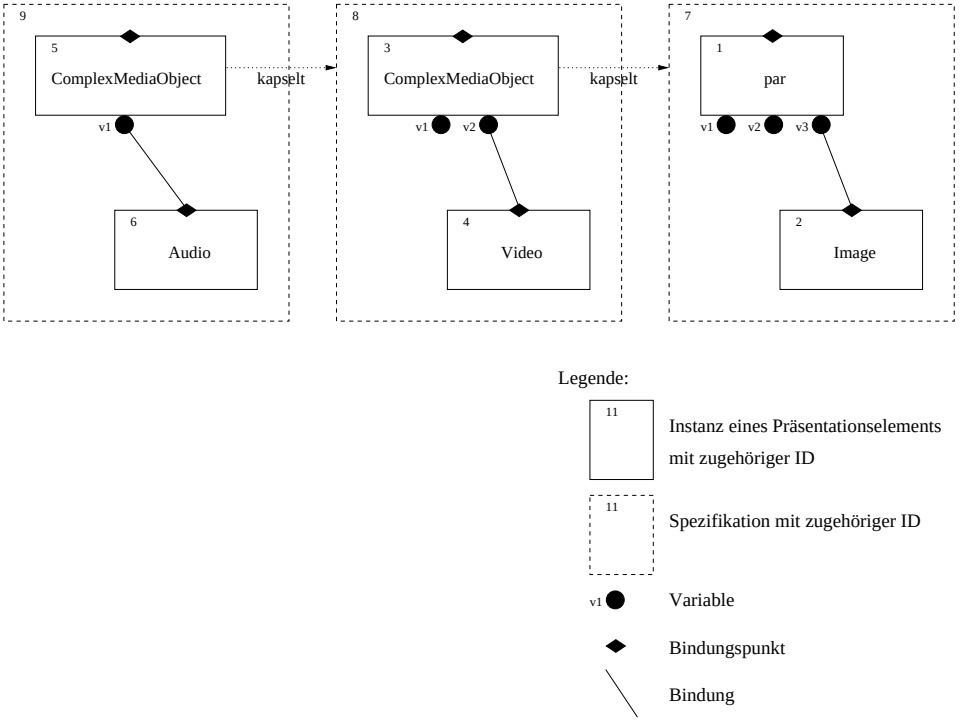


Abbildung 6.12: Beispiel einer verschachtelten Spezifikation

Abbildung 6.12 soll dieses anschaulich verdeutlichen. Möchte man im Spezifikationsbaum mit der ID 7 die an die Variablen der Instanz des *par*-Operatorelements gebundenen Instanzen anderer Präsentationselemente über `followVariable` ermitteln, so liefern die in Abbildung 6.13 gezeigten Aufrufe der Routine die dort angegebenen Ergebnisse.

Eine weitere Navigationsroutine implementiert die serverbasierte Adaption durch das *query*-Operatorelement. Die Routine `query` nimmt die ID eines *query*-Operatorelementes entgegen und sucht gemäß dem in Kapitel 5 spezifizierten Verfahren das passendste Fragment in der Datenbank und liefert dessen ID zurück. Sie erlaubt

```
followVariable(7, 1, 3, "list{}")      = 2, ID der Instanz des Image
followVariable(7, 1, 2, "list{}")      = -1
followVariable(7, 1, 1, "list{}")      = -1
followVariable(7, 1, 3, "list{3}")     = 2, ID der Instanz des Image
followVariable(7, 1, 2, "list{3}")     = 4, ID der Instanz des Videos
followVariable(7, 1, 1, "list{3}")     = -1
followVariable(7, 1, 3, "list{3, 5}") = 2, ID der Instanz des Image
followVariable(7, 1, 2, "list{3, 5}") = 4, ID der Instanz des Videos
followVariable(7, 1, 1, "list{3, 5}") = 6, ID der Instanz des Audios
```

Abbildung 6.13: Beispiel zur Verwendung von `followVariable`

somit die Navigation von einem *query*-Operatorelement zu dem von ihm repräsentierten Fragment.

### 6.5.3 Konvertierungsroutinen

Aus der präsentationsneutralen Ablage von Dokumenten resultiert die Notwendigkeit nach Konvertierungsfunktionalität in andere multimediale Dokumentmodelle und multimediale Dokumentstandards. In dieser Arbeit ist die Konvertierung von Spezifikationsbäumen des ZYX-Dokumentmodell in eine XML-Repräsentation implementiert worden, die u.a. von Präsentationssoftware zur Präsentation dieser Spezifikationsbäume genutzt werden soll. Die Konvertierung wird durch die serverbasierte Routine `exportXML` durchgeführt. Man übergibt dieser Routine die ID des zu konvertierenden Spezifikationsbaums und das Benutzerprofil. Die Routine erzeugt daraufhin eine Datei mit der XML-Beschreibung des Spezifikationsbaums unter Berücksichtigung des Benutzerprofils und liefert den Pfad der Datei zurück.

Die Funktionalität der Routine `exportXML` ist dabei nicht nur auf die simple 1 : 1-Abbildung der Präsentationselemente, die im Spezifikationsbaum verwendet werden, auf XML-Elemente beschränkt. Sie leistet zusätzliche Arbeiten, um die Präsentationssoftware zu entlasten.

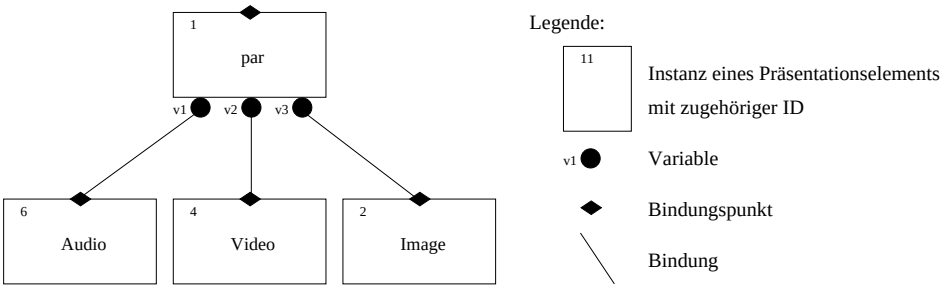


Abbildung 6.14: Abflachung des Spezifikationsbaumes aus Abbildung 6.12

Zum einen wird der Spezifikationsbaum abgeflacht. Das bedeutet, daß die in einem Spezifikationsbaum vorkommenden Instanzen von komplexen Medienobjekten durch die von ihnen gekapselten Spezifikationsbäume ersetzt werden. Das entspricht der *flatten*-Funktion aus Kapitel 5. Diese Ersetzung hat den Vorteil, daß die Präsentationssoftware sich nur noch mit einem einzigen, flachen Spezifikationsbaum beschäftigen muß. Von der Ersetzung ausgenommen sind lediglich die Instanzen derjenigen komplexen Medienobjekte, welche Linkziele beim *link*-Operatorelement bzw. beim SMIL-Medienobjekt darstellen. Diese Linkziele sind Referenzen auf vollständige Dokumente. Es ist möglich, zwischen diesen Dokumenten einen Zyklus von

Linkzielen zu konstruieren, der zu einer unendlichen Ersetzung der Instanzen von komplexen Medienobjekten führen würde. Das Nichtdurchführen der Ersetzung bei solchen komplexen Medienobjekten verhindert dieses. Abbildung 6.14 zeigt, wie der Spezifikationsbaum mit der ID 9 aus Abbildung 6.12 abgeflacht wird.

Zum anderen löst die Konvertierungsroutine serverbasierte Adaption auf. Im Spezifikationsbaum auftretende Instanzen von *query*-Operatorelementen werden über die Routine *query* durch die Instanzen passender Fragmente ersetzt.

Die Konvertierung eines Spezifikationsbaumes in das XML-Format kann aus zwei Gründen fehlschlagen. Die erste Möglichkeit des Fehlschlags besteht im Vorhandensein von ungebundenen Variablen. Spezifikationsbäume, die ungebundene Variablen enthalten, stellen keine vollständigen Dokumentbeschreibungen dar, weswegen die Konvertierung abgebrochen wird. Die zweite Möglichkeit des Fehlschlags liegt im Fehlen passender Fragmente zu einem *query*-Operatorelement.

```
1     <!ELEMENT element (metainformation?,parameters?, variables?,
2                        projectorvariables?)>
3     <!ATTLIST element
4               class         CDATA    #REQUIRED
5               url           CDATA    #IMPLIED
6               type          CDATA    #IMPLIED
7               format        CDATA    #IMPLIED
8               description   CDATA    #IMPLIED>
9
10    <!ELEMENT parameters (parameter+)>
11
12    <!ELEMENT parameter (#PCDATA| metainformation|
13                         variableconnections)>
14    <!ATTLIST parameter
15              name          CDATA    #REQUIRED>
16
17    <!ELEMENT metainformation (pair*)>
18
19    <!ELEMENT pair EMPTY>
20    <!ATTLIST pair
21              key           CDATA    #REQUIRED
22              value         CDATA    #REQUIRED>
23
24    <!ELEMENT variableconnections (connection*)>
25
26    <!ELEMENT connection EMPTY>
27    <!ATTLIST connection
28              external      CDATA    #REQUIRED
29              varno         CDATA    #REQUIRED>
30
31    <!ELEMENT variables (variable+)>
32
33    <!ELEMENT projectionvariables (variable+)>
34
35    <!ELEMENT variable (element)>
36    <!ATTLIST variable
37              no            CDATA    #REQUIRED
38              name          CDATA    #IMPLIED
39              description   CDATA    #IMPLIED>
```

Abbildung 6.15: DTD der XML-Repräsentation von ZYX

Die DTD, die das XML-Format beschreibt, zeigt Abbildung 6.15. Der Elementtyp `element` bildet die Grundlage dieser DTD. Ein XML-Element dieses Typs repräsentiert ein Präsentationselement des ZYX-Dokumentmodells . Der Elementtyp `element` definiert zur näheren Beschreibung des Präsentationselements ein Attribut namens `class`. Der Wert dieses Attributs spezifiziert die Art des Präsentationselementes. Er stimmt mit dem Klassennamen des jeweiligen Präsentationselementes im konzeptionellen Entwurf überein. So bedeutet der Wert `"par"`, daß es sich um ein *par*-Operatorelement handelt, während ein Attributwert `"atomicmediaobject"`

von einem atomaren Medienobjekt kündet. Es gibt jedoch eine Wertbelegung des Attributs `class`, die keine Entsprechung im konzeptionellen Entwurf des ZYX-Dokumentmodells hat: `"specification"`. Wenn das Attibut `class` diesen Wert zugewiesen bekommt, handelt es sich bei dem XML-Element um eine Referenz auf einen Spezifikationsbaum. Das Attribut `url` enthält die ID des Spezifikationbaumes. Das ist nötig, um Linkziele eines *link*-Operatorelements bzw. eines SMIL-Medienobjekts handhaben zu können.

Handelt es sich beim Präsentationselement um ein Fragment, können zusätzlich die Attribute `url`, `type`, `format` deklariert werden, die den gleichnamigen Attributen der Klassen `Fragment`, `AtomicMediaObject` und `SMILMediaObject` im konzeptionellen Entwurf entsprechen.

Der Inhalt eines XML-Elements des Typs `element` besteht aus einer Sequenz von vier optionalen XML-Elementen der Typen `metainformation`, `parameters` sowie `variables` und `projectionvariables`.

```
1   <?xml version="1.0" encoding="UTF-8" ?>
2   <!doctype zyx system "zyx.dtd">
3
4   <element class="par">
5   <parameters>
6   <parameter name="lipsync">0</parameter>
7   <parameter name="finish">max</parameter>
8   </parameters>
9   <variables>
10  <variable no="1" name="v1">
11  <element class="atomicmediaobject" url="somewhere1"
12          type="audio" format="wav">
13  <metainformation>
14  <pair key="bandwidth" value="medium"></pair>
15  </metainformation>
16  </element>
17  </variable>
18  <variable no="2" name="v2">
19  <element class="atomicmediaobject" url="somewhere2"
20          type="video" format="mpeg">
21  <metainformation>
22  <pair key="bandwidth" value="high"></pair>
23  </metainformation>
24  </element>
25  </variable>
26  <variable no="3" name="v3">
27  <element class="atomicmediaobject" url="somewhere3"
28          type="image" format="gif">
29  <metainformation>
30  <pair key="bandwidth" value="low"></pair>
31  </metainformation>
32  </element>
33  </variable>
34  </variables>
35  </element>
```

Abbildung 6.16: Beschreibung eines Spezifikationsbaumes in XML

Ein `metainformation`-Element wird nur bei Fragmenten definiert und enthält eine Folge von $(key, value)$-Paaren, die das Fragment näher beschreiben.

Das `parameters`-Element enthält eine Folge von Instanzen des XML-Elementtyps `parameter` und dient zur Beschreibung von präsentationselementabhängigen Parametern, beispielsweise den Parametern `lipSync` und `finish` bei einem *par*-Operatorelement. Ein XML-Element des Typs `parameter` nimmt dabei jeweils einen Parameter auf. Hierzu dient das Attribut `name`, in dem der Name des Parameters abgelegt wird. Der Inhalt eines `parameter`-Elements bildet den Wert des Parameters. Da manche Präsentationselemente Mengen von Metainformationen bzw. Kopplungen von Variablen als Werte von Parametern haben, kann der Inhalt eines

`parameter`-Elementes auch ein XML-Element des Typs `metainformation` bzw. ein XML-Element des Typs `variableconnections` sein.

Die XML-Elemente des Typs `variables` bzw. `projectionvariables` enthalten ein Folge von `variable`-Elementen, welche die Variablen bzw. Projektionsvariablen eines Präsentationselements aufnehmen. Hierzu definiert der Elementtyp `variable` die Attribute `no`, `name` und `description`, die zur Speicherung der Position der Variable in der Exportliste des Präsentationselements, zur Speicherung des Namens der Variable und zur Aufnahme der Beschreibung der Variable dienen. Der Inhalt eines `variable`-Elements ist ein XML-Element des Typs `element`, welches die an die Variable gebundene Instanz eines Präsentationselement repräsentiert.

Das Listing in Abbildung 6.16 zeigt, wie der in Abbildung 6.14 gezeigte Spezifikationsbaum in das XML-Format übersetzt wird.

# Kapitel 7

# Zusammenfassung und Ausblick

In dieser Arbeit wurde das Forschungsprojekt „Galerie der Herzchirurgie" vorgestellt. Die Anforderungen, die dieses Projekt an ein multimediales Dokumentmodell stellt, wurden dargelegt und näher untersucht. Im einzelnen ergaben sich die Forderungen nach der Wiederverwendbarkeit von multimedialen Dokumenten bzw. von deren Fragmenten, nach der Möglichkeit der Adaption des Dokumentinhaltes an den Kenntnisstand und die Vorlieben des Betrachters unter Berücksichtigung der ihm zur Verfügung stehenden technischen Möglichkeiten, nach der Modellierbarkeit sowohl von navigierender als auch gestalterische Interaktion und die Beschreibung des Dokumentinhaltes auf einem hohen semantischen Niveau, um die präsentationsneutrale Ablage der Dokumente zu erlauben.

Daraufhin wurden die multimedialen Dokumentstandards MHEG-5, HyTime und SMIL auf die Erfüllung dieser Anforderungen untersucht. Es ergab sich, daß keiner dieser Standards die gestellten Forderungen zufriedenstellend erfüllt. MHEG-5 beschreibt multimediale Dokumente auf einem sehr niedrigen semantischen Niveau, was die präsentationsneutrale Verwaltung von Dokumenten in einer Datenbank behindert. Außerdem schränkt der MHEG-5 Standard die Wiederverwendbarkeit von Dokumenten und deren Fragmente stark ein.

HyTime dagegen erfüllt wegen des mächtigen Mechanismus der Locators die an die Wiederverwendbarkeit gestellten Forderungen. Zudem bechreibt der Standard Dokumente auf einer hohen semantischen Ebene, wodurch er sich sehr gut zur präsentationsneutralen Ablage von Dokumenten eignet. Jedoch bietet der Standard keine Möglichkeit zur Modellierung von Interaktion und zur Modellierung von Adaption in multimedialen Dokumenten.

SMIL stellt in mancher Beziehung ein Komromiß zwischen MHEG-5 und HyTime dar. Das semantische Niveau der Beschreibung eines Dokuments liegt zwischen dem von MHEG-5 und HyTime, weswegen der Standard recht gut Präsenationsneutralität unterstützt. Jedoch weist SMIL Einschränkungen bei der Wiederverwendbarkeit und bei der Modellierung von Interaktion auf.

Aufgrund der Mängel der untersuchten Standards wurde das ZYX-Dokumentmodell formal definiert. Zusammenfassend läßt sich ZYX wie folgt charakterisieren:

- ZYX beschreibt Dokumente hierarchisch mit Hilfe eines Spezifikationsbaumes. Das semantische Niveau der Beschreibung von multimedialen Dokumenten ist sehr hoch, wodurch ZYX sich zur präsentationsneutralen Ablage von Dokumenten eignet.
- ZYX bietet Wiederverwendbarkeit über die Modellierungsmittel der atomaren und komplexen Medienobjekte auf der Ebene der Medienobjekte, der Fragmentebene sowie auf der Dokumentebene. Mit Hilfe von externen Medienobjekten ist es möglich, Dokumente anderer Standards ohne Konvertierung wiederzuverwenden.
- ZYX erlaubt die Erstellung von Dokumentschablonen.
- Durch die Mechanismen der Projektionsvariablen und Projektoren wird eine starke Trennung von Struktur und Layout eines Dokuments erreicht.
- Fragmente können mit Metainformationen zur softwarebasierten Suche und Selektion versehen werden.
- ZYX erlaubt die Modellierung von clientbasierter und serverbasierter Adaption des Dokumentinhalts an den Benutzer.
- Die Beschreibung von navigierender und gestalterischer Interaktion ist möglich.

Das ZYX-Modell wurde daraufhin in Form eines DataBlades für den Informix Dynamic Server/ Universal Data Option implementiert. Bei dieser Implementierung wurden sich insbesondere die Named Row Types dieses objektrelationalen Systems und die Möglichkeit der Programmierung von serverbasierten Routinen zunutze gemacht. Es wurden Routinen zur bequemen Erstellung und Löschung von Präsentationselementen und Spezifikationsbäumen implementiert. Zudem wurden Routinen zur einfachen Navigation über Spezifikationsbäume sowie zur Auflösung von serverbasierten Adaptionen umgesetzt. Schließlich wurde eine Routine zur Konvertierung in ein XML-basiertes Format erstellt.

Diese Arbeit hinterläßt noch einige „offene Baustellen", welche die die Grundlage für weitere Arbeiten bilden können:

- Es stellt sich die Frage, ob die vorgestellten Operatorelemente des ZYX-Modells ausreichend sind. Denkbar ist beispielsweise die Definition eines Operatorelements für Überblendeffekte (Fading). Zudem ist die Modellierung des räumlichen Layouts eines Dokumentes über den $spatial_p$-Projektor sehr statisch. Möglich wäre auch die Modellierung von räumlichen Beziehungen über relative Positionsangaben, beispielsweise durch Operatorelemente in der Art von *above*, *under*, *left* oder *right*.
- Die Implementierung des DataBlades kann noch erweitert werden. So ist eine Konvertierungsroutine von ZYX nach SMIL geplant. Außerdem ist zu untersuchen, ob das DataBlades durch Verwendung von Indizes verbessert werden kann. Der IDS/UDO bietet die Möglichkeit, eigene Zugriffstechniken zu implementieren. Hier ist es interessant zu wissen, ob sich speziell für das ZYX-Modell geeignete Zugriffstechniken finden und implementieren lassen.
- Die Entwicklung einer Präsentationssoftware für ZYX-Dokumente auf Grundlage des in dieser Arbeit eingeführten XML-basierten Beschreibungsformats ist nötig. Eine sich hiermit befassende Diplomarbeit wird gerade bearbeitet.

- Ebenso muß ein Autorenwerkzeug für ZYX-Dokumente zur Verfügung gestellt werden. Es sollte die einfache Erstellung von Spezifikationsbäumen erlauben und die im ZYX-Dokumentmodell verankerte Wiederverwendbarkeit von Dokumenten, deren Fragmenten und von atomaren Medienobjekten unterstützen.

- Schließlich wurde die Speicherung der von den atomaren Medienobjekten gekapselten Daten in dieser Arbeit offen gelassen. Hier stellt sich Frage nach der Möglichkeit der Ablage dieser Daten in einem DBMS. Zudem ist zu untersuchen, wie kontinuierliche Mediendaten mit hohem Datenvolumen sinnvoll und ohne große Qualitätseinbußen über ein Netzwerk zu übertragen sind. Dieses Thema wird gerade ebenfalls in einer Diplomarbeit bearbeitet.

# Anhang A

# Klassen des konzeptionellen Entwurfs von ZYX

Dieser Anhang zeigt die Klassen des objektorientierten Entwurfs des ZYX-Dokumentmodells zusammen mit ihren Attributen und einer Kurzbeschreibung in alphabetischer Reihenfolge. Die Assoziationen zwischen den Klassen sind aus den Abbildungen 6.1 auf den Seiten 74 und 76 ersichtlich.

| **Klasse:** | AcousticP |
|---|---|
| **Subklasse von:** | Projector |
| **Kurzbeschreibung:** | Modelliert das $acoustic_p$-Operatorelement. |
| **Attribut** | **Typ** |
| Volume | Integer |
| Base | Real |
| Treble | Real |
| Balance | Real |

| **Klasse:** | AcousticPI |
|---|---|
| **Subklasse von:** | AcousticP |
| **Kurzbeschreibung:** | Modelliert das $acoustic_{pi}$-Operatorelement. |

| **Klasse:** | Adaption |
|---|---|
| **Subklasse von:** | Operator |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die Adaptionsoperatorelemente des ZYX-Dokumentmodells . |

| **Klasse:** | AtomicMediaObject |
|---|---|
| **Subklasse von:** | Fragment |
| **Kurzbeschreibung:** | Modelliert ein atomares Medienobjekt. |
| **Attribut** | **Typ** |
| Url | String |
| Type | String |
| Format | String |

| **Klasse:** | Binding |
|---|---|
| **Kurzbeschreibung:** | Repräsentiert die Bindung einer Instanz eines Präsentationselements an eine Variable eines anderen Präsentationselements innerhalb eines Spezifikationsbaumes. |

| **Klasse:** | ComplexMediaObject |
|---|---|
| **Subklasse von:** | Fragment |
| **Kurzbeschreibung:** | Repräsentiert ein komplexes Medienobjekt. |

| **Klasse:** | Delay |
|---|---|
| **Subklasse von:** | Temporal |
| **Kurzbeschreibung:** | Repräsentiert das *delay*-Operatorelement. |
| **Attribut** | **Typ** |
| T | Integer |

| **Klasse:** | Element |
|---|---|
| **Kurzbeschreibung:** | Modelliert ein Präsentationselement. |
| **Attribut** | **Typ** |
| ID | Integer |

| **Klasse:** | Export (Assoziationsklasse) |
|---|---|
| **Kurzbeschreibung:** | Dient zur Modellierung der zu einem Präsentationselement gehörenden Variablen. |
| **Attribut** | **Typ** |
| No | Integer |

| **Klasse:** | ExternalMediaObject |
|---|---|
| **Subklasse von:** | Fragment |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für externe Medienobjekte. |
| **Attribut** | **Typ** |
| Url | String |
| Format | String |

| **Klasse:** | Fragment |
|---|---|
| **Subklasse von:** | Element |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die Fragmente des ZYX-Dokumentmodells . |
| **Attribut** | **Typ** |
| Description | String |

| **Klasse:** | Instance |
|---|---|
| **Kurzbeschreibung:** | Stellt die Instanz eines Präsentationselements dar. |
| **Attribut** | **Typ** |
| ID | Integer |

| **Klasse:** | Interaction |
|---|---|
| **Subklasse von:** | Operator |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die Interaktionsoperatorelemente des ZYX-Dokumentmodells . |

| **Klasse:** | Link |
|---|---|
| **Subklasse von:** | Interaction |
| **Kurzbeschreibung:** | Stellt das *link*-Operatorelement von ZYX dar. |

| **Klasse:** | Loop |
|---|---|
| **Subklasse von:** | Temporal |
| **Kurzbeschreibung:** | Repräsentiert das *loop*-Operatorelement. |
| **Attribut** | **Typ** |
| N | Integer |

| **Klasse:** | Metainformation |
|---|---|
| **Kurzbeschreibung:** | Repräsentiert $(Key, Value)$-Paar. |
| **Attribut** | **Typ** |
| Key | String |
| Value | String |

| **Klasse:** | Menu |
|---|---|
| **Subklasse von:** | Interaction |
| **Kurzbeschreibung:** | Modelliert das *menu*-Operatorelement des ZYX-Dokumentmodells . |
| **Attribut** | **Typ** |
| Mode | $\{vanish, prevail\}$ |

| **Klasse:** | Operator |
|---|---|
| **Subklasse von:** | Element |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die Operatorelemente des ZYX-Dokumentmodells . |

| **Klasse:** | Par |
|---|---|
| **Subklasse von:** | Temporal |
| **Kurzbeschreibung:** | Modelliert das *par*-Operatorelement des ZYX-Dokumentmodells . |
| **Attribut** | **Typ** |
| Finish | $\{min, max, 1, \ldots, n\}$ |
| LipSync | Integer |

| **Klasse:** | Projector |
|---|---|
| **Subklasse von:** | Operator |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die Projektoren des ZYX-Dokumentmodells . |

| **Klasse:** | Query |
|---|---|
| **Subklasse von:** | Adaption |
| **Kurzbeschreibung:** | Modelliert das *query*-Operatorelement des ZYX-Dokumentmodells . |
| **Attribut** | **Typ** |
| MetainformationList | list{set{Metainformation}} |

| **Klasse:** | Selector |
|---|---|
| **Subklasse von:** | Operator |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die Selektoren des ZYX-Dokumentmodells . |

| **Klasse:** | Seq |
|---|---|
| **Subklasse von:** | Temporal |
| **Kurzbeschreibung:** | Modelliert das *seq*-Operatorelement des ZYX-Modells. |

| **Klasse:** | SMILMediaObject |
|---|---|
| **Subklasse von:** | ExternalMediaObject |
| **Kurzbeschreibung:** | Modelliert ein externes Medienobjekt, welches SMIL-Spezifikationen kapselt. |

| **Klasse:** | SpatialP |
|---|---|
| **Subklasse von:** | Projector |
| **Kurzbeschreibung:** | Modelliert das $spatial_p$-Operatorelement. |
| **Attribut** | **Typ** |
| X | Integer |
| Y | Integer |
| Width | Integer |
| Height | Integer |
| Priority | Integer |
| Type | $\{pixel, percent\}$ |

| **Klasse:** | SpatialPI |
|---|---|
| **Subklasse von:** | SpatialP |
| **Kurzbeschreibung:** | Modelliert das $spatial_{pi}$-Operatorelement. |

| **Klasse:** | SpatialS |
|---|---|
| **Subklasse von:** | Selector |
| **Kurzbeschreibung:** | Modelliert das $spatial_s$-Operatorelement. |
| **Attribut** | **Typ** |
| X | Integer |
| Y | Integer |
| Width | Integer |
| Height | Integer |

| **Klasse:** | SpatialSI |
|---|---|
| **Subklasse von:** | SpatialS |
| **Kurzbeschreibung:** | Modelliert das $spatial_{si}$-Operatorelement. |

| **Klasse:** | Specification |
|---|---|
| **Kurzbeschreibung:** | Repräsentiert die Spezifikationsbäume des ZYX-Modells. |
| **Attribut** | **Typ** |
| ID | Integer |

| **Klasse:** | Switch |
|---|---|
| **Subklasse von:** | Adaption |
| **Kurzbeschreibung:** | Modelliert das *switch*-Operatorelement des ZYX-Dokumentmodells . |
| **Attribut** | **Typ** |
| MetainformationList | list{set{Metainformation}} |

| **Klasse:** | Temporal |
|---|---|
| **Subklasse von:** | Operator |
| **Kurzbeschreibung:** | Abstrakte Basisklasse für die temporalen Operatorelemente des ZYX-Dokumentmodells . |

| **Klasse:** | TemporalP |
|---|---|
| **Subklasse von:** | Projector |
| **Kurzbeschreibung:** | Modelliert das $temporal_p$-Operatorelement. |
| **Attribut** | **Typ** |
| Direction | Integer |
| Speed | Real |

| **Klasse:** | TemporalPI |
|---|---|
| **Subklasse von:** | TemporalP |
| **Kurzbeschreibung:** | Modelliert das $temporal_{pi}$-Operatorelement. |

| **Klasse:** | TemporalS |
|---|---|
| **Subklasse von:** | Selector |
| **Kurzbeschreibung:** | Modelliert das $temporal_s$-Operatorelement. |
| **Attribut** | **Typ** |
| Start | Integer |
| Duration | Integer |

| **Klasse:** | TemporalSI |
|---|---|
| **Subklasse von:** | TemporalS |
| **Kurzbeschreibung:** | Modelliert das $temporal_{si}$-Operatorelement. |

| **Klasse:** | TypographicP |
|---|---|
| **Subklasse von:** | Projector |
| **Kurzbeschreibung:** | Modelliert das $typographic_p$-Operatorelement. |
| **Attribut** | **Typ** |
| Font | String |
| Size | Integer |
| Style | $\{normal, italic, bold, bold + italic\}$ |

| **Klasse:** | TypographicPI |
|---|---|
| **Subklasse von:** | TypegraphicP |
| **Kurzbeschreibung:** | Modelliert das $typographic_{pi}$-Operatorelement. |

| **Klasse:** | Variable |
|---|---|
| **Kurzbeschreibung:** | Modelliert eine Variable bzw. eine Projektionsvariable. Aufgrund des Exportmechanismus komplexer Medienobjekte kann eine Variable zu mehreren Elementen gehören (i.e. von diesen exportiert werden). |
| **Attribut** | **Typ** |
| ID | Integer |
| Kind | $\{normal, projection\}$ |
| Name | String |
| Description | String |

| **Klasse:** | VariableConnection |
|---|---|
| **Kurzbeschreibung:** | Modelliert die Verbindung eines SMIL-Link-Ziels mit einer Variable des die SMIL-Spezifikation kapselnden SMILMediaObjects. |
| **Attribut** | **Typ** |
| LinkTarget | String |

# Anhang B

# Datenbankschema des logischen Entwurfs

In diesem Anhang wird das Datenbankschema spezifiziert, auf das der konzeptionelle Entwurf abgebildet wurde. Er führt die einzelnen Named Row Types des logischen Entwurfs zusammen mit deren Datenfeldern, Supertypen und evtl. zugehörigen Typed Tables tabellarisch in alphabetischer Reihenfolge auf. Außerdem wird bei jedem Named Row Type angegeben, von welcher Klasse des konzeptionellen Entwurfs sich dieser herleitet.

| **Named Row Type:** | AcousticP | |
|---|---|---|
| **Subtyp von:** | Projector | |
| **zug. Typed Table:** | AcousticPs | |
| **Klasse:** | AcousticP | |
| **Datenfeld** | **Typ** | **NULL** |
| Volume | integer | nein |
| Base | real | nein |
| Treble | real | nein |
| Balance | real | nein |

| **Named Row Type:** | AcousticPI |
|---|---|
| **Subtyp von:** | AcousticP |
| **zug. Typed Table:** | AcousticPIs |
| **Klasse** | AcousticPI |

| **Named Row Type:** | Adaption |
|---|---|
| **Subtyp von:** | Operator |
| **zug. Typed Table:** | Adaptions |
| **Klasse:** | Adaption |

| **Named Row Type:** | AtomicMediaObject | |
|---|---|---|
| **Subtyp von:** | Fragment | |
| **zug. Typed Table:** | AtomicMediaObjects | |
| **Klasse:** | AtomicMediaObject | |
| **Datenfeld** | **Typ** | **NULL** |
| Url | lvarchar | nein |
| Type | lvarchar | nein |
| Format | lvarchar | nein |

| **Named Row Type:** | Binding | |
|---|---|---|
| **Klasse:** | Binding | |
| **Datenfeld** | **Typ** | **NULL** |
| ParentInstance | integer | nein |
| VariableNo | integer | nein |
| ChildInstance | integer | nein |

| **Named Row Type:** | ComplexMedia | |
|---|---|---|
| **Subtyp von:** | Fragment | |
| **zug. Typed Table:** | ComplexMedias | |
| **Klasse:** | ComplexMediaObject | |
| **Datenfeld** | **Typ** | **NULL** |
| Specification | integer | nein |

| **Named Row Type:** | Delay | |
|---|---|---|
| **Subtyp von:** | Temporal | |
| **zug. Typed Table:** | Delays | |
| **Klasse:** | Delay | |
| **Datenfeld** | **Typ** | **NULL** |
| T | integer | nein |

| **Named Row Type:** | Element | |
|---|---|---|
| **zug. Typed Table:** | Elements | |
| **Klasse:** | Element | |
| **Datenfeld** | **Typ** | **NULL** |
| ID | integer | nein |
| Class | lvarchar | nein |
| Variables | list(Export not null) | nein |
| ProjectorVariables | list(Export not null) | nein |

| **Named Row Type:** | Export | |
|---|---|---|
| **Klasse:** | Export | |
| **Datenfeld** | **Typ** | **NULL** |
| ID | integer | nein |
| ImportInstance | lvarchar | nein |
| ImportNo | integer | nein |

| **Named Row Type:** | ExternMediaObject | |
|---|---|---|
| **Subtyp von:** | Fragment | |
| **zug. Typed Table:** | ExternMediaObjects | |
| **Klasse:** | ExternalMediaObject | |
| **Datenfeld** | **Typ** | **NULL** |
| Url | lvarchar | nein |
| Format | lvarchar | nein |

| **Named Row Type:** | Fragment | |
|---|---|---|
| **Subtyp von:** | Element | |
| **zug. Typed Table:** | Fragments | |
| **Klasse:** | Fragment | |
| **Datenfeld** | **Typ** | **NULL** |
| Metainformations | set(Metainformation not null) | nein |
| Description | lvarchar | ja |

| **Named Row Type:** | Instance | |
|---|---|---|
| **zug. Typed Table:** | Instances | |
| **Klasse:** | Instance | |
| **Datenfeld** | **Typ** | **NULL** |
| ID | integer | nein |
| Element | integer | nein |
| Class | lvarchar | nein |
| Specification | integer | ja |

| **Named Row Type:** | Interaction |
|---|---|
| **Subtyp von:** | Operator |
| **zug. Typed Table:** | Interactions |
| **Klasse:** | Interaction |

| **Named Row Type:** | Link |
|---|---|
| **Subtyp von:** | Interaction |
| **zug. Typed Table:** | Links |
| **Klasse:** | Link |

| **Named Row Type:** | Loop | |
|---|---|---|
| **Subtyp von:** | Temporal | |
| **zug. Typed Table:** | Loops | |
| **Klasse:** | Loop | |
| **Datenfeld** | **Typ** | **NULL** |
| N | integer | nein |

| **Named Row Type:** | Metainformation | |
|---|---|---|
| **Klasse:** | Metainformation | |
| **Datenfeld** | **Typ** | **NULL** |
| Key | lvarchar | nein |
| Value | lvarchar | nein |

| **Named Row Type:** | Menu | |
|---|---|---|
| **Subtyp von:** | Interaction | |
| **zug. Typed Table:** | Menus | |
| **Klasse:** | Menu | |
| **Datenfeld** | **Typ** | **NULL** |
| Mode | integer ($vanish = 0$, $prevail = 1$) | nein |

| **Named Row Type:** | Operator |
|---|---|
| **Subtyp von:** | Element |
| **zug. Typed Table:** | Operators |
| **Klasse:** | Operator |

| **Named Row Type:** | Par | |
|---|---|---|
| **Subtyp von:** | Temporal | |
| **zug. Typed Table:** | Pars | |
| **Klasse:** | Par | |
| **Datenfeld** | **Typ** | **NULL** |
| Finish | integer ($min = -1$, $max = 0$) | nein |
| LipSync | integer | nein |

| **Named Row Type:** | Projector |
|---|---|
| **Subtyp von:** | Operator |
| **zug. Typed Table:** | Projectors |
| **Klasse:** | Projector |

| **Named Row Type:** | Query | |
|---|---|---|
| **Subtyp von:** | Adaption | |
| **zug. Typed Table:** | Queries | |
| **Klasse:** | Query | |
| **Datenfeld** | **Typ** | **NULL** |
| MetainformationList | list(set(Metainformation not null) not null) | nein |

| **Named Row Type:** | Selector |
|---|---|
| **Subtyp von:** | Operator |
| **zug. Typed Table:** | Selectors |
| **Klasse:** | Selector |

| **Named Row Type:** | Seq |
|---|---|
| **Subtyp von:** | Temporal |
| **zug. Typed Table:** | Seqs |
| **Klasse:** | Seq |

| **Named Row Type:** | SMILMediaObject | |
|---|---|---|
| **Subtyp von:** | ExternMediaObject | |
| **zug. Typed Table:** | SMILMediaObjects | |
| **Klasse:** | SMILMediaObject | |
| **Datenfeld** | **Typ** | **NULL** |
| Connections | list(VarConnection not null) | nein |

| **Named Row Type:** | SpatialP | |
|---|---|---|
| **Subtyp von:** | Projector | |
| **zug. Typed Table:** | SpatialPs | |
| **Klasse:** | SpatialP | |
| **Datenfeld** | **Typ** | **NULL** |
| X | integer | nein |
| Y | integer | nein |
| Width | integer | nein |
| Height | integer | nein |
| Priority | integer | nein |
| Type | integer ($pixel = 0$, $percent = 1$) | nein |

| **Named Row Type:** | SpatialPI |
|---|---|
| **Subtyp von:** | SpatialP |
| **zug. Typed Table:** | SpatialPIs |
| **Klasse:** | SpatialPI |

| **Named Row Type:** | SpatialS | |
|---|---|---|
| **Subtyp von:** | Selector | |
| **zug. Typed Table:** | SpatialSs | |
| **Klasse:** | SpatialS | |
| **Datenfeld** | **Typ** | **NULL** |
| X | integer | nein |
| Y | integer | nein |
| Width | integer | nein |
| Height | integer | nein |

| **Named Row Type:** | SpatialSI |
|---|---|
| **Subtyp von:** | SpatialS |
| **zug. Typed Table:** | SpatialSIs |
| **Klasse:** | SpatialSI |

| **Named Row Type:** | Specification | |
|---|---|---|
| **zug. Typed Table:** | Specifications | |
| **Klasse:** | Specification | |
| **Datenfeld** | **Typ** | **NULL** |
| ID | integer | nein |
| VariableBindings | set(Binding not null) | nein |
| ProjectorBindings | set(Binding not null) | nein |

| **Named Row Type:** | Switch | |
|---|---|---|
| **Subtyp von:** | Adaption | |
| **zug. Typed Table:** | Switches | |
| **Klasse:** | Switch | |
| **Datenfeld** | **Typ** | **NULL** |
| MetainformationList | list(set(Metainformation not null) not null) | nein |

| **Named Row Type:** | Temporal |
|---|---|
| **Subtyp von:** | Operator |
| **zug. Typed Table:** | Temporals |
| **Klasse:** | Temporal |

| **Named Row Type:** | TemporalP | |
|---|---|---|
| **Subtyp von:** | Projector | |
| **zug. Typed Table:** | TemporalPs | |
| **Klasse:** | TemporalP | |
| **Datenfeld** | **Typ** | **NULL** |
| Direction | integer | nein |
| Speed | real | nein |

| **Named Row Type:** | TemporalPI |
|---|---|
| **Subtyp von:** | TemporalP |
| **zug. Typed Table:** | TemporalPIs |
| **Klasse:** | TemporalPI |

| **Named Row Type:** | TemporalS | |
|---|---|---|
| **Subtyp von:** | Selector | |
| **zug. Typed Table:** | TemporalSs | |
| **Klasse:** | TemporalS | |
| **Datenfeld** | **Typ** | **NULL** |
| Start | integer | nein |
| Duration | integer | nein |

| **Named Row Type:** | TemporalSI |
|---|---|
| **Subtyp von:** | TemporalS |
| **zug. Typed Table:** | TemporalSIs |
| **Klasse:** | TemporalSI |

| **Named Row Type:** | TypographicP | |
|---|---|---|
| **Subtyp von:** | Projector | |
| **zug. Typed Table:** | TypographicPs | |
| **Klasse:** | TypographicP | |
| **Datenfeld** | **Typ** | **NULL** |
| Font | lvarchar | nein |
| Size | integer | nein |
| Style | integer ($normal = 0$, $italic = 1$, $bold = 2$, $bold + italic = 3$) | nein |

| **Named Row Type:** | TypographicPI |
|---|---|
| **Subtyp von:** | TypographicP |
| **zug. Typed Table:** | TypographicPIs |
| **Klasse:** | TypographicPI |

| **Named Row Type:** | Variable | |
|---|---|---|
| **zug. Typed Table:** | Variables | |
| **Klasse:** | Variable | |
| **Datenfeld** | **Typ** | **NULL** |
| ID | integer | nein |
| Kind | integer ($normal = 0$, $Projection = 1$) | nein |
| Name | lvarchar | ja |
| Description | lvarchr | ja |

| **Named Row Type:** | VarConnection | |
|---|---|---|
| **Klasse:** | VariableConnection | |
| **Datenfeld** | **Typ** | **NULL** |
| VariableNo | integer | nein |
| ExternalVariable | lvarchar | nein |

# Anhang C

# Serverbasierte Routinen

In diesem Anhang werden die in dieser Arbeit implementierten serverbasierten Routinen des Datablade zur Verwaltung von Dokumenten des ZYX-Dokumentmodells alphabetisch aufgeführt und deren Funktionsweisen, Parameter, Rückgabewerte und Fehlerfälle beschrieben.

| | | |
|---|---|---|
| **Name:** | `addInstance` | |
| **Beschreibung:** | Die Routine fügt eine Instanz eines Präsentationslements zu einem Spezifikationsbaum hinzu (Siehe auch den Abschnitt über Konstruktionsroutinen in Kapitel 6). | |
| **Parameter:** | | |
| `SpecificationID` | `integer` | ID des Spezifikationbaumes, zu dem die Instanz hinzugefügt werden soll. |
| `InstanceID` | `integer` | ID der hinzuzufügenden Instanz. |
| **Rückgabewert:** | keiner. | |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Der Spezifikationsbaum existiert nicht.<br>• Die Instanz wird schon in einem Spezifikationsbaum verwendet. | |

| | | |
|---|---|---|
| **Name:** | `addProjVariable` | |
| **Beschreibung:** | Die Routine fügt einem Präsentationselement eine Projektionsvariable hinzu. Die Variable muß zuvor mit `createVariable` erstellt worden sein. | |
| **Parameter:** | | |
| `ElementID` | `integer` | ID des Präsentationselementes, zu dem die Variable hinzugefügt werden soll. |

| | | |
|---|---|---|
| `VariableID` | `integer` | ID der Projektionsvariable, die dem Präsentationselement hinzugefügt werden soll. |

**Rückgabewert:** keiner.

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Beim Präsentationselement handelt es sich um einen Projektor.
- Das Präsentationselement wird schon in mehr als in einem Spezifikationsbaum verwendet.
- Das Präsentationselement oder die Projektionsvariable existieren nicht.

**Name:** `addVariable`

**Beschreibung:** Die Routine fügt einem Präsentationselement eine Variable hinzu. Die Variable muß zuvor mit `createVariable` erstellt worden sein.

**Parameter:**

| | | |
|---|---|---|
| `ElementID` | `integer` | ID des Präsentationselementes, zu dem die Variable hinzugefügt werden soll. |
| `VariableID` | `integer` | ID der Variable, die dem Element hinzugefügt werden soll. |

**Rückgabewert:** keiner.

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Beim Präsentationselement handelt es sich um einen Projektor.
- Das Präsentationselement wird schon in mehr als in einem Spezifikationsbaum verwendet.
- Das Präsentationselement oder die Variable existieren nicht.

**Name:** `bindProjVariable`

**Beschreibung:** Diese Routine bindet eine Instanz eines Projektors an die $n$-te (Zählung startet bei 1) Projektionsvariable einer Vaterinstanz in einem gegebenen Spezifikationsbaum. (Siehe auch den Abschnitt über Konstruktionsroutinen in Kapitel 6).

**Parameter:**

| | | |
|---|---|---|
| `SpecificationID` | `integer` | ID des Spezifikationbaums, in dem die Bindung stattfinden soll. |

| | | |
|---|---|---|
| `ParentInstanceID` | `integer` | ID der Instanz des Präsentationselements, das die Projektionsvariable bereitstellt. |
| `VarNo` | `integer` | Position der Projektionsvariable in der Exportliste des zur Vaterinstanz gehörenden Präsentationselements (i.e. $n$). |
| `InstanceID` | `integer` | ID der Instanz des Projektors, der an die Projektionsvariable gebunden werden soll. |

**Rückgabewert:** keiner.

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Der Spezifikationsbaum existiert nicht.
- Die Vaterinstanz oder die Instanz des Projektors existieren nicht oder sind nicht Teil des Spezifikationsbaumes.
- Das zur Vaterinstanz gehörende Präsentationselement definiert weniger als $n$-Projektionsvariablen.
- Die $n$-te Projektionsvariable des zur Vaterinstanz gehörenden Präsentationselements ist schon gebunden.
- Das zur Instanz gehörende Präsentationselement ist kein Projektor.

**Name:** `bindVariable`

**Beschreibung:** Diese Routine bindet eine Instanz eines Präsentationselements an die $n$-te (Zählung startet bei 1) Variable einer Vaterinstanz in einem gegebenen Spezifikationsbaum. (Siehe auch den Abschnitt über Konstruktionsroutinen in Kapitel 6).

**Parameter:**

| | | |
|---|---|---|
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, in der die Bindung stattfinden soll. |
| `ParentInstanceID` | `integer` | ID der Instanz des Präsentationselements, das die Variable bereitstellt. |
| `VarNo` | `integer` | Position der Variable in der Exportliste des zur Vaterinstanz gehörenden Präsentationselements (i.e. $n$). |
| `InstanceID` | `integer` | ID der Instanz des Präsentationselements, das an die Variable gebunden werden soll. |

**Rückgabewert:** keiner.

| | |
|---|---|
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Der Spezifikationsbaum existiert nicht.<br>• Die Vaterinstanz oder die Instanz des Präsentationselements existieren nicht oder sind nicht Teil der Spezifikation.<br>• Das zur Vaterinstanz gehörende Präsentationselement definiert weniger als $n$-Variablen.<br>• Die $n$-te Variable des zur Vaterinstanz gehörenden Präsentationselements ist schon gebunden.<br>• Das zur Instanz gehörende Präsentationselement ist ein Projektor. |

| | | |
|---|---|---|
| **Name:** | `createAcousticP` | |
| **Beschreibung:** | Diese Routine erzeugt einen $acoustic_p$-Projektor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Volume` | `integer` | Die gewünschte Lautstärke im Intervall $[0 \ldots 100]$. |
| `Base` | `real` | Die gewünschte Baßstärke im Intervall $[-1 \ldots 1]$. |
| `Treble` | `real` | Die gewünschte Höhenstärke im Intervall $[-1 \ldots 1]$. |
| `Balance` | `real` | Die gewünschte Balance im Intervall $[-1 \ldots 1]$. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createAcousticPI` | |
| **Beschreibung:** | Diese Routine erzeugt einen $acoustic_{pi}$-Interaktor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Volume` | `integer` | Die gewünschte Lautstärke im Intervall $[0 \ldots 100]$. |
| `Base` | `real` | Die gewünschte Baßstärke im Intervall $[-1 \ldots 1]$. |
| `Treble` | `real` | Die gewünschte Höhenstärke im Intervall $[-1 \ldots 1]$. |
| `Balance` | `real` | Die gewünschte Balance im Intervall $[-1 \ldots 1]$. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createAtomic` | |
| **Beschreibung:** | Diese Routine erzeugt ein atomares Medienobjekt in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Description` | `lvarchar` inkl. `NULL` | Kurze Beschreibung des atomaren Medienobjekts. |
| `Metainformation` | `set(Metainformation not null)` | Metainformationsmenge, die mit dem atomaren Medienobjekt assoziiert ist. |
| `Url` | `lvarchar` | Referenz auf die Mediendaten. |
| `Type` | `lvarchar` | Typ der Mediendaten. |
| `Format` | `lvarchar` | Format der Mediendaten. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createComplex` | |
| **Beschreibung:** | Diese Routine erzeugt ein komplexes Medienobjekt in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Description` | `lvarchar` inkl. `NULL` | Kurze Beschreibung des komplexen Medienobjekts. |
| `Metainformation` | `set(Metainformation not null)` | Metainformationsmenge, die mit dem ComplexMediaObject assoziiert ist. |
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, den das komplexe Medienobjekt kapseln soll. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | Die Routine wird in folgenden Fällen mit einem Fehler abgebrochen:<br>• Der Spezifikationsbaum existiert nicht.<br>• Der Spezifikationsbaum wird schon von einem komplexen Medienobjekt gekapselt. | |

| | | |
|---|---|---|
| **Name:** | `createDelay` | |
| **Beschreibung:** | Diese Routine erzeugt ein *delay*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `T` | `integer` | Dauer der Zeitverzögerung in msec. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createInstance` | |
| **Beschreibung:** | `createInstance` erzeugt eine Instanz eines Präsentationselements. Diese kann dann mit `addInstance` in einem Spezifikations verwendet werden. | |
| **Parameter:** | | |
| `ElementID` | `integer` | ID des Präsentationselements, von dem eine Instanz erzeugt werden soll. |
| **Rückgabewert:** | integer | ID der erzeugten Instanz. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Das Präsentationselement existiert nicht. | |

| | | |
|---|---|---|
| **Name:** | `createLink` | |
| **Beschreibung:** | Diese Routine erzeugt ein *link*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `N` | `integer` | Anzahl der (Anker, Linkziel)-Paare, mit denen das *link*-Operatorelement ausgestattet sein soll. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createLoop` | |
| **Beschreibung:** | Diese Routine erzeugt ein *loop*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `N` | `integer` | Zahl der Wiederholungen, $0 = infinite$ |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createMenu` | |
| **Beschreibung:** | Diese Routine erzeugt ein *menu*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `N` | `integer` | Anzahl der (Anker, Linkziel)-Paare, mit denen das *menu*-Operatorelement ausgestattet sein soll. |
| `Mode` | `integer` | Modus des Menüs:<br>$1 = vanish$, $2 = prevail$. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createPar` | |
| **Beschreibung:** | Diese Routine erzeugt ein *par*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `N` | `integer` | Zahl der zu erstellenden Variablen. |
| `Finish` | `integer` | Nummer der Variable in der Exportliste des *par*-Operatorelements, die dessen Präsentationsdauer bestimmt, bzw. $-1$ für minimale und 0 für maximale Präsentationsdauer. |
| `LipSync` | `integer` | 0, wenn keine lippensynchrone Präsentation erwünscht ist, $i$ sonst zur Bestimmung des Masterelements. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createQuery` | |
| **Beschreibung:** | Diese Routine erzeugt ein *query*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `MetainfoList` | `list(set(`<br>`Metainformation`<br>`not null)`<br>`not null)` | Entspricht den Parametern $Meta_1 \ldots Meta_n$ des *query*-Operatorelements. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSeq` | |
| **Beschreibung:** | Diese Routine erzeugt ein *seq*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `N` | `integer` | Anzahl der Variablen, mit denen das *seq*-Operatorelement ausgestattet sein soll. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSMIL` | |
| **Beschreibung:** | Diese Routine erzeugt ein SMILMediaObject in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Description` | `lvarchar` inkl. `NULL` | Kurze Beschreibung des SMILMediaObjects. |

| | | |
|---|---|---|
| `Metainformation` | `set(Metainformation not null)` | Metainformationsmenge, die mit dem SMILMediaObject assoziiert ist. |
| `Url` | `lvarchar` | Referenz auf die Mediendaten. |
| `Format` | `lvarchar` | Muß `SMIL` sein. |
| `Connections` | `list(VarConnection not null)` | Liste der Kopplungen von SMIL-Linkzielen an Variablen |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSpatialP` | |
| **Beschreibung:** | Diese Routine erzeugt einen $spatial_p$-Projektor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `X` | `integer` | x-Koordinate des Ursprungs des rechteckigen Präsentationsbereichs. |
| `Y` | `integer` | y-Koordinate des Ursprungs des rechteckigen Präsentationsbereichs. |
| `Width` | `integer` | Breite des rechteckigen Präsentationsbereichs. |
| `Height` | `integer` | Höhe des rechteckigen Präsentationsbereichs. |
| `Priority` | `integer` | Priorität des Präsentationsbereichs. |
| `Type` | `integer` | Art des Koordinatensystems: $0 = absolute$, $1 = percentual$ |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSpatialPI` | |
| **Beschreibung:** | Diese Routine erzeugt einen $spatial_{pi}$-Interaktor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `X` | `integer` | x-Koordinate des Ursprungs des rechteckigen Präsentationsbereichs. |
| `Y` | `integer` | y-Koordinate des Ursprungs des rechteckigen Präsentationsbereichs. |
| `Width` | `integer` | Breite des rechteckigen Präsentationsbereichs. |
| `Height` | `integer` | Höhe des rechteckigen Präsentationsbereichs. |
| `Priority` | `integer` | Priorität des Präsentationsbereichs. |

| | | |
|---|---|---|
| `Type` | `integer` | Art des Koordinatensystems: $0 = absolute$, $1 = percentual$ |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSpatialS` | |
| **Beschreibung:** | Diese Routine erzeugt einen $spatial_s$-Selektor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `X` | `integer` | x-Koordinate des des Ursprungs des selektierten Rechtecks. |
| `Y` | `integer` | y-Koordinate des des Ursprungs des selektierten Rechtecks. |
| `Width` | `integer` | Breite des selektierten Rechtecks in Pixeln. |
| `Height` | `integer` | Höhe des selektierten Rechtecks in Pixeln. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSpatialSI` | |
| **Beschreibung:** | Diese Routine erzeugt einen $spatial_{si}$-Interaktor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `X` | `integer` | x-Koordinate des des Ursprungs des selektierten Rechtecks. |
| `Y` | `integer` | y-Koordinate des des Ursprungs des selektierten Rechtecks. |
| `Width` | `integer` | Breite des selektierten Rechtecks in Pixeln. |
| `Height` | `integer` | Höhe des selektierten Rechtecks in Pixeln. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSpec` | |
| **Beschreibung:** | `createSpec` erzeugt einen neuen, leeren Spezifikationsbaum in der Datenbank. | |
| **Parameter:** | keine. | |
| **Rückgabewert:** | integer | ID des erzeugten Spezifikationsbaumes. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createSwitch` | |
| **Beschreibung:** | Diese Routine erzeugt ein *switch*-Operatorelement in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `N` | `integer` | Anzahl der Variablen (*default* nicht mitgezählt), über die das *switch*-Operatorelement verfügen soll. |
| `MetainfoList` | `list(set(` `Metainformation` `not null)` `not null)` | Entspricht den Parametern $Meta_1 \ldots Meta_n$ des *switch*-Operatorelements. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createTemporalP` | |
| **Beschreibung:** | Diese Routine erzeugt einen $temporal_p$-Projektor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Direction` | `integer` | Die gewünschte Abspielrichtung $\in \{-1, 1\}$. |
| `Speed` | `real` | Faktor, mit dem die Abspielgeschwindigkeit multipliziert wird. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createTemporalPI` | |
| **Beschreibung:** | Diese Routine erzeugt einen $temporal_{pi}$-Interaktor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Direction` | `integer` | Die gewünschte Abspielrichtung $\in \{-1, 1\}$. |
| `Speed` | `real` | Faktor, mit dem die Abspielgeschwindigkeit multipliziert wird. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createTemporalS` | |
| **Beschreibung:** | Diese Routine erzeugt einen $temporal_s$-Selektor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Start` | `integer` | Startzeitpunkt in msec. |
| `Duration` | `integer` | Dauer des Zeitintervalls in msec. |

| | | |
|---|---|---|
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createTemporalSI` | |
| **Beschreibung:** | Diese Routine erzeugt einen $temporal_{si}$-Interaktor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Start` | `integer` | Startzeitpunkt in msec. |
| `Duration` | `integer` | Dauer des Zeitintervalls in msec. |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createTypoP` | |
| **Beschreibung:** | Diese Routine erzeugt einen $typographic_p$-Projektor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Font` | `lvarchar` | Name des Zeichensatzes. |
| `Size` | `integer` | Größe des Zeichensatzes in Punkten. |
| `Style` | `integer` | Stil des Zeichsatzes: $0 = normal$, $1 = italic$, $2 = bold$, $3 = italic + bold$ |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createTypoPI` | |
| **Beschreibung:** | Diese Routine erzeugt einen $typographic_{pi}$-Interaktor in der Datenbank und liefert dessen ID zuück. | |
| **Parameter:** | | |
| `Font` | `lvarchar` | Name des Zeichensatzes. |
| `Size` | `integer` | Größe des Zeichensatzes in Punkten. |
| `Style` | `integer` | Stil des Zeichsatzes: $0 = normal$, $1 = italic$, $2 = bold$, $3 = italic + bold$ |
| **Rückgabewert:** | integer | ID des erzeugten Präsentationselements. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `createVariable` | |
| **Beschreibung:** | Diese Routine erzeugt eine Variable in der Datenbank und liefert deren ID zuück. | |
| **Parameter:** | | |
| `Kind` | `integer` | Art der Variable: $0 = normal$, $1 = projection$. |

| | | |
|---|---|---|
| `Name` | `lvarchar` inkl. `NULL` | Optionaler Name der Variable. |
| `Description` | `lvarchar` inkl. `NULL` | Optionale Beschreibung der Variable. |
| **Rückgabewert:** | integer | ID der erzeugten Variable. |
| **Fehlerfälle:** | keine. | |

| | |
|---|---|
| **Name:** | `deleteElement` |
| **Beschreibung:** | `deleteElement` entfernt ein Präsentationselement aus der Datenbank. |
| **Parameter:** | |
| `ElementID` | `integer` — ID des zu entfernenden Präsentationselements. |
| **Rückgabewert:** | keiner. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Das Präsentationselement existiert nicht.<br>• Vom Präsentationselement existieren immer noch Instanzen. |

| | |
|---|---|
| **Name:** | `deleteInstance` |
| **Beschreibung:** | `deleteInstance` entfernt eine Instanz eines Präsentationselementes aus der Datenbank. |
| **Parameter:** | |
| `InstanceID` | `integer` — ID der zu entfernenden Instanz. |
| **Rückgabewert:** | keiner. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Die Instanz existiert nicht.<br>• Die Instanz wird noch in Spezifikationsbäumen verwendet. |

| | |
|---|---|
| **Name:** | `export` |
| **Beschreibung:** | Diese Routine berechnet die in einem Spezifikationsbaum ungebundenen Variablen und Projektionsvariablen. Die Exportlisten des den Spezifikationsbaum kapselnden komplexen Medienobjekts werden entsprechend angepaßt. Diese Routine sollte immer dann aufgerufen werden, wenn irgendwelche Änderungen an einem Spezifikationsbaum durchgeführt wurden (beispielweise `bind` oder `unbind`), der von einem komplexen Medienobjekt gekapselt wird. |
| **Parameter:** | |

| | | |
|---|---|---|
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, dessen Variablen exportiert werden sollen. |

**Rückgabewert:** keiner.

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Der Spezifikationsbaum existiert nicht.
- Der Spezifikationsbaum wird nicht von einem komplexen Medienobjekt gekapselt.
- Das den Spezifikationsbaum kapselnde komplexe Medienobjekt wird schon in anderen Spezifikationsbäumen verwendet und `export` würde die exportierten Variablen oder Projektionsvariablen des komplexen Medienobjekts ändern.

**Name:** `exportXML`

**Beschreibung:** Diese Routine erzeugt zu einem Spezifikationsbaum eine Beschreibung im XML-Format unter Berücksichtigung des Benutzerprofils und legt diese in einer Datei ab. Näheres zum XML-Format entnehme man bitte in Kapitel 6 dem Abschnitt über Konvertierungsfunktionalität.

**Parameter:**

| | | |
|---|---|---|
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, von der eine XML-Beschreibung erstellt werden soll. |
| `Profile` | `set(Metainformation not null)` | Das Profil des Benutzers. |

**Rückgabewert:** `lvarchar` — Pfad der erzeugten Datei.

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Eine Variable ist nicht gebunden.
- Zu einem *query*-Operatorelement gibt es kein passendes Fragment.
- Der Spezifikationsbaum existiert nicht.

| | | |
|---|---|---|
| **Name:** | `followProjVariable` | |
| **Beschreibung:** | In einem gegebenen Spezifikationsbaum liefert diese Funktion die ID derjenigen Instanz eines Projektors zurück, die an die $n$-te Projektionsvariable einer spezifizierten Vaterinstanz gebunden ist.<br>Sollte die $n$-te Projektionsvariable ungebunden sein, wird der Parameter `Context` überprüft. Dieser Parameter stellt einen Stack dar, auf dem die IDs von Instanzen komplexer Medienobjekte abgelegt werden können, welche den angegebenen Spezifikationsbaum direkt oder indirekt kapseln. Die oberste Instanz-ID des Stack wird entfernt und derjenige Spezifikationsbaum ermittelt, in dem die Instanz eines komplexen Medienobjektes mit dieser ID vorkommt. Es wird nun überprüft, ob die Projektionsvariable in diesem Spezifikationsbaum gebunden ist. Falls nicht, wird das nächste Element vom Stack genommen und so weiter.<br>Zum besseren Verständnis empfiehlt sich die Lektüre des Abschnittes über Navigationsroutinen in Kapitel 6. | |
| **Parameter:** | | |
| `SpecificationID` | `integer` | ID des Spezifikationsbaums, in dem die gebundene Instanz ermittelt werden soll. |
| `ParentInstanceID` | `integer` | ID der Instanz, deren Projektionsvariable verfolgt werden soll. |
| `VarNo` | `integer` | Position der Projektionsvariable in der Exportliste des zur Vaterinstanz gehörenden Präsentationselements (i.e. $n$). |
| `Context` | `list(integer not null)` | Stack mit den IDs der Instanzen komplexer Medienobjekte. Das erste Element der Liste ist das oberste Element des Stack. |
| **Rückgabewert:** | `integer` | -1, wenn keine Instanz gefunden wurde, die an die spezifizierte Projektionsvariable gebunden ist. Ansonsten wird die ID der an die Projektionsvariable gebundenen Instanz zurückgeliefert. |
| **Fehlerfälle:** | In den folgenden Fällen bricht die Routine mit einem Fehler ab:<br>• Der Spezifikationsbaum existiert nicht.<br>• Die Vaterinstanz ist nicht Teil des Spezifikationsbaumes oder definiert weniger als $n$-Projektionsvariablen.<br>• `Context` enthält IDs von Instanzen, die nicht zu komplexen Medienobjekten gehören. | |

| | | |
|---|---|---|
| **Name:** | `followVariable` | |
| **Beschreibung:** | In einem gegebenen Spezifikationsbaum liefert diese Funktion die ID derjenigen Instanz eines Präsentationselements zurück, die an die $n$-te Variable einer spezifizierten Vaterinstanz gebunden ist.<br>Sollte die $n$-te Variable ungebunden sein, wird der Parameter `Context` überprüft. Dieser Parameter stellt einen Stack dar, auf dem die IDs von Instanzen komplexer Medienobjekte abgelegt werden können, welche den angegebenen Spezifikationsbaum direkt oder indirekt kapseln. Die oberste Instanz-ID des Stack wird entfernt und derjenige Spezifikationsbaum ermittelt, in dem die Instanz eines komplexen Medienobjektes mit dieser ID vorkommt. Es wird nun überprüft, ob die Variable in diesem Spezifikationsbaum gebunden ist. Falls nicht, wird das nächste Element vom Stack genommen und so weiter.<br>Zum besseren Verständnis empfiehlt sich die Lektüre des Abschnittes über Navigationsroutinen in Kapitel 6. | |
| **Parameter:** | | |
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, in dem die gebundene Instanz ermittelt werden soll. |
| `ParentInstanceID` | `integer` | ID der Instanz, deren Variable verfolgt werden soll. |
| `VarNo` | `integer` | Position der Variable in der Exportliste des zur Vaterinstanz gehörenden Präsentationselements (i.e. $n$). |
| `Context` | `list(integer not null)` | Stack mit den IDs der Instanzen komplexer Medienobjekte. Das erste Element der Liste ist das oberste Element des Stack. |
| **Rückgabewert:** | `integer` | -1, wenn keine Instanz gefunden wurde, die an die spezifizierte Variable gebunden ist. Ansonsten wird die ID der an die Variable gebundenen Instanz zurückgeliefert. |
| **Fehlerfälle:** | In den folgenden Fällen bricht die Routine mit einem Fehler ab:<br>• Der Spezifikationsbaum existiert nicht.<br>• Die Vaterinstanz ist nicht Teil des Spezifikationbaumes oder definiert weniger als $n$-Variablen.<br>• `Context` enthält IDs von Instanzen, die nicht zu komplexen Medienobjekten gehören. | |

| | | |
|---|---|---|
| **Name:** | `getElement` | |
| **Beschreibung:** | `getElement` liefert zu einer Instanz das zugehörige Präsentationselement. | |
| **Parameter:** | | |
| `InstanceID` | `integer` | ID der Instanz, an deren zugehörigem Präsentationselement man interessiert ist. |
| **Rückgabewert:** | Element | Das zugehörige Präsentationselement. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Die Instanz existiert nicht. | |

| | | |
|---|---|---|
| **Name:** | `getNoProjVariables` | |
| **Beschreibung:** | Diese Funktion liefert die Zahl der Projektionsvariablen, die einem Präsentationselement zugeordnet sind. | |
| **Parameter:** | | |
| `ElementID` | `integer` | ID des Präsentationselements, an dessen Projektionsvariablen man interessiert ist. |
| **Rückgabewert:** | `integer` | Die Zahl der zugeordneten Projektionsvariablen. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:<br>• Das Präsentationselement existiert nicht. | |

| | | |
|---|---|---|
| **Name:** | `getNoVariables` | |
| **Beschreibung:** | Diese Funktion liefert die Zahl der Variablen, die einem Präsentationselement zugeordnet sind. | |
| **Parameter:** | | |
| `ElementID` | `integer` | ID des Elements, an dessen Variablen man interessiert ist. |
| **Rückgabewert:** | `integer` | Die Zahl der zugeordneten Variablen. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:<br>• Das Präsentationselement existiert nicht. | |

| | |
|---|---|
| **Name:** | `getProjVariable` |
| **Beschreibung:** | Diese Funktion liefert die $n$-te Projektionsvariable aus der Exportliste eines Präsentationselementes. Die Zählung beginnt dabei mit 1. |
| **Parameter:** | |

| | | |
|---|---|---|
| `ElementID` | `integer` | ID des Präsentationselements, an dessen Projektionsvariable man interessiert ist. |
| `No` | `integer` | Die Position der Projektionsvariable in der Exportliste des Präsentationselements (i.e. $n$). |
| **Rückgabewert:** | `Variable` | Die gewünschte Projektionsvariable. |

**Fehlerfälle:** Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:

- Das Präsentationselement existiert nicht.
- Das Präsentationselement exportiert weniger als $n$ Projektionsvariablen.

---

**Name:** `getRoot`

**Beschreibung:** Diese Funktion liefert die ID derjenigen Instanz zurück, welche die Wurzel des angegebenen Spezifikationsbaumes bildet.

**Parameter:**

| | | |
|---|---|---|
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, an dessen Wurzel man interessiert ist. |
| **Rückgabewert:** | `integer` | Die ID der Wurzelinstanz. |

**Fehlerfälle:** Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:

- Der Spezifikationsbaum existiert nicht.
- Die Spezifikationbaum hat keine Wurzel.

---

**Name:** `getSpecification`

**Beschreibung:** `getSpecification` liefert zu einer Instanz die ID des Spezifikationsbaumes, in dem sie verwendet wird.

**Parameter:**

| | | |
|---|---|---|
| `InstanceID` | `integer` | ID der Instanz, an deren Spezifikationsbaum man interessiert ist. |
| **Rückgabewert:** | Element | Die ID des Spezifikationsbaumes. |

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Die Instanz existiert nicht.
- Die Instanz wird in keinem Spezifikationsbaum verwendet.

| | | |
|---|---|---|
| **Name:** | `getVariable` | |
| **Beschreibung:** | Diese Funktion liefert die $n$-te Variable aus der Exportliste eines Präsentationselementes. Die Zählung beginnt dabei mit 1. | |
| **Parameter:** | | |
| `ElementID` | `integer` | ID des Präsentationselements, an dessen Variable man interessiert ist. |
| `No` | `integer` | Die Position der Variable in der Exportliste des Präsentationselements (i.e. $n$). |
| **Rückgabewert:** | `Variable` | Die gewünschte Variable. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:<br>• Das Präsentationselement existiert nicht.<br>• Das Präsentationselement exportiert weniger als $n$ Variablen. | |

| | | |
|---|---|---|
| **Name:** | `inSpecification` | |
| **Beschreibung:** | Diese Funktion überprüft, ob die gegebene Instanz eines Präsentationselements in einem gewissen Spezifikationsbaum verwendet wird. | |
| **Parameter:** | | |
| `SpecificationID` | `integer` | ID des Spezifikationbaumes, in dem die Instanz angeblich Verwendung finden soll. |
| `InstanceID` | `integer` | ID der Instanz eines Präsentationselements, die geprüft werden soll. |
| **Rückgabewert:** | `boolean` | `True`, wenn die Instanz im Spezifikationsbaum verwendet wird, ansonsten `False`. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:<br>• Der Spezifikationsbaum existiert nicht.<br>• Die Instanz existiert nicht. | |

| | | |
|---|---|---|
| **Name:** | `isMatching` | |
| **Beschreibung:** | Diese Funktion stellt fest, ob sich zwei Metainformationsmengen ähneln. Sie entspricht dem Prädikat *matching* aus Kapitel 5 und wird in der Regel in Verbindung mit der Funktion `noMatches` benutzt. | |
| **Parameter:** | | |
| `Meta1` | `set(Metainformation not null)` | Erste Metainformationsmenge. |
| `Meta2` | `set(Metainformation not null)` | Zweite Metainformationsmenge. |

| | | |
|---|---|---|
| **Rückgabewert:** | `boolean` | Wahr gdw. $matching(Meta1, Meta2)$. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `isProjector` | |
| **Beschreibung:** | Diese Funktion überprüft, ob das zu einer Instanz gehörige Präsentationselement ein Projektor ist. | |
| **Parameter:** | | |
| `InstanceID` | `integer` | ID der Instanz, in deren Präsentationselement man interessiert ist. |
| **Rückgabewert:** | `boolean` | `True`, wenn das zugehörige Präsentationselement ein Projektor ist, ansonsten `False`. |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in nachstehenden Fällen ab:<br>• Die Instanz existiert nicht. | |

| | | |
|---|---|---|
| **Name:** | `noMatches` | |
| **Beschreibung:** | Diese Funktion stellt fest, wie stark sich zwei Metainformationsmengen ähneln. Sie entspricht dem Maß *matches* aus Kapitel 5 und wird in der Regel in Verbindung mit der Funktion `isMatching` benutzt. | |
| **Parameter:** | | |
| `Meta1` | `set(Metainformation not null)` | Erste Metainformationsmenge. |
| `Meta2` | `set(Metainformation not null)` | Zweite Metainformationsmenge. |
| **Rückgabewert:** | `integer` | Die Korrelation zwischen `Meta1` und `Meta2`. |
| **Fehlerfälle:** | keine. | |

| | | |
|---|---|---|
| **Name:** | `query` | |
| **Beschreibung:** | `query` sucht in der Datenbank zu einem *query*-Operatorelement das passendste Fragment, wie in Kapitel 5 beschrieben. | |
| **Parameter:** | | |
| `ElementID` | `integer` | ID des *query*-Operatorelements. |
| **Rückgabewert:** | ID des Fragments, welches am besten den Anforderungen des *query*-Operatorelements gerecht wird. | |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Das *query*-Operatorelement exisitiert nicht.<br>• Es kann kein passendes Fragment in der Datenbank gefunden werden. | |

| | | |
|---|---|---|
| **Name:** | `removeInstance` | |
| **Beschreibung:** | `removeInstance` entfernt eine Instanz eines Präsentationselementes aus einem Spezifikationsbaum. | |
| **Parameter:** | | |
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, aus dem die Instanz entfernt werden soll. |
| `InstanceID` | `integer` | ID der zu entfernenden Instanz. |
| **Rückgabewert:** | keiner. | |
| **Fehlerfälle:** | Die Routine bricht mit einem Fehler in folgenden Fällen ab:<br>• Der Spezifikationsbaum existiert nicht.<br>• Die Instanz existiert nicht.<br>• Die Instanz ist nicht Teil des Spezifikationbaumes. | |

| | | |
|---|---|---|
| **Name:** | `unbindProjVariable` | |
| **Beschreibung:** | Diese Routine entfernt die Bindung einer Instanz eines Projektors an die $n$-te (Zählung startet bei 1) Projektionsvariable einer Vaterinstanz in einem gegebenen Spezifikationsbaum. | |
| **Parameter:** | | |
| `SpecificationID` | `integer` | ID des Spezifikationbaumes, in dem die Bindung existiert. |
| `ParentInstanceID` | `integer` | ID der Instanz des Präsentationselements, das die Projektionsvariable bereitstellt. |
| `VarNo` | `integer` | Position der Projektionsvariable in der Exportliste des zur Vaterinstanz gehörenden Präsentationselements (i.e. $n$). |
| `InstanceID` | `integer` | ID der Instanz des Projektors, der von der spezifizierten Projektionsvariable entbunden werden soll. |
| **Rückgabewert:** | keiner. | |

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Der Spezifikationsbaum existiert nicht.
- Die Vaterinstanz oder die Instanz des Projektors existieren nicht oder sind nicht Teil des Spezifikationbaumes.
- Das zur Vaterinstanz gehörende Präsentationselement definiert weniger als $n$-Projektionsvariablen.
- Die $n$-te Projektionsvariable des zur Vaterinstanz gehörenden Präsentationselements ist nicht an die Instanz des Projektors gebunden.

**Name:** `unbindVariable`

**Beschreibung:** Diese Routine entfernt die Bindung einer Instanz eines Präsentationselements an die $n$-te (Zählung startet bei 1) Variable einer Vaterinstanz in einem gegebenen Spezifikationsbaum.

**Parameter:**

| | | |
|---|---|---|
| `SpecificationID` | `integer` | ID des Spezifikationsbaumes, in dem die Bindung existiert. |
| `ParentInstanceID` | `integer` | ID der Instanz des Präsentationselements, das die Variable bereitstellt. |
| `VarNo` | `integer` | Position der Variable in der Exportliste des zur Vaterinstanz gehörenden Präsentationselements (i.e. $n$). |
| `InstanceID` | `integer` | ID der Instanz des Präsentationselements, das von der spezifizierten Variable entbunden werden soll. |

**Rückgabewert:** keiner.

**Fehlerfälle:** Die Routine bricht mit einem Fehler in folgenden Fällen ab:

- Der Spezifikationsbaum existiert nicht.
- Die Vaterinstanz oder die Instanz des Präsentationselements existieren nicht oder sind nicht Teil des Spezifikationsbaumes.
- Das zur Vaterinstanz gehörende Präsentationselement definiert weniger als $n$-Variablen.
- Die $n$-te Variable des zur Vaterinstanz gehörenden Präsentationselements ist nicht an die zu entbindende Instanz gebunden.

# Anhang D

# Das PresFragments-DataBlade

Das in dieser Arbeit entwickelte DataBlade für den Informix Dynamic Server/ Universal Data Option zur Verwaltung von multimedialen Dokumenten und Fragmenten des ZYX-Dokumentmodells trägt den Namen *PresFragments*. In diesem Anhang soll ein Überblick über den Aufbau und die Installation des DataBlade gegeben werden.

## D.1 Aufbau

Die Distribution des PresFragments-DataBlade erfolgt in einem mit GNU-Zip komprimierten TAR-File namens *fragments.tgz*[1]. In dieser Datei sind folgende Verzeichnisse zusammengefaßt:

- *fragments*: In diesem Verzeichnis existiert die Datei *zyx.dtd*, welche die DTD des XML-Formates für ZYX-Dokumente enthält (siehe auch Abbildung 6.15 auf Seite 92).

- *fragments/scripts*: Dieses Verzeichnis enthält die zur Registrierung bzw. Deregistrierung des PresFragments-DataBlade notwendigen SQL-Skripte. Im einzelnen sind dies die Dateien *object.sql*, *prepare.en_us.8859-1.sql*, *prepare.sql* und *test.sql*. Die wesentliche Datei ist hier *objects.sql*, da sie die Definitionen der zum DataBlade gehörenden Datenbankelemente enthält.

- *fragments/src*: Dieses Verzeichnis beherbergt die C-Quellen der serverbasierten Routinen des DataBlade. Diese sind in drei Dateien organisiert. Die Datei *PresFragment.h* enthält sämtliche Funktionsprototypen. Die Datei des Namens *exportXML.c* enthält die Implementierung des Exports nach XML, während die Implementierungen der restlichen Routinen in der Datei mit dem Namen *PresFragments.c* zu finden sind. Die Datei *UNIX.mak* stellt das Makefile für den GNU-C Compiler dar.

[1] Erhältlich unter: *http://www.informatik.uni-ulm.de/dbis/Cardio-OP/fragments.tgz*

- *fragments/src/solaris-sparc*: In diesem Verzeichnis werden die bei der Übersetzung des DataBlade entstehenden Object-Files und die Shared Library namens *PresFragments.bld* abgelegt.
- *fragments/sql*: Dieses Verzeichnis enthält diverse SQL-Skripte, u.a. einen Test für die Installation in der Datei *demo.sql*.

## D.2 Installation

Das PresFragments-DataBlade ist auf einer SUN ULTRASPARC unter Solaris 2.6 entwickelt worden. Voraussetzung für die Installation des DataBlades ist die Verfügbarkeit des Informix Dynamic Server/ Universal Data Option und des GNU-C Compiler auf solch einem System. Der IDS/UD muß fertig installiert sein. Insbesonders sind die für den Betrieb des Datenbank-Servers notwendigen Umgebungsvariablen korrekt zu setzen. Näheres hierzu findet sich in der IDS/UD-Dokumentation. Zur Installation auf diesem System sind folgende Schritte durchzuführen:

1. **Entpacken der Distribution:**
   Die Befehle `gzip -d fragments.tgz` und `tar xf fragments.tar` entpacken die Distribution und legen im aktuellen Verzeichnis das Verzeichnis *fragments* mit den oben beschriebenen Unterverzeichnissen an.
2. **Anlegen des DataBlade-Verzeichnisses:**
   Im Verzeichnis *$INFORMIXDIR/extend* ist ein Unterverzeichnis mit dem Namen *PresFragments* anzulegen. Dies geschieht mit:
   `mkdir $INFORMIXDIR/extend/PresFragments`
3. **Kopieren der Installationsskripte:**
   Wechseln Sie in das Verzeichnis *fragments/scripts*. Kopieren Sie alle dort befindlichen Skripte in das DataBlade-Verzeichnis. Dies geschieht mit der Anweisung: `cp * $INFORMIXDIR/extend/PresFragments`
4. **Übersetzen des DataBlade:**
   Wechseln Sie in das Verzeichnis *fragments/src*. Führen Sie dort aus:
   `make -f UNIX.mak TARGET=$INFORMIXDIR/incl/dbdk/makeinc1.sol`
   Es erscheinen einige Warnungen am Bildschirm. Lassen Sie sich von diesen nicht beirren. Es wird im Verzeichnis *fragments/src/solaris-sparc* die Shared Library *PresFragments.bld* erzeugt.
5. **Kopieren der Shared Library:**
   Wechseln Sie in das Verzeichnis *fragments/src/solaris-sparc*. Kopieren Sie die Shared Library *PresFragments.bld* in das DataBlade-Verzeichnis:
   `cp PresFragment.bld $INFORMIXDIR/extend/PresFragments`
6. **Anlegen einer Datenbank:**
   Starten Sie das Programm `dbaccess`. Legen Sie als Benutzer *informix* unter dem Menüpunkt „Database" eine neue Datenbank mit dem Namen *zyx* an.
7. **Registration des Datablade:**
   Starten Sie das Program `blademgr`. Geben Sie dort am Prompt folgende Zeile ein: `register PresFragments zyx`.
8. **Durchführen eines Tests:**
   Wechseln Sie in das Verzeichnis *fragments/sql*. Starten Sie von dort erneut

`dbaccess`. Wählen Sie den Menüpunkt ,,Query-language" und selektieren Sie die Datenbank *zyx*. Wählen Sie ,,Choose" und selektieren Sie die Datei mit Namen *demo.sql*. Wählen Sie danach „Run". Der IDS/UD liefert zu vielen Anweisungen einen Ergebniswert zurück, der jeweils mit der Return-Taste quittiert werden muß. Es dürfen keine Fehlermeldungen auftreten.

Die Datei *demo.sql* (siehe Abbildung D.1 auf Seite 138) legt mehrere Spezifikationsbäume in der Datenbank an. Man kann zum Beispiel den Spezifikationsbaum mit der ID 20 in das XML-Format exportieren. Wählen Sie dazu ,,New" und geben sie ein: `execute function exportXML(20, "set{}");` Wählen Sie ,,Run". Als Ergebnis sollte der Pfad der angelegten XML-Datei zurückgeliefert werden (*/tmp/spec20.xml*). Die Datei kann man mit einem XML-Browser betrachten.

```
1   -- Create atomic media objects.
2
3   execute function createAtomic(NULL, "SET{ROW('bandwith','high'),
4      ROW('quality','high')}", "sound1.wav","audio","wav");
5   execute function createAtomic(NULL, "SET{}", "sound2.wav","audio","wav");
6   execute function createAtomic(NULL, "SET{ROW('quality','low')}",
7      "sound3.wav","audio","wav");
8
9   -- Create switch operatorelement with 3 choices (including default).
10
11  execute function createSwitch(2,
12  "LIST{SET{ROW('bandwidth','high')},SET{ROW('bandwidth','low')}}");
13
14  -- Create instances of above elements.
15
16  execute function createInstance(1); -- sound1.wav, InstanceID = 8
17  execute function createInstance(2); -- sound2.wav, InstanceID = 9
18  execute function createInstance(3); -- sound3.wav, InstanceID = 10
19  execute function createInstance(4); -- switch, InstanceID = 11
20
21  -- Create 1st specification with ID = 12.
22
23  execute function createSpec();
24  execute procedure addInstance(12, 11); -- switch
25  execute procedure addInstance(12, 8);  -- sound1.wav
26  execute procedure bindVariable(12, 11, 3, 8);
27
28  -- Encapsulate specification in complex media object no. 1. This
29  -- complex media object exports two variables of the switch-element.
30
31  execute function createComplex(NULL, "SET{}", 12);
32
33  -- Create 2nd specification with ID = 15.
34
35  execute function createInstance(13); -- complex media object no.1,Instance ID = 14
36  execute function createSpec();
37  execute procedure addInstance(15, 14); -- complex media object no. 1
38  execute procedure addInstance(15, 9);  -- sound2.wav
39  execute procedure bindVariable(15, 14, 2, 9);
40
41  -- Encapsulate specification with ID = 15 in complex media object no.2.
42  -- This exports one variable of the switch-element.
43
44  execute function createComplex(NULL, "SET{}", 15);
45
46  -- Create specification with ID = 18.
47
48  execute function createInstance(16); -- complex media object no. 2,InstanceID = 17
49  execute function createSpec();
50  execute procedure addInstance(18, 17); -- complex media object no. 2
51  execute procedure addInstance(18, 10); -- sound3.wav
52  execute procedure bindVariable(18, 17, 1, 10);
53
54  -- Encapsulate it in complex media object no. 3.
55
56  execute function createComplex(NULL, "SET{ROW('theme','sounds')}", 18);
57
58  -- Create specification with ID = 20 for a test of the query-operatorelement.
59
60  execute function createSpec();
61
62  -- Create a query-operatorelement. It should find the complex media object
63  -- encapsulating the specification with ID = 20.
64
65  execute function createQuery("LIST{SET{ROW('theme','sounds')}}");
66  execute function createInstance(21); -- query, InstanceID = 22
67  execute procedure addInstance(20,22);
68
69  -- Create seq-Operator element.
70
71  execute function createSeq(2);
72  execute function createInstance(23); -- seq, InstanceID = 26
73  execute procedure addInstance(20,26);
74
75  -- Create another query-operatorelement. It should find the atomic
76  -- media object encapsulating sound1.wav.
77
78  execute function createQuery("LIST{SET{ROW('quality','high')}}");
79  execute function createInstance(27); -- query, InstanceID = 28
80  execute procedure addInstance(20,28);
81
82  -- Bind both query-operatorelements to the seq-operatorelement.
83
84  execute procedure bindVariable(20, 26, 1, 28);
85  execute procedure bindVariable(20, 26, 2, 22);
```

Abbildung D.1: Beispiel zur Verwendung des Datablade

# Literaturverzeichnis


[ABC+97] Adler, S., A. Berglund, J. Clark et al.: *A Proposal for XSL.* W3C, URL: http://www.w3.org/TR/NOTE-XSL-970910, August 1997.

[BD97] Bray, T. und S. DeRose: *Extensible Markup Language (XML): Part 2. Linking – W3C Working Draft July-31-97.* W3C, URL: http://www.w3.org/TR/WD-xml-link-970731, Juli 1997.

[BM98a] Behme, H. und S. Mintert: *Klammern gehört zum Handwerk – DSSSL: XML-Dokumente für das Web formatieren.* iX, (3), März 1998.

[BM98b] Behme, H. und S. Mintert: *XML in der Praxis.* Addison-Wesley, Bonn, 1998.

[Bol94] Boles, D.: *Das IMRA-Modell – Integration von Interaktionen in das Autorenwerkzeug FMAD.* Diplomarbeit, Universität Oldenburg, Fachbereich für Informatik, 1994.

[BPSM98] Bray, T., J. Paoli und C.M. Sperberg-McQueen: *Extensible Markup Language (XML) 1.0 – W3C Recommendation 10-February-1998.* W3C, URL: http://www.w3.org/TR/1998/REC-xml-19980210, Februar 1998.

[DD94] DeRose, S. und D.G. Durand: *Making Hypermedia Work: A User's Guide to HyTime.* Kluwer Academic Publishers, Dordrecht, 1994.

[DGHL91] Duce, Gomes, Hopgood und Lee (Herausgeber): *User Interface Management and Design.* Springer-Verlag, Berlin, 1991.

[DK95] Duda, A. und C. Keramane: *Structured Temporal Composition of Multimedia Data.* In: *Proc. IEEE International Workshop on Multimedia- Database-Management Systems*, Blue Mountain Lake, August 1995.

[Gol90] Goldfarb, C.F.: *The SGML Handbook.* Clarendon Press, Oxford, 1990.

[Gra92] Graf, W. H.: *Constraint-Based Graphical Layout of Multimodal Presentations.* In: Catarci, T., M.F. Costabile und S. Levialdi (Herausgeber): *Advanced Visual Interfaces, Proceedings of the International Workshop AVI '92.* World Scientific Press, Singapore, 1992.

[Gra95] Graf, W. H.: *The Constraint-Based Layout Framework LayLab and Its Applications.* In: *Proceedings of the ACM Workshop on Effective Abstractions in Multimedia*, San Francisco, November 1995.

[Gra97] Graf, W. H.: *Intent-Based Layout in Multimedia Communication.* In: Lee, J. (Herausgeber): *Intelligence and Multimodality in Multimedia Interfaces: Research & Applications.* AAAI Press, Menlo Park, 1997.

[Hen94] HENNINGER, S.: *Using Iterative Refinement to Find Reusable Software.* IEEE Software, 11(5), September 1994.

[Hof96] HOFMANN, P.: *MHEG-5 and MHEG-6: Multimedia Standards for Minimal Resource Systems.* Technischer Bericht, Technische Universität Berlin, April 1996.

[Inf97a] INFORMIX: *DataBlade API Programmer's Manual.* Informix Press, Menlo Park, 1997.

[Inf97b] INFORMIX: *DataBlade Developers Kit User's Guide.* Informix Press, Menlo Park, 1997.

[ISO95] ISO/IEC JTC1/SC29/WG12: *Information Technology – Coding of Multimedia and Hypermedia Information – Part 5: Support for Base-Level Interactive Applications.* ISO/IEC IS 13522-5, 1995.

[ISO96] ISO/IEC JTC1/SC29/WG12: *Information Technology – Coding of Multimedia and Hypermedia Information – Part 6: Support for Enhanced Interactive Applications.* ISO/IEC IS 13522-6, 1996.

[JR95] JOSEPH, R. und J. ROSENGREN: *MHEG-5: An Overview.* Technischer Bericht, URL: http://www.fokus.gmd.de/ovma/mug/-archives/doc/mheg-reader/rd1206.html, Dezember 1995.

[KD96] KERAMANE, C. und A. DUDA: *Interval Expressions – a Functional Model for Interactive Dynamic Multimedia Presentations.* In: *Proc. IEEE International Conference on Multimedia Computing and Systems*, Hiroshima, 1996.

[Kru92] KRUEGER, C.W.: *Software Reuse.* ACM Computing Surveys, 24(2), Juni 1992.

[MBE95] MEYER-BOUDNIK, T. und W. EFFELSBERG: *MHEG Explained.* IEEE Multimedia, 2(1), Spring 1995.

[NKN91] NEWCOMB, S.R., N.A. KIPP und V.T. NEWCOMB: *"HyTime" – The Hypermedia/Time-Based Document Structuring Language.* Communications of the ACM, 34(11), November 1991.

[PBD+98] P.HOSCHKA, S. BUGAJ, D.BULTERMAN et al.: *Synchronized Multimedia Integration Language – W3C Working Draft 2-February-98.* W3C, URL: http://www.w3.org/TR/1998/WD-smil-0202, Februar 1998.

[Pri89] PRIETO-DIAZ, R.: *Classification of Reusable Modules.* In: BIGGERSTAFF, T.J. und A.J. PERLIS (Herausgeber): *Software Reusability Volume I – Concepts and Models.* Addison-Wesley, Reading, 1989.

[RvOB97] RUTLEDGE, L., J. VAN OSSENBRUGGEN und D.C.A. BULTERMAN: *A Framework for Generating Adaptable Hypermedia Documents.* In: *Proc. ACM Multimedia Conference*, Seattle, November 1997.

[SN95] STEINMETZ, R. und K. NAHRSTEDT: *Multimedia: Computing, Communications and Applications.* Prentice Hall, Upper Saddle River, 1995.

[Som96] SOMMERVILLE, I.: *Software Engineering, 5th Edition.* Addison-Wesley, Wokingham, 1996.

[STS97] SAAKE, G., C. TÜRKER und I. SCHMITT: *Objektdatenbanken.* International Thomson Publishing, Bonn, 1997.

[Tol96] TOLKSDORF, R.: *Die Sprache des Web: HTML 3.* dpunkt, Heidelberg, 1996.

[Weg87] WEGNER, P.: *Varieties of Reusability.* In: FREEMAN, P. (Herausgeber): *Tutorial: Software Reusability.* IEEE Computer Society Press, Washington, 1987.

[Wei97] WEITHÖNER, T.: *Adaption in Multimedia-Informationssystemen.* Seminararbeit, Universität Ulm, Fakultät für Informatik, Abteilung Datenbanken und Informationssysteme, Wintersemester 1997.

[WRW94] WIRAG, S., K. ROTHERMEL und T. WAHL: *Modelling Interaction With HyTime.* Fakultätsbericht, Universität Stuttgart, Fakultät für Informatik, 1994.